Width-Based Dynamic Programming for Automated Theorem Proving in Graph Theory

Sam Urmian

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2026



UNIVERSITY OF BERGEN

Width-Based Dynamic Programming for Automated Theorem Proving in Graph Theory

Sam Urmian



Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 24.08.2026

© Copyright Sam Urmian

The material in this publication is covered by the provisions of the Copyright Act.

Year: 2026
Title: Width-Based Dynamic Programming for Automated Theorem Proving in Graph Theory

Name: Sam Urmian
Print: Skipnes AS / University of Bergen

Scientific environment

This research was carried out at the University of Bergen (UiB), Department of Informatics, under the supervision of Mateus de Oliveira Oliveira. The work was funded by the Research Council of Norway (RCN) through the project *AUTOPROVING: Automated Theorem Proving from the Mindset of Parameterized Complexity Theory* (Project no. 288761). I gratefully acknowledge the RCN for this support.

Computational experiments were supported by the Sigma2 infrastructure (Project no. NN9535K).

The TreeWidzard engine used in this thesis is described in the system paper [de Oliveira Oliveira and Urmian, 2026b], and the scripts and data used to produce the experiments reported in this thesis are available at `mlgorithm/phd-thesis-experiments` [Urmian, 2025].

Acknowledgements

This dissertation was completed at the Department of Informatics, University of Bergen. I am grateful to the many people whose support made this work possible.

My deepest thanks go to my supervisor, Mateus de Oliveira Oliveira, for his guidance, patience, and encouragement throughout these years. His careful feedback and mathematical insight shaped both the technical development of this thesis and my growth as a researcher.

I am also deeply grateful to the Department of Informatics, and in particular to the Algorithms group, for providing such a stimulating and welcoming research environment. I thank the professors of the group for their inspiring lectures, generous advice, and the high standard of scholarship they uphold, which has been a constant source of motivation. To my excellent colleagues — thank you for the countless discussions, the shared challenges, and the friendship that made these years so rewarding. I also thank the administrative and technical staff at the department for their reliable practical support.

I feel truly fortunate to be surrounded by so many wonderful friends — far too many to name here, though each of you knows the role you have played. Thank you for the laughter, the long conversations, and the steady company that made the difficult moments lighter and the good ones brighter.

Above all, I am profoundly grateful to my parents, whose love, sacrifices, and unwavering belief in me have been the foundation of everything I have achieved. To my family, thank you for your patience, encouragement, and constant support throughout this journey — this thesis would not exist without you.

Preface

I submit this dissertation in partial fulfillment of the requirements for the degree of Philosophiae Doctor (PhD) at the University of Bergen.

The thesis is presented as a monograph. It develops an ITD-centered framework for width-based reasoning on graphs: the representation layer is stated for general treelike decomposition classes, while the operational development, correctness instantiations, implementation, and validation focus on instructive tree decompositions and their bounded-width search workflows.

Author name update. I now publish under the name **Sam Urmian**. Earlier publication records and external repositories may list the former name **Farhad Vadiee**.

Parts of the results presented here were previously disseminated in joint publications with Mateus de Oliveira Oliveira (see the List of publications). This monograph consolidates the framework, unifies notation, expands proofs and system documentation, and integrates the results into a single end-to-end methodology.

Abstract in English

Many hard graph problems admit efficient dynamic-programming algorithms on bounded-width decompositions. This thesis develops a framework that lifts such algorithms into procedures for width-bounded automated theorem proving.

In the framework, graphs of bounded width are represented by treelike terms, and graph properties are specified by dynamic programming cores (DP-cores): explicit, modular dynamic programs evaluated bottom-up on those terms. For each fixed width bound k, a DP-core can either evaluate a property on a single input graph or determine whether the property holds for all graphs whose width is at most k.

The framework is modular: at the representation level, it accommodates several well-studied graph width measures. The operational and implementation development is carried out for treewidth and pathwidth, using instructive tree decompositions. As a quantitative application, the thesis proves that for several long-standing graph-theoretic conjectures, there exists an algorithm that, given k, decides in time double-exponential in a polynomial function of k whether the conjecture holds for all graphs of treewidth at most k. These bounds may be viewed as upper bounds on the size of proofs and disproofs within the class of graphs of treewidth at most k, and they significantly improve previously available theoretical bounds.

The thesis also studies two methods for making the framework more effective in practice. One is state canonization, which reduces the number of states explored while searching for counterexamples. The other is an early-pruning strategy for conjectures of the form $P_1 \to P_2$, where P_1 and P_2 are graph properties and P_1 is closed under subgraphs. The framework is then applied to coloring conjectures for triangle-free graphs. In this setting, it automatically verifies the triangle-free Reed bound on every bounded-width class we report, including all instances up to treewidth 3, the additional case $(\mathrm{tw}, \Delta) = (4, 2)$, and the published case of pathwidth at most 5 and maximum degree 3. The framework also produces fully automated counterexamples to invalid strengthenings of the conjecture. These results show that width-bounded automated theorem proving is a

promising approach to the study of graph-theoretic conjectures.

Abstract in Norwegian

Mange vanskelige grafproblemer tillater effektive dynamiske programmeringsalgoritmer på dekomposisjoner med begrenset bredde. Denne avhandlingen utvikler et rammeverk som løfter slike algoritmer til prosedyrer for breddebegrenset automatisert teorembevisning.

I rammeverket representeres grafer med begrenset bredde ved hjelp av trelignende termer, og grafegenskaper spesifiseres ved dynamiske programmeringskjerner (DP-kjerner): eksplisitte, modulære dynamiske programmer som evalueres nedenfra og opp på disse termene. For hver fast verdi av breddegrensen k kan en DP-kjerne enten evaluere en egenskap på en enkelt inngangsgraf eller avgjøre om egenskapen gjelder for alle grafer med bredde høyst k.

Rammeverket er modulært: på representasjonsnivå dekker det flere velstuderte breddeparametre for grafer. Den operasjonelle utviklingen og implementasjonen i avhandlingen er gjennomført for *treewidth* og *pathwidth* ved hjelp av *instructive tree decompositions*. Som en kvantitativ anvendelse viser avhandlingen at det, for flere langvarige grafteoretiske formodninger, finnes en algoritme som, gitt k, avgjør i tid som er dobbelteksponentiell i en polynomfunksjon av k, om formodningen gjelder for alle grafer med *treewidth* høyst k. Disse grensene kan tolkes som øvre grenser for størrelsen på bevis og motbevis innen klassen av grafer med *treewidth* høyst k, og de forbedrer tidligere tilgjengelige teoretiske grenser betydelig.

Avhandlingen studerer også to metoder for å gjøre rammeverket mer effektivt i praksis. Den ene er tilstandskanonisering, som reduserer antallet tilstander som utforskes under søk etter moteksempler. Den andre er en strategi for tidlig beskjæring for formodninger av formen $P_1 \to P_2$, der P_1 og P_2 er grafegenskaper og P_1 er lukket under delgrafer.

Rammeverket anvendes deretter på formodninger om fargelegging av trekantfrie grafer. I denne sammenhengen verifiserer det automatisk Reed-grensen for trekantfrie grafer på alle de rapporterte breddebegrensede klassene, inkludert alle instanser med *treewidth* høyst 3, det ekstra tilfellet $(\mathrm{tw}, \Delta) = (4, 2)$, og det publiserte tilfellet med *path-*

width høyst 5 og maksimalgrad 3. Rammeverket konstruerer også fullt automatiserte moteksempler til ugyldige styrkinger av formodningen. Disse resultatene viser at breddebegrenset automatisert teorembevisning er en lovende tilnærming til studiet av grafteoretiske formodninger.

List of publications

The following publications arise from the research reported in this thesis.

Author name update. I changed my legal name to **Sam Urmian**. Earlier publication records may list the former name **Farhad Vadiee**.

1. Mateus de Oliveira Oliveira and Farhad Vadiee. *From Width-Based Model Checking to Width-Based Automated Theorem Proving.* Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2023), 37(5):6297–6304, 2023. DOI: `10.1609/aaai.v37i5.25775`. Full version: arXiv:2205.10995.

2. Mateus de Oliveira Oliveira and Farhad Vadiee. *State Canonization and Early Pruning in Width-Based Automated Theorem Proving.* 9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024), LIPIcs, Vol. 299, Art. 33, pages 33:1–33:17, 2024. DOI: `10.4230/LIPIcs.FSCD.2024.33`. Full version: arXiv:2605.11025.

3. Mateus de Oliveira Oliveira and Sam Urmian. *TreeWidzard: An Engine for Width-Based Dynamic Programming and Automated Theorem Proving.* arXiv preprint arXiv:2605.09732, 2026. DOI: `10.48550/arXiv.2605.09732`. Full version: arXiv:2605.09732.

Other publications during the PhD

The following additional publication was produced during the PhD.

4. Fidel Ernesto Diaz Andino, Maria Kokkou, Mateus de Oliveira Oliveira, and Farhad Vadiee. *Unitary Branching Programs: Learnability and Lower Bounds.* In Proceedings of the 38th International Conference on Machine Learning (ICML 2021), Proceedings of Machine Learning Research (PMLR) 139:297–306, 2021.

Contents

Scientific environment i

Acknowledgements iii

Preface v

Abstract in English vii

Abstract in Norwegian ix

List of publications xi

I Introduction 1

1 Introduction 3

II Background 13

2 Background 15

2.1 Parameterized Complexity 16

2.2 Width Measures 19

2.2.1 Treewidth and Pathwidth 19

2.2.2 Clique-width 23
2.3 Terms and Tree Automata 24
2.3.1 Terms 24
2.3.2 Bottom-up finite tree automata 25
2.3.3 Standard facts 26

III Dynamic Programming Cores 27

3 Treelike Width Measures and Instructive Tree Decomposition 29

3.1 Treelike Decomposition Classes 29
3.2 Treelike Width Measures 30
3.3 Automaton Realizability and Complexity 31
3.4 Instructive Tree Decomposition 31
3.4.1 Characterizing Treewidth via Instructive Decompositions 39

4 Treelike Dynamic Programming Cores 47

4.1 Treelike DP-cores 48
4.1.1 Witness-level branching and witness-set propagation 49
4.2 Dynamization and Acceptance 50
4.3 Graph Property 52
4.4 Coherence 53
4.4.1 Graph Invariants via Coherent DP-cores 55
4.5 Complexity Measures 55
4.6 Polynomial DP-cores 58
4.7 DP-core evaluation on a fixed decomposition term 60

4.7.1 Model Checking and Invariant Computation 60

4.8 Canonical DP-core Catalog 66

4.8.1 c-Colorability 68

4.8.2 Catalog index (pointers) 70

IV Width-Based Automated Theorem Proving 73

5 Width-Based Automated Theorem Proving and Double-Exponential Upper Bounds 75

5.1 Inclusion Test 76

5.2 Dynamic Programming Refutation 77

5.3 Counterexample Size Bounds 87

5.4 Bounded-Size Inclusion Test 89

5.5 Combinators and Combinations 91

5.5.1 Combinators: Syntax and Semantics 91

5.5.2 Constructing Combined DP-cores 93

5.5.3 Algorithmic Implications 96

5.6 Applications to Classical Conjectures 98

6 Symmetry Breaking 103

6.1 Witness Actions and Relabeled Refutations 103

6.1.1 Witness Actions 104

6.1.2 Relabeled Refutations 114

6.2 Canonical Refutations 121

6.3 Summary 134

7 Premise Search **135**

7.1 Premise Search and Subgraph-Closed Properties 135

7.2 From Pairs to Triples: Premise-Based Refutation 137

7.3 Boolean Combinations of Graph Properties 139

7.4 Refutations with Combined Cores 139

7.5 The Premise Search Algorithm 141

7.6 Summary 146

8 Triangle-Free Reed Case Study **147**

8.1 Reed's Conjecture and the Triangle-Free Case 148

8.2 Results and Special Cases 150

8.3 Parameterized Version of Reed's Conjecture for Triangle-Free Graphs . . 152

8.4 DP-core Encoding of the Triangle-Free Case 154

8.4.1 Boolean Encoding via DP-cores 154

8.4.2 Applying Theorem 10 157

8.5 Premise Search and Symmetry Breaking 159

8.6 Experiments 160

8.7 Summary 163

V Correctness and Validation **165**

9 Systematic Correctness **167**

9.1 A Systematic Proof Framework 168

9.1.1 Dynamic semantics as the unique homomorphism 171

9.1.2 Denotational semantics and the local law 176

9.1.3 Master equivalence theorems 176
9.1.4 Equivariance and coherence 177
9.1.5 Bitlength and complexity 183
9.1.6 Local interpretation predicates and the DP proof rule 183
9.2 c-Colorability 185
9.2.1 Witness space, cleaning, and acceptance 186
9.2.2 Transitions as set-valued functions 188
9.2.3 Denotation and the local interpretation predicate 189
9.2.4 Local law (INT / LSC) 190
9.2.5 From local law to global correctness 191
9.2.6 Equivariance and acceptance invariance 192
9.2.7 Bitlength, multiplicity, and running time 193
9.3 Further Instantiations and Extensions 194
9.3.1 Guidance for New Cores 195

10 Empirical Validation of the Implemented DP-cores 197

10.1 Experiment setup and graph generation 197
10.2 Benchmarked properties and DP-cores 199
10.3 Execution environment and run setup 199
10.4 Agreement results 200
10.5 What the results establish about the implemented DP-cores 201

VI Conclusion and Future Directions 203

11 Conclusion and Future Directions 205

11.1 Limitations 206

11.2 Outlook 207

Bibliography **211**

Appendices: Reader's Guide **222**

A Notation and Conventions **223**

A.1 Standing Assumptions 223

A.2 Sets and Asymptotics 223

A.3 Graphs 224

A.4 Ranked Alphabets and Terms 224

A.5 Automata, ITD Semantics, and DP-cores 225

A.5.1 Automata and transitions 225

A.5.2 ITD semantics and boundaried graphs 225

A.5.3 DP-cores and dynamization 225

A.6 Overloaded Symbols 226

A.7 SAT Convention 229

B Problem Definitions **230**

B.1 Notation 230

B.2 Problems used as cores / SAT encodings 230

B.3 Additional standard problems referenced in the background 233

B.4 Additional properties used in automated theorem proving 235

C Clique-Width **237**

C.1 Definition and Properties 237

C.2 Clique-width as a Treelike Decomposition Class 240

D Supplementary DP-core Correctness Proofs 243

D.1 Witness spaces and bitlength bounds for the cores of Table 5.1 243

D.2 Connectedness (`Conn`) 248

D.2.1 Witness space, cleaning, and acceptance 249

D.2.2 Transitions 249

D.2.3 Denotation and local interpretation predicate 250

D.2.4 Local law and global correctness 251

D.2.5 Complexity summary 252

D.3 Minor Containment (`Minor(H)`) 253

D.3.1 Witness space, cleaning, and acceptance 253

D.3.2 Global objects, bag projection, and interpretation 253

D.3.3 Transitions 255

D.3.4 Join witness composition 255

D.3.5 Local law 258

D.3.6 Global correctness 260

D.3.7 Complexity summary 261

D.4 Maximum Degree ($\geq d$) 262

D.4.1 Witness space, cleaning, and acceptance 262

D.4.2 Transitions as set-valued functions 263

D.4.3 Denotation and the local interpretation predicate 263

D.4.4 Local law (INT / LSC) 263

D.4.5 From local law to global correctness 265

D.4.6 Equivariance and coherence 265
D.4.7 Bitlength, multiplicity, and running time 266
D.5 Clique of Size ℓ 266
D.5.1 Witness space, cleaning, and acceptance 266
D.5.2 Transitions as set-valued functions 267
D.5.3 Denotation and the local interpretation predicate 268
D.5.4 Local law (INT / LSC) 269
D.5.5 From local law to global correctness 272
D.5.6 Equivariance and coherence 272
D.5.7 Bitlength, multiplicity, and running time 272
D.6 Multi-Edge Detection 273
D.6.1 Witness space, cleaning, and acceptance 273
D.6.2 Transitions as set-valued functions 273
D.6.3 Denotation and the local interpretation predicate 273
D.6.4 Local law (INT / LSC) 274
D.6.5 From local law to global correctness 275
D.6.6 Equivariance and coherence 276
D.6.7 Bitlength, multiplicity, and running time 276
D.7 Perfect Matching 276
D.7.1 Witness space, cleaning, and acceptance 276
D.7.2 Transitions as set-valued functions 276
D.7.3 Denotation and the local interpretation predicate 277
D.7.4 Local law (INT / LSC) 277
D.7.5 From local law to global correctness 278

D.7.6 Equivariance and coherence 280
D.7.7 Bitlength, multiplicity, and running time 280
D.8 Hamiltonian Path: Endpoint Matchings (Involutions) 281
D.8.1 Witness space, cleaning, and acceptance 281
D.8.2 Transitions as set-valued functions 282
D.8.3 Denotation and the local interpretation predicate 287
D.8.4 Local law (INT / LSC) 289
D.8.5 From local law to global correctness 291
D.8.6 Equivariance and coherence 292
D.8.7 Bitlength, multiplicity, and running time 292
D.9 Hamiltonian Cycle: Endpoint Matchings 293
D.9.1 Witness space, cleaning, and acceptance 294
D.9.2 Transitions as set-valued functions 294
D.9.3 Denotation and the local interpretation predicate 295
D.9.4 Local law (INT / LSC) 296
D.9.5 From local law to global correctness 299
D.9.6 Equivariance and coherence 299
D.9.7 Bitlength, multiplicity, and running time 299
D.10 Feedback Vertex Set ($\leq t$) 300
D.10.1 Witness space, cleaning, and acceptance 300
D.10.2 Transitions as set-valued functions 300
D.10.3 Denotation and the local interpretation predicate 301
D.10.4 A join lemma for gluing forests along a bag 302
D.10.5 Local law (INT / LSC) 303

D.10.6 Global correctness 305
D.10.7 Equivariance and coherence 306
D.10.8 Bitlength, multiplicity, and running time 307
D.11 Supplementary Framework Notes 307
D.11.1 Abstract interpretation (optional) 307
D.11.2 Derivations and proof certificates 308
D.11.3 Dominance cleaning for optimization cores 310
D.12 Vertex Cover ($\leq t$): Worked LSC Proof 311
D.12.1 Witness space, cleaning, and acceptance 311
D.12.2 Transitions as set-valued functions 311
D.12.3 Denotation and the local interpretation predicate 312
D.12.4 Local law (INT / LSC) 313
D.12.5 From local law to global correctness 314
D.12.6 Equivariance and acceptance invariance 316
D.12.7 Bitlength, multiplicity, and running time 316
D.13 Maximum Independent Set 317
D.13.1 Witness space, cleaning, and acceptance 317
D.13.2 Transitions as set-valued functions 318
D.13.3 Denotation and the local interpretation predicate 318
D.13.4 Local law (INT / LSC) 318
D.13.5 From local law to global correctness 320
D.13.6 Equivariance and coherence 321
D.13.7 Bitlength, multiplicity, and running time 321
D.14 Vertex Counting 322

D.14.1 Witness space, cleaning, and acceptance 322

D.14.2 Transitions as set-valued functions 322

D.14.3 Denotation and the local interpretation predicate 322

D.14.4 Local law (INT / LSC) 323

D.14.5 From local law to global correctness 323

D.14.6 Equivariance and coherence 324

D.14.7 Bitlength, multiplicity, and running time 324

D.15 Boundary Myhill and Nerode equivalence and lower bounds 325

E SAT Encodings **329**

E.1 Boolean Logic and SAT Formulations 329

E.2 Encoding Non-CNF Constraints in SAT 329

E.3 Encoding Graph Problems into SAT 333

E.3.1 Vertex Cover 333

E.3.2 Graph Coloring 335

E.3.3 Independent Set 338

E.3.4 Hamiltonian Path 340

E.3.5 Hamiltonian Cycle 345

E.3.6 Clique 349

E.3.7 Perfect Matching 352

E.3.8 Feedback Vertex Set 353

E.3.9 Degree-at-least-d Vertices 356

E.3.10 Multiple Edge Detection 357

E.4 Summary 359

Part I

Introduction

Chapter 1

Introduction

Automated theorem proving has long been a central goal in logic and artificial intelligence [Harrison, 2009; Robinson and Voronkov, 2001]. Over the years, this goal has led to a rich body of methods for mechanized reasoning, including resolution-based proof systems, proof-planning techniques, mature first-order provers such as SPASS, E, and Vampire, and finite-model search tools such as Mace4 [Bundy, 1988; Davis and Putnam, 1960; Fitting, 1996; Kovács and Voronkov, 2013; McCune, 2003; Robinson, 1965; Schulz, 2002; Weidenbach et al., 1999]. These developments show that large parts of formal reasoning can be automated in practice.

At the same time, complete automation is impossible in full generality. Already for first-order logic, the classical undecidability results of Church and Turing show that there is no algorithm that decides the validity of arbitrary formulas [Church, 1936; Turing, 1937]. Over finite structures the obstruction is even stronger: by Trakhtenbrot's theorem, finite satisfiability is undecidable and finite validity is not recursively enumerable for relational signatures with at least one binary relation symbol [Ebbinghaus and Flum, 1995; Fagin, 1993; Kolaitis, 2007; Libkin, 2004]. Since finite graphs are finite relational structures over a binary adjacency relation, these barriers apply directly to graph-theoretic reasoning. Moreover, undecidability persists even on highly structured graph families: Garfunkel and Shank proved hereditary undecidability for finite planar graphs and for finite planar cubic graphs [Garfunkel and Shank, 1971, 1972]. Fully general automated theorem proving over graphs is therefore impossible already at a basic first-order level.

Graph theory is nevertheless a particularly compelling setting for automated reasoning. Many graph-theoretic conjectures are universal claims, so a single finite counterexample suffices to refute them. An automatic method that can systematically test such statements would therefore be highly valuable. This motivation is reinforced by a

substantial tradition of computer-assisted reasoning in graph theory. Cvetković and Pevac developed a man-machine theorem-proving system for graph theory [Cvetković and Pevac, 1988], while Epstein's GT system conjectured and proved relations among graph properties from recursive descriptions of graph classes [Epstein, 1987, 1988]. Graffiti shifted the emphasis toward automatic conjecture generation by testing inequalities between graph invariants on databases of known graphs [Fajtlowicz, 1988], and later work used intensive computation both to disprove Graffiti conjectures and to prove some of them [Brewster et al., 1995]. AutoGraphiX continued this line of computer-assisted graph theory and stimulated a broader literature on automated conjecturing in extremal and spectral graph theory [Aouchiche and Hansen, 2010; Aouchiche et al., 2005; Jooken, 2025]. Taken together, these works show that even when full automation is impossible, computer-assisted methods can play a meaningful role in graph-theoretic research.

The natural question is therefore whether useful automation can nevertheless be achieved on restricted graph classes. Existing work suggests several routes. On the logic side, Courcelle's theorem yields an algorithm for deciding monadic second-order properties on bounded-treewidth graphs [Courcelle, 1990]. Later results extended first-order model checking to broader sparse classes, including locally tree-decomposable classes and nowhere dense classes [Demaine and Hajiaghayi, 2008; Flum and Grohe, 2007; Frick and Grohe, 2001; Grohe, 2014; Grohe et al., 2017; Kreutzer, 2008]. A complementary route is exhaustive but isomorph-free exploration inside restricted families, supported by tools such as nauty/Traces and plantri [Brinkmann and McKay, 2007; McKay and Piperno, 2014]. These lines of work do not yield unrestricted automated theorem proving, but they show that structural restrictions can make the automatic testing of graph-theoretic conjectures feasible. This observation leads naturally to the parameterized-complexity viewpoint, in which graph classes are organized by width measures that quantify their structural complexity.

This restricted-class perspective is mathematically valuable in its own right. Before attacking a conjecture in full generality, mathematicians often test it on simpler graph classes to search for counterexamples, discover missing hypotheses, and isolate the structural core of the problem. An automated method that reasons effectively over special graph classes can therefore be a serious research tool even when unrestricted theorem proving is impossible.

The parameterized-complexity viewpoint offers a systematic way to study hard problems on restricted graph classes. Rather than measuring complexity only in terms of input size, parameterized complexity analyzes how the difficulty of a problem depends on auxiliary parameters that capture structural features of the input [Abrahamson et al., 1989; Cygan et al., 2015; Downey, 2012; Downey and Fellows, 1995a,b, 1999, 2013; Fellows, 2002; Flum

and Grohe, 2006; Niedermeier, 2006]. In graph algorithms, width measures are among the most useful such parameters. Each width measure w (a generic letter we use in this introduction; in the rest of the thesis we specialize to treewidth tw and pathwidth pw) gives rise to a nested hierarchy of graph classes

$$\mathcal{G}_0^w \subseteq \mathcal{G}_1^w \subseteq \mathcal{G}_2^w \subseteq \cdots, \qquad \mathcal{G}_k^w := \{G \mid w(G) \leq k\},$$

so that fixing k isolates an infinite but structurally controlled slice of graph space. Among the many width parameters studied in the literature, treewidth and pathwidth are the most relevant for the present thesis. They measure how close a graph is to a tree or to a path, respectively, and they lie at the center of a large body of structural and algorithmic work [Bodlaender, 1993, 1996, 1998, 2005; Bodlaender and Kloks, 1996; Bodlaender and Koster, 2010; Cygan et al., 2015].

The treewidth hierarchy already gives useful intuition. Graphs of treewidth at most 1 are exactly forests. Graphs of treewidth at most 2 are the partial 2-trees; this class properly extends forests and includes important families such as series-parallel graphs and outerplanar graphs [Bodlaender, 1998]. Graphs of treewidth at most 3 are the partial 3-trees, a class that already contains standard nontrivial examples such as Halin graphs [Arnborg and Proskurowski, 1989; Arnborg et al., 1987, 1990; Bodlaender, 1998].

More broadly, many natural graph classes have bounded treewidth: every k-outerplanar graph has treewidth at most $3k - 1$ [Bodlaender, 1988, 1998], every minor-closed class that excludes a fixed planar graph as a minor has bounded treewidth [Robertson and Seymour, 1986a, 1991; Robertson et al., 1994], and several graph families arising from structured programs also have small treewidth [Burgstaller et al., 2004; Gustedt et al., 2002; Krause et al., 2020; Thorup, 1998]. From an algorithmic perspective, treewidth also comes with a mature toolbox for constructing and approximating decompositions, which is one reason it remains the default structural parameter in parameterized graph algorithms [Bodlaender, 1996; Bodlaender and Kloks, 1996; Bodlaender et al., 2016].

Pathwidth is more restrictive, because it requires the underlying decomposition structure to be a path rather than an arbitrary tree. This additional linearity makes pathwidth especially natural when the relevant combinatorics has an essentially sequential character. At the bottom of the hierarchy, graphs of pathwidth at most 1 are exactly disjoint unions of caterpillars. The class of graphs of pathwidth at most 2 is already much richer: the class admits finite obstruction descriptions and explicit structural characterizations in terms of a small collection of basic building blocks [Barat and Hajnal, 2001; Barát et al., 2012; Kinnersley and Langston, 1994; Takahashi et al., 1994].

As with treewidth, one also obtains broad boundedness results at the class level:

excluding a fixed forest as a minor forces bounded pathwidth [Bienstock et al., 1991; Diestel, 1995; Robertson and Seymour, 1983; Seymour, 2023], and structured control-flow graphs provide a natural modern source of bounded-pathwidth graph classes [Conrado et al., 2023]. For these reasons, treewidth and pathwidth provide the most natural starting point for the width-bounded automated-theorem-proving framework developed in this thesis.

Beyond treewidth and pathwidth, parameterized complexity has developed width measures that remain informative on denser graph classes. The most prominent example is clique-width [Courcelle et al., 1993]: unlike treewidth, it can remain bounded on graph classes that are not sparse, such as cographs and distance-hereditary graphs [Corneil and Rotics, 2005; Courcelle and Olariu, 2000; Oum, 2008]. Rank-width [Oum, 2008, 2017] provides a closely related cut-based perspective and is equivalent to clique-width up to a fixed function [Oum, 2008; Oum and Seymour, 2006]. More recent parameters such as Boolean-width [Bui-Xuan et al., 2011] and mim-width [Brettell et al., 2022; Vatshelle, 2012] refine this viewpoint by measuring neighborhood complexity across cuts, and they have proved useful for a variety of dense and geometrically structured graph classes [Belmonte and Vatshelle, 2013; Bui-Xuan et al., 2011; Golovach et al., 2018; Jaffke et al., 2019; Kang et al., 2017; Vatshelle, 2012].

These parameters show that the width-based perspective is not limited to sparse graphs. Nevertheless, treewidth and pathwidth provide the most natural starting point for the present thesis, both because their structural meaning is especially transparent and because they are supported by a particularly rich dynamic-programming toolbox.

Having introduced width measures as a way of organizing graph classes by structural complexity, we can now formulate the central question of the thesis more precisely. The goal is not unrestricted automated theorem proving over all graphs, but automated reasoning over a structurally controlled graph class. Fix a width measure w. For each $k \in \mathbb{N}$, let

$$\mathcal{G}_k^w := \{G \mid w(G) \leq k\}$$

denote the class of graphs whose w-width is at most k. For the purposes of this thesis, a graph property is simply a set of graphs, and we write $G \in P$ to mean that the graph G satisfies the property P. The main question of the thesis is whether the restriction to $\mathcal{G}_k^w$ is strong enough to make automated theorem proving possible in a systematic and generic way.

Problem 1 (Width-bounded automated theorem proving). Fix a width measure w. Given a graph property P and a width bound $k \in \mathbb{N}$, determine whether

$$\mathcal{G}_k^w \subseteq P.$$

Equivalently, determine whether there exists a graph

$$G \in \mathcal{G}_k^w \setminus P.$$

This formulation makes explicit the sense in which the thesis is parameterized. The parameter is the structural width bound k, not the number of vertices. Consequently, the problem concerns an infinite graph class, but one restricted by a meaningful structural condition. The central question is whether this restriction can be exploited to obtain a systematic and reusable approach to automated theorem proving on graphs.

A common and particularly important special case of width-bounded automated theorem proving arises from graph-theoretic conjectures, which are often naturally expressed as implications between two graph properties. If P_1 denotes the premise of the conjecture and P_2 its conclusion, then the corresponding width-bounded theorem-proving task is to determine whether

$$\mathcal{G}_k^w \cap P_1 \subseteq P_2.$$

Equivalently, one asks whether there exists a graph

$$G \in \mathcal{G}_k^w \cap P_1 \setminus P_2.$$

In this form, automated theorem proving becomes an automated search for bounded-width counterexamples to the conjecture.

This viewpoint differs fundamentally from other forms of bounded-search automated reasoning, such as exhaustive search over graphs of bounded order or SAT/SMT solving instantiated over a fixed n-vertex universe. Such methods are complete only up to the chosen size bound, and therefore provide no guarantees for graphs with more than n vertices. By contrast, width-bounded automated theorem proving targets the entire structural slice $\mathcal{G}_k^w$, and therefore concerns an infinite graph class defined by a width bound rather than a finite collection of small instances. This distinction is one of the central motivations for the framework developed in this thesis.

The approach taken in this thesis is to parameterize automated theorem proving by structural width and to exploit the same decomposition machinery that underlies width-based graph algorithms. The key idea is to separate the representation of bounded-width graph classes from the specification of graph properties.

In the settings considered here, graphs of width at most k are encoded by validated terms over a finite constructor signature describing k-instructive tree decompositions. For each fixed k, the set of validated terms forms a regular tree language [Comon

et al., 2008; Doner, 1970; Thatcher and Wright, 1968], which makes the surrounding width-bounded graph class amenable to automata-theoretic reasoning. Graph properties are correspondingly represented by dynamic programming cores (DP-cores): explicit bottom-up dynamic programs whose witness sets, local transitions, and root conditions are part of the formal specification [Baste, 2024; Baste et al., 2022; de Oliveira Oliveira and Vadiee, 2023, 2024].

Once the representation layer and the property layer are separated, width-bounded automated theorem proving becomes a language-inclusion problem over validated terms [Courcelle, 1990; Courcelle and Engelfriet, 2012]. Let $L^{\mathrm{val}}_{w,k}$ denote the language of validated terms of width at most k, and let L_P denote the set of validated terms whose represented graph belongs to the property P. The theorem-proving task is then to decide whether

$$L^{\mathrm{val}}_{w,k} \subseteq L_P.$$

Equivalently, one searches for a term $\tau \in L^{\mathrm{val}}_{w,k} \setminus L_P$, in which case the represented graph $G(\tau)$ is a concrete counterexample of width at most k.

This reformulation lifts the same bottom-up machinery that evaluates a property on a single bounded-width graph to exhaustive reasoning over an entire width slice. The resulting completeness is structural rather than size-based: when the induced state space is finite, the absence of a counterexample certifies validity on the whole class $\mathcal{G}^w_k$, not merely on graphs up to some prescribed number of vertices [de Oliveira Oliveira and Vadiee, 2023, 2024].

This viewpoint yields explicit fixed-width guarantees for several classical graph-theoretic conjectures. In particular, for the c-th case of Hadwiger's conjecture, with c fixed, the conjecture can be tested on graphs of treewidth at most k in time $2^{2^{O(k \log k + c^2)}}$. Likewise, Tutte's 5-flow, 4-flow, and 3-flow conjectures, as well as Barnette's conjecture, can be tested on graphs of treewidth at most k in time $2^{2^{O(k \log k)}}$ [Barnette, 1969; de Oliveira Oliveira and Vadiee, 2023; Hadwiger, 1943; Tutte, 1954, 1966; Zhang, 1997]. These bounds should be read as fixed-width completeness guarantees: they show that broad and mathematically natural conjecture families can be decided uniformly on every width-k slice, even though the unrestricted problem remains out of reach.

It is worth noting that the same general objective could also be pursued through the logic-first machinery surrounding Courcelle's theorem. Courcelle's original work shows that MSO-definable properties on bounded-width graph classes can be compiled into tree automata [Courcelle, 1990], and this perspective is closely related to Seese's results on decidable monadic theories and bounded-width structure [Seese, 1991], as well as to earlier work connecting logic and tree decompositions [Arnborg et al., 1991; Borie et al., 1992].

More recent work on fly-automata makes this viewpoint significantly more operational by computing transitions on demand rather than materializing very large transition tables [Courcelle and Durand, 2011, 2012, 2016], and practical implementations of MSO model checking on tree-decomposable graphs are also available [Kneis et al., 2009; Langer et al., 2014].

However, the dependence on the logical specification can still be very severe: standard MSO-to-automaton constructions may be non-elementary in the formula size, and more generally the formula dependence in FO/MSO model checking is known to face non-elementary barriers [Frick and Grohe, 2004]. For graph-theoretic conjectures, whose natural formulations often combine minor conditions, connectivity conditions, coloring bounds, and Hamiltonicity-type constraints, this specification blow-up can quickly dominate. The approach of this thesis therefore retains the regular-language viewpoint of the Courcelle tradition, but replaces formula compilation by explicit DP-core constructions.

A second central question of this thesis is whether width-based automated theorem proving can be made practical. This is far from automatic. Even when the relevant graph properties admit dynamic programming algorithms of the kind routinely used in parameterized complexity, the exhaustive exploration required by width-based ATP can generate an enormous number of intermediate states.

At the same time, this setting offers an important opportunity: dynamic programming over tree decompositions has become one of the main tools for solving combinatorial problems on bounded-width graph classes, and there is now a large literature explaining how to design such algorithms efficiently [Arnborg et al., 1987; Bannach and Berndt, 2018; Bodlaender and Kloks, 1996; Bodlaender et al., 2015; Cygan et al., 2015; Kloks, 1994]. One of the main strengths of the present framework is that many of these decomposition-based algorithms can be translated, often with relatively little conceptual overhead, into the DP-core language. In this translation, the familiar table of a tree-decomposition algorithm becomes a witness domain, local update rules become constructor-wise operations, and the root condition becomes a finality predicate. In this sense, width-based ATP does not discard the algorithmic toolbox developed for tree decompositions; rather, it turns that toolbox into a reusable specification language for automated reasoning.

This thesis adopts an explicitly algebraic version of the dynamic-programming viewpoint. Rather than working directly with ad hoc tables attached to nice tree decompositions, graph properties are represented by algebraic DP-cores operating on validated k-instructive tree decompositions. This places the dynamic program itself at the center of the formalism: witness sets, constructor transitions, cleaning operations, and root

acceptance are all explicit objects. The resulting perspective is closely aligned with recent work that treats dynamic programming as a compositional and reusable object rather than as a collection of isolated problem-specific routines [Baste, 2024; Baste et al., 2022].

The present thesis develops this line in a direction tailored to graph-theoretic theorem proving. Once graph classes are represented by validated term languages, the same DP-core interface supports model checking on a single decomposition, Boolean combinations of properties, invariant computation, and exhaustive counterexample search over width-bounded graph classes [de Oliveira Oliveira and Vadiee, 2023, 2024].

To make this approach practical, the thesis develops search-space reductions that significantly decrease the number of states that must be explored. The first is a general-purpose symmetry-breaking technique based on a local notion of witness isomorphism, which should not be confused with graph isomorphism: states are canonized modulo relabelings of the active bag, so that two states differing only in temporary bag names are identified before the search proceeds. The second is an early-pruning technique for conjectures of the form $A \Rightarrow B$ when the premise A is subgraph-closed. In that setting, once a partial construction violates the premise, no later extension can restore it, and the corresponding branch can be discarded immediately.

These two reductions address different sources of state explosion. Canonization removes redundancy caused by bag-label symmetries, whereas premise pruning avoids extending partial constructions that can no longer yield a counterexample. Their combination is therefore substantially more powerful than either technique in isolation [de Oliveira Oliveira and Vadiee, 2024].

To evaluate the framework experimentally, we implemented it in the tool TreeWidzard [de Oliveira Oliveira and Urmian, 2026b] and used it to investigate the triangle-free version of Reed's conjecture, a long-standing conjecture in graph coloring [de Oliveira Oliveira and Urmian, 2026a; de Oliveira Oliveira and Vadiee, 2024; Reed, 1998]. The tool verifies the triangle-free Reed bound fully automatically on the bounded-width classes considered, including all treewidth instances up to treewidth 3 as well as the case $(\mathrm{tw}, \Delta) = (4, 2)$ and, in the largest published pathwidth run, graphs of pathwidth at most 5 and maximum degree 3; it also produces explicit counterexamples to natural false strengthenings of the statement [de Oliveira Oliveira and Urmian, 2026a; de Oliveira Oliveira and Vadiee, 2024]. The state-reduction techniques are crucial for making these computations feasible. In a representative pathwidth-3 instance with maximum degree 2, raw breadth-first exploration generates 24,482,279 states, whereas the combined canonization-and-pruning configuration explores only 141 states. In the

largest published positive verification run, concerning pathwidth at most 5 and maximum degree 3, the combined search remains within 746,187 states and terminates in about 41 hours. On the treewidth side, the same combination keeps the search within a few thousand states for all reported instances up to treewidth 3, and also terminates within hours on the additional case $(\mathrm{tw}, \Delta) = (4, 2)$ [de Oliveira Oliveira and Urmian, 2026a; de Oliveira Oliveira and Vadiee, 2024].

Taken together, these results provide concrete evidence that width-based automated theorem proving is a promising approach to the study of graph-theoretic conjectures. They show that the large body of knowledge on tree-decomposition algorithms for combinatorial problems can be repurposed for automated reasoning: the same dynamic programs that were designed to solve instances of bounded width can also serve as reusable proof-search components for testing conjectures on entire width-bounded graph classes. This is a central conceptual message of the thesis. Width-based ATP is not an alternative to the dynamic-programming tradition of parameterized complexity, but a way of lifting that tradition from single-instance computation to automated reasoning over structurally restricted graph classes [Cygan et al., 2015; de Oliveira Oliveira and Vadiee, 2023, 2024].

The remainder of the thesis is organized into five substantive parts, grouped into three thematic movements: foundations, automated theorem proving, and correctness/validation, followed by a concluding part.

Foundations (Part II–Part III; Chapters 2–4). Chapter 2 fixes notation and reviews the necessary background on parameterized complexity and width measures. Chapter 3 develops the formalism of instructive decompositions and validated terms that represent bounded-width graphs as algebraic objects. Chapter 4 introduces the DP-core interface and builds the core library for common graph properties — connectivity, coloring, and subgraph containment — that the later theorem-proving chapters draw on.

Automated theorem proving (Part IV; Chapters 5–8). Chapter 5 constructs the width-based ATP engine and establishes its correctness and complexity guarantees. The engine's raw state space is large, so Chapters 6–7 develop the two main search-space reductions — state canonization and premise-based pruning — that make bounded-width exploration practical. Chapter 8 then applies the full framework to the triangle-free case of Reed's conjecture.

Correctness and validation (Part V; Chapters 9–10). Chapter 9 develops the systematic local-to-global proof methodology used to certify individual DP-cores, and Appendix D applies it to the full core catalog. Chapter 10 then empirically validates the selected DP-core implementations against independent SAT encodings, with Appendix E documenting the SAT side of the cross-validation protocol in detail.

Conclusion (Part VI). Chapter 11 summarizes the contributions and outlines directions for future work.

Part II

Background

Chapter 2

Background

In this chapter, we fix notation and review the background material used throughout the thesis. Our goal is to align terminology across graph theory, parameterized complexity, and finite tree automata, and to provide a single reference point for definitions that later chapters use without repetition.

Scope and Notation

We work with finite undirected graphs. Unless stated otherwise, graphs are simple and loopless: a graph is denoted by $G = (V, E)$ with $V = V(G)$ and $E = E(G) \subseteq \binom{V}{2}$. In later chapters (notably Sections 3.4 and 4.8) we also consider finite undirected multigraphs without loops (parallel edges allowed), represented as incidence structures (V_G, E_G, ρ_G). Whenever multigraphs are in scope, graph isomorphism and graph properties are interpreted with respect to this incidence structure so that edge multiplicities are preserved. Whenever a statement targets a simple-graph notion, we either assume the realized graph is simple or enforce this promise (e.g. using the multi-edge detector core). The degree of a vertex v is $\deg_G(v)$, the maximum degree is $\Delta(G)$, the clique number is $\omega(G)$, and the chromatic number is $\chi(G)$. For $n \in \mathbb{N}$ we write $[n] = \{1, \ldots, n\}$.

Basic graph operations and notation follow standard graph-theoretic conventions. See, for example, Bondy and Murty [2008]. If $X \subseteq V(G)$, the induced subgraph on X is written $G[X]$ and has vertex set X and all edges of G with both endpoints in X. We write $G - X$ for the induced subgraph on $V(G) \setminus X$. For a vertex $v \in V(G)$, the (open) neighborhood is

$$N_G(v) = \{u \in V(G) : \{u, v\} \in E(G)\},$$

and for $X \subseteq V(G)$ we set $N_G(X) = \bigcup_{x \in X} N_G(x)$ and $N_G[X] = X \cup N_G(X)$.

Two graphs G and H are *isomorphic*, written $G \simeq H$, if they are equivalent up to relabeling of vertices (and, in the multigraph setting, of edges) in a way that preserves incidence. Concretely, for simple graphs $G = (V, E)$ and $H = (W, F)$ this means there exists a bijection $\varphi : V \to W$ such that $\{u, v\} \in E$ if and only if $\{\varphi(u), \varphi(v)\} \in F$ for all $u, v \in V$. For multigraph incidence structures (V_G, E_G, ρ_G) and (V_H, E_H, ρ_H), we additionally require a bijection $\psi : E_G \to E_H$ such that $(e, x) \in \rho_G$ if and only if $(\psi(e), \varphi(x)) \in \rho_H$. In particular, parallel edges are preserved. A *graph property* is a set $\mathcal{P}$ of finite graphs (or multigraphs, depending on context) that is closed under isomorphism.

We write $tw(G)$ and $pw(G)$ for the treewidth and pathwidth of G. Tree decompositions are used in their standard sense. Formal definitions appear where first needed, and a canonical term representation is developed in Chapter 3.

For sets A, $\mathcal{P}(A)$ denotes the power set. Strings and terms over ranked alphabets are written using the customary notation from automata theory. The set of ground terms over an alphabet Σ is denoted by $\mathrm{Terms}(\Sigma)$ and the set of contexts by $\mathcal{C}(\Sigma)$.

For asymptotics, we use $\mathcal{O}(\cdot)$ and suppress polynomial factors in the input size by the notation $\mathcal{O}^*(\cdot)$ (see the discussion in the section on parameterized complexity). Throughout, logarithms are base 2 unless otherwise specified.

2.1 Parameterized Complexity

Classical computational complexity theory evaluates the efficiency of algorithms in terms of the overall size of the input. While this approach has provided foundational insights, it often fails to capture the tractability of problems encountered in practice. Many such problems are NP-hard and are unlikely to admit polynomial-time algorithms in the general case. Yet, in practical instances, additional structure can often be exploited to develop efficient algorithms.

Parameterized complexity addresses this gap by introducing a multivariate perspective on algorithm design. Rather than treating all aspects of the input uniformly, it identifies specific *parameters*, which encapsulate structural features of the input, and studies how these influence the algorithm's performance. This framework enables the design of algorithms whose running time is efficient when the parameter is small, even if the total input size is large. As a result, parameterized complexity has become a powerful and widely applicable tool in modern algorithmic theory.

The success of parameterized complexity relies on selecting parameters that are both meaningful and useful. A good parameterization satisfies two criteria: the parameter tends to be small in real-world instances, and the problem becomes efficiently solvable when the parameter is bounded [Cygan et al., 2015].

Definition 1 (Cygan et al. [2015], p. 12)**.** *A parameterized problem is a language* $L \subseteq \Sigma^* \times \mathbb{N}$*, where* Σ *is a finite alphabet and the second component of each pair is called the parameter.*

For a fixed value p, the p-th slice of the problem is the set

$$L[p] := \{x \in \Sigma^* \mid (x, p) \in L\},$$

which allows us to study the problem on instances with a fixed parameter value [Downey and Fellows, 2013]. These slices provide insight into how the problem behaves across different values of the parameter, revealing complexity patterns that are often obscured in classical formulations.

Consider the Vertex Cover problem: given a graph $G = (V, E)$ and an integer t, does there exist a subset $V' \subseteq V$ of size at most t such that every edge in E is incident to at least one vertex in V'? This can be modeled as a parameterized problem where each instance is a pair (G, t), and the parameter t determines the slice $L[t]$ consisting of all graphs that admit a vertex cover of size at most t.

The central notion in parameterized complexity is that of *fixed-parameter tractability.*

Definition 2 (Cygan et al. [2015], p. 13)**.** *A parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ *is said to be fixed-parameter tractable (FPT) if there exists an algorithm* A*, a computable function* $f : \mathbb{N} \to \mathbb{N}$*, and a constant* $c \in \mathbb{N}$ *such that for every input* (x, p)*, the algorithm decides whether* $(x, p) \in L$ *in time* $f(p) \cdot |x|^c$*.*

In FPT algorithms, the exponential growth is confined to the parameter p, while the input size $|x|$ appears only polynomially, making such algorithms scalable for small parameters even on large inputs.

Definition 3 (Cygan et al. [2015], p. 13)**.** *A parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ *belongs to the class XP if there exist an algorithm and computable functions* $f, g : \mathbb{N} \to \mathbb{N}$ *such that for every input* (x, p)*, the problem can be solved in time* $f(p) \cdot |x|^{g(p)}$*.*

XP is weaker than FPT: the exponent $g(p)$ can grow with p, making algorithms impractical even for moderate parameter values. This distinction provides a fine-grained view of algorithmic tractability and guides both theoretical analysis and practical algorithm design.

Hardness, Kernelization, and Lower Bounds

Two complementary tools guide negative results in parameterized complexity. The first is the W-hierarchy: a problem that is W[1]-hard (typically by reduction from Clique) is not expected to admit an FPT algorithm. The hierarchy provides a calibrated notion of intractability parameterized by structural depth. See Cygan et al. [2015]; Downey and Fellows [2013].

The second is kernelization. A *kernelization* for a parameterized problem is a polynomial-time preprocessing algorithm that maps an instance (x, p) to an equivalent instance (x', p') whose size is bounded by a function of p alone. Many classical FPT problems admit polynomial kernels, but for others one can rule out any polynomial kernel under widely believed assumptions using composition techniques. See Cygan et al. [2015] for an overview.

Lower bounds on the running time are often based on the Exponential Time Hypothesis (ETH) and its strong variant, the Strong Exponential Time Hypothesis (SETH) [Impagliazzo and Paturi, 2001; Impagliazzo et al., 2001]. For many specific problems, known reductions under ETH/SETH rule out running times of the form $2^{o(p)}\, n^{\mathcal{O}(1)}$ or $n^{o(p)}$ when parameterized by p, thereby explaining why known algorithms are essentially optimal up to subexponential factors.

The theory naturally extends to problems with multiple parameters, where the parameter becomes a tuple $(p_1, p_2, \ldots, p_d) \in \mathbb{N}^d$. In such cases, the functions f and g depend on all components, and the definitions of FPT and XP generalize accordingly. This multivariate analysis is particularly useful for problems where structural constraints interact in nontrivial ways.

The significance of fixed-parameter tractability lies in its practical promise: for many problems that are hopelessly intractable in general, efficient solutions emerge when we exploit structure. This has led to breakthroughs in diverse domains such as computational biology, software verification, and network analysis. Moreover, parameterized analysis has reshaped algorithmic thinking by encouraging the search for structure, modularity, and layered complexity.

In parameterized complexity it is common to use the $\mathcal{O}^*(\cdot)$-notation to hide polynomial factors in the input size. Formally, if $T(n, p)$ is a running time for instances of size n with parameter p, then writing

$$T(n, p) = \mathcal{O}^*\big(f(p)\big)$$

means that there exists a polynomial q such that $T(n,p) \leq f(p) \cdot q(n)$ for all relevant n and p. Intuitively, $\mathcal{O}^*(\cdot)$ lets us focus on the parameter-dependent part $f(p)$ and suppress the "lower-order" polynomial dependence on n. We will use this notation throughout the thesis.

2.2 Width Measures

Width measures are numerical invariants that describe how structurally complex a graph is, often in relation to a more constrained class such as trees or paths. These measures play a central role in both structural graph theory and algorithmic applications, especially in the design of fixed-parameter tractable (FPT) algorithms. The general philosophy is that many NP-hard problems become tractable on graphs of bounded width, making these parameters powerful tools for algorithm designers [Bodlaender and Kloks, 1996; Cygan et al., 2015; Downey and Fellows, 2013].

The most well-known width measure is *treewidth*, introduced in the influential work of Robertson and Seymour [Robertson and Seymour, 1984], which captures how treelike a graph is. A more restrictive variant is *pathwidth*, which requires the decomposition to be a path rather than a tree [Bodlaender, 1998]. On the other hand, *clique-width* generalizes treewidth in a different direction and allows algorithmic techniques to extend beyond minor-closed graph classes [Corneil and Rotics, 2005; Courcelle and Olariu, 2000]. In this thesis, clique-width is included mainly to show the breadth of the representation interface; the later operational development remains ITD-centered.

This section introduces these measures, explores their definitions and relationships, and highlights their algorithmic consequences. These notions lay the groundwork for the width-based reasoning developed in later chapters.

2.2.1 Treewidth and Pathwidth

Treewidth measures how closely a graph resembles a tree and plays a central role in parameterized complexity. Many NP-hard problems become fixed-parameter tractable when parameterized by treewidth, typically via dynamic programming on tree decompositions [Bodlaender, 1998; Robertson and Seymour, 1986b].

Definition 4 (Tree decomposition, [Downey and Fellows, 2013])**.** *(a) A tree decomposition of a graph $G = (V, E)$ is a tree T together with a collection of subsets*

T_x (called bags) of V, labeled by the vertices x of T, such that $\bigcup_{x \in T} T_x = V$ and the following connectivity properties (1) and (2) hold:

(1) For every edge uv of G there is some x such that $\{u, v\} \subseteq T_x$.

(2) (Interpolation property) If y is a vertex on the unique path in T from x to z, then $T_x \cap T_z \subseteq T_y$.

(b) The width of a tree decomposition is the maximum value of $|T_x| - 1$ taken over all the vertices x of the tree T of the decomposition.

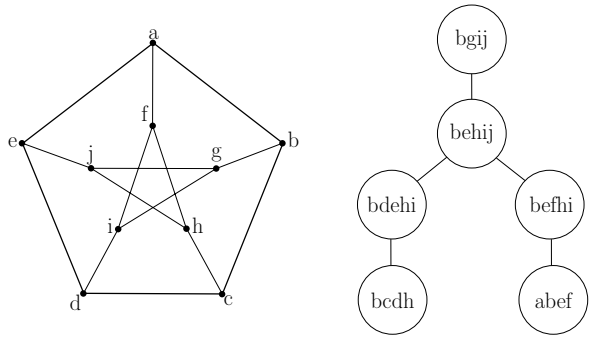


Figure 2.1: The Petersen graph and its tree decomposition.

Definition 5 (Treewidth, [Cygan et al., 2015])**.** *The treewidth of a graph G, denoted by $tw(G)$, is the minimum possible width of a tree decomposition of G.*

For dynamic programming on tree decompositions, the standard notion is a *nice tree decomposition* [Cygan et al., 2015, Def. 7.1], which requires a rooted binary tree where each node is a leaf node, introduce node, forget node, or join node, and the root bag is empty. Any tree decomposition of width k can be efficiently transformed into a nice decomposition of the same width [Cygan et al., 2015, p. 161]. We use the following edge-introducing refinement throughout the thesis.

Definition 6 (Nice edge-introducing tree decomposition)**.** *Let $G = (V, E)$ be a graph. A nice edge-introducing tree decomposition of G is a tuple $(T, \{X_x\}_{x \in V(T)}, \lambda)$ where $(T, \{X_x\})$ is a tree decomposition of G, T is rooted and binary, λ is a partial map whose domain is exactly the introduce-edge nodes, and each node $x \in V(T)$ is of one of the following five types:*

1. ***Leaf node:*** *x is a leaf of T and $X_x = \varnothing$.*
2. ***Introduce-vertex node:*** *x has a unique child y and there exists $v \in V$ such that $X_x = X_y \cup \{v\}$.*

3. ***Forget-vertex node:*** *x has a unique child y and there exists $v \in V$ such that $X_x = X_y \setminus \{v\}$.*

4. ***Introduce-edge node:*** *x has a unique child y, $X_x = X_y$, and $\lambda(x) = \{u, v\} \in E$ for some $u, v \in X_x$.*

5. ***Join node:*** *x has exactly two children y_1, y_2 and $X_x = X_{y_1} = X_{y_2}$.*

We require that every edge of G is introduced exactly once, i.e., for every $e \in E$ there is exactly one introduce-edge node x with $\lambda(x) = e$.

Remark 1. Unlike Definition 6, the standard definition of a nice decomposition additionally requires an empty root bag. This is without loss of generality: if $(T, \{X_x\}, \lambda)$ is a nice edge-introducing decomposition with root bag X_r, then adding a chain of forget nodes above r that forgets all vertices in X_r yields a decomposition of the same width whose root bag is empty.

Lemma 1 (Nice decompositions admit edge-introducing refinement)**.** *If a graph G admits a nice tree decomposition of width at most k, then it also admits a nice edge-introducing tree decomposition of width at most k.*

Proof. Let $(T, \{X_x\}_{x \in V(T)})$ be a nice tree decomposition of G of width at most k, rooted at an empty bag. For each edge $e = \{u, v\} \in E(G)$, the set $S_e := \{x \in V(T) : \{u, v\} \subseteq X_x\}$ is a nonempty connected subtree by the tree-decomposition axioms. Let x_e be the vertex of S_e closest to the root, and let p_e be its parent. Since x_e is chosen closest to the root, $\{u, v\} \nsubseteq X_{p_e}$, so p_e cannot be a join node and x_e is its unique child.

For each node $x \in V(T)$, let $E_x := \{e \in E(G) : x_e = x\}$. If $E_x = \{e_1, \ldots, e_m\}$, then x is not the root; let p_x denote its parent. Subdivide the edge from p_x to x by inserting a chain $p_x \to z_{x,1} \to \cdots \to z_{x,m} \to x$, where each new bag equals X_x and $\lambda(z_{x,j}) = e_j$. Declare each $z_{x,j}$ to be an introduce-edge node.

The resulting tree is binary (we only subdivide edges), the width remains at most k (no bag is enlarged), and the tree-decomposition axioms are preserved (new bags are copies of X_x). Each edge $e \in E(G)$ is introduced exactly once on the chain above x_e, yielding a nice edge-introducing tree decomposition of width at most k. □

For the purposes of later chapters, we will use the edge-introducing refinement from Lemma 1 as the decomposition model underlying instructive tree decompositions.

Closely related to treewidth is the notion of *pathwidth*, which imposes an even stricter structure by requiring the decomposition tree to be a path. Pathwidth captures how

"path-like" a graph is and finds use in applications such as very-large-scale integration (VLSI) design, scheduling, and string processing [Kloks, 1994].

Definition 7 (Pathwidth [Cygan et al., 2015])**.** *A path decomposition is a tree decomposition in which the underlying tree is a path. The pathwidth of a graph G, denoted $pw(G)$, is the minimum width over all valid path decompositions of G.*

Since every path is a tree, it follows that $tw(G) \leq pw(G)$ for every graph G, and the inequality may be strict. For example, a complete binary tree on n vertices has treewidth 1 but pathwidth $\Theta(\log n)$ [Kloks, 1994]. The same inequality holds at the level of the instructive decompositions introduced in the next chapter: every k-instructive path decomposition (a join-free term in IPD_k) is in particular a k-instructive tree decomposition (a term in ITD_k), so a graph realizable at pathwidth at most k is also realizable at treewidth at most k.

By convention we set $tw(\emptyset) = pw(\emptyset) = 0$ for the empty graph, in agreement with the leaf-only ITD term, which realizes the empty graph at width 0. While many results for bounded treewidth also apply to bounded pathwidth, the latter often allows simpler dynamic programming formulations and smaller state spaces. However, like treewidth, computing the exact pathwidth is NP-hard [Arnborg et al., 1987]. In the following, we will present some representative results for problems parameterized by treewidth. They illustrate how bounded treewidth turns many classical NP-hard problems into fixed-parameter tractable ones via dynamic programming on a given decomposition.

Theorem 1. *[Cygan et al., 2015, p. 176] Let G be an n-vertex graph together with a tree decomposition of width at most k. Then for the graph G, we can solve:*

- Vertex Cover *and* Independent Set *in time* $2^k \cdot k^{\mathcal{O}(1)} \cdot n$,
- Dominating Set *in time* $4^k \cdot k^{\mathcal{O}(1)} \cdot n$,
- Odd Cycle Transversal *in time* $3^k \cdot k^{\mathcal{O}(1)} \cdot n$,
- MaxCut *in time* $2^k \cdot k^{\mathcal{O}(1)} \cdot n$,
- *for every fixed* $q \in \mathbb{N}$, q-Coloring *in time* $q^k \cdot k^{\mathcal{O}(1)} \cdot n$.

Theorem 2. *[Cygan et al., 2015, p. 176] Let G be an n-vertex graph together with a tree decomposition of width at most k. Then one can solve each of the following problems in G in time $k^{\mathcal{O}(k)} \cdot n$:*

- Steiner Tree,

- Feedback Vertex Set,
- Hamiltonian Path *and* Longest Path,
- Hamiltonian Cycle *and* Longest Cycle,
- Chromatic Number,
- Cycle Packing,
- Connected Vertex Cover,
- Connected Dominating Set,
- Connected Feedback Vertex Set.

Together, these two theorems illustrate the algorithmic power of bounded treewidth. A wide range of classical NP-complete problems, such as Vertex Cover, Dominating Set, and Hamiltonian Path, admit fixed-parameter tractable algorithms once we are given a tree decomposition of width k. In each case, the running time is of the form $f(k) \cdot n^{\mathcal{O}(1)}$, where the exponential dependence on k comes from a dynamic program over the decomposition, while the dependence on the input size n remains polynomial.

One of our main goals is to provide a uniform framework for expressing and reasoning about such dynamic programs. In later chapters, we will see how tree decompositions, and more generally *treelike decompositions*, can be translated into terms, and how dynamic programming algorithms over these decompositions can be captured by treelike DP-cores and verified using tree automata.

2.2.2 Clique-width

Clique-width is an alternative width measure that applies to broader graph classes than treewidth. Unlike treewidth, clique-width can remain bounded on non-sparse classes such as cographs and distance-hereditary graphs [Corneil and Rotics, 2005; Courcelle and Olariu, 2000; Oum, 2008]. While the present thesis focuses on treewidth and pathwidth, clique-width demonstrates that the width-based perspective extends beyond sparse graphs.

A detailed treatment of clique-width, including formal definitions, expression trees, computational complexity, and its relationship to treewidth, is provided in Appendix C. That appendix also shows how clique-width can be expressed as a treelike decomposition class in the sense of Chapter 3, demonstrating that the general framework developed in this thesis is not limited to tree decompositions.

2.3 Terms and Tree Automata

We use bottom-up finite tree automata only as a lightweight validation tool for decomposition terms. Later chapters do not rely on the broader textbook theory of tree automata. Instead, Chapter 3 instantiates the material below with a concrete recognizer for instructive tree decompositions (ITD), and the actual property computation is then carried out by DP-cores on those validated terms. This section therefore keeps only the notation and facts needed to follow that development.

Our terminology follows the standard presentation of Comon et al. [2008].

2.3.1 Terms

We first recall the term notation used throughout the representation and automata chapters.

Definition 8 (Ranked alphabet)**.** *A ranked alphabet is a pair* (Σ, r) *where* Σ *is a finite set of symbols and* $\mathsf{r} : \Sigma \to \mathbb{N}$ *assigns an arity to each symbol. For* $p \in \mathbb{N}$ *we write*

$$\Sigma_p := \{f \in \Sigma : \mathsf{r}(f) = p\}$$

for the symbols of arity p*.*

Definition 9 (Ground terms)**.** *Let* (Σ, r) *be a ranked alphabet. The set* $\mathrm{Terms}(\Sigma)$ *of ground terms over* Σ *is the smallest set such that:*

- *every constant* $a \in \Sigma_0$ *belongs to* $\mathrm{Terms}(\Sigma)$*, and*
- *whenever* $f \in \Sigma_p$ *with* $p \geq 1$ *and* $\tau_1, \ldots, \tau_p \in \mathrm{Terms}(\Sigma)$*, the term* $f(\tau_1, \ldots, \tau_p)$ *also belongs to* $\mathrm{Terms}(\Sigma)$*.*

We write $\mathcal{C}(\Sigma)$ for the one-hole contexts over Σ. Operationally, a context is a term with one distinguished hole, and $C[\tau]$ denotes the term obtained by plugging τ into that hole.

Definition 10 (Size and height of a term)**.** *Let* $\tau \in \mathrm{Terms}(\Sigma)$*. Its size* $|\tau|$ *and height* $\textsc{Height}(\tau)$ *are defined inductively by:*

- *if* $a \in \Sigma_0$*, then* $|a| := 1$ *and* $\textsc{Height}(a) := 0$*;*

- *if* $\tau = f(\tau_1, \ldots, \tau_p)$ *with* $p \geq 1$, *then*

$$|\tau| := 1 + \sum_{i=1}^{p} |\tau_i|, \qquad \textsc{Height}(\tau) := 1 + \max_{1 \leq i \leq p} \textsc{Height}(\tau_i).$$

These conventions are enough for the later algorithmic bounds, where term size and term height are used as the natural input measures.

2.3.2 Bottom-up finite tree automata

The automata used later in the thesis are bottom-up recognizers for regular languages of decomposition terms.

Definition 11 (Bottom-up finite tree automaton)**.** *Let* (Σ, r) *be a ranked alphabet. A bottom-up finite tree automaton over* Σ *is a tuple*

$$\mathcal{A} = (Q, \Sigma, F, \delta),$$

where:

- Q *is a finite set of states,*
- $F \subseteq Q$ *is the set of final states, and*
- *for every symbol* $f \in \Sigma_p$ *of arity* p*, the transition relation* δ *specifies a finite set of rules of the form*

$$f(q_1, \ldots, q_p) \to q$$

with $q_1, \ldots, q_p, q \in Q$.

Such an automaton is evaluated bottom-up on a ground term $\tau \in \mathrm{Terms}(\Sigma)$. A *run* on τ assigns a state to every node of τ such that, at each node labeled by $f \in \Sigma_p$, the state q assigned to that node together with the states already assigned to the p child subterms forms a transition $f(q_1, \ldots, q_p) \to q$. A run is *accepting* if the state assigned to the root belongs to F, and τ is accepted if and only if there exists an accepting run on τ. We write $L(\mathcal{A})$, or equivalently $L(\mathcal{A})$, for the set of all ground terms accepted by $\mathcal{A}$. A tree language is called *recognizable* (or *regular*) if it is of the form $L(\mathcal{A})$ for some bottom-up finite tree automaton $\mathcal{A}$.

2.3.3 Standard facts

The thesis uses only the following standard closure and decidability facts.

Proposition 1 (Facts about finite tree automata)**.** *Fix a ranked alphabet* Σ*. Recognizable tree languages* $L \subseteq \mathrm{Terms}(\Sigma)$ *(accepted by bottom-up finite tree automata over* Σ*) enjoy the following properties:*

- *they are closed under union, intersection, and complement inside* $\mathrm{Terms}(\Sigma)$*;*
- *emptiness and finiteness are decidable; and*
- *nondeterministic and deterministic bottom-up finite tree automata have the same expressive power.*

See, for instance, Doner [1970] and Comon et al. [2008, Chapter 1]. We cite Proposition 1 as needed, without reproving it.

Part III

Dynamic Programming Cores

Chapter 3

Treelike Width Measures and Instructive Tree Decomposition

This chapter develops the representation layer used throughout the thesis. We represent bounded-width graphs as decomposition terms in regular tree languages, equipped with explicit constructor-wise graph semantics. This yields a uniform, automaton-checkable notion of well-formed decomposition and a common interface for later DP-core evaluation and width-bounded exploration. We state the framework for general treelike decomposition classes and then instantiate it to instructive tree decompositions.

3.1 Treelike Decomposition Classes

Definition 12 (Treelike Decomposition Class)**.** *Let $r \in \mathbb{N}$. A treelike decomposition class of arity r is a sequence*

$$\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}},$$

where for each $k \in \mathbb{N}$ (and where GRAPHS *denotes the ambient class of all finite graph objects under the chosen semantics; in the ITD instantiation below this is the class of incidence structures of finite loopless undirected multigraphs):*

- *Σ_k is a ranked alphabet of arity at most r,*
- *L_k is a regular tree language over Σ_k,*
- *$\mathcal{G}_k : \mathsf{L}_k \to$ GRAPHS is a function assigning a graph to each term.*

In addition, we require a monotone hierarchy and compatibility across width bounds:

- $\Sigma_k \subseteq \Sigma_{k+1}$,
- $\mathsf{L}_k \subseteq \mathsf{L}_{k+1}$,
- *(Compatibility) for every* $\tau \in \mathsf{L}_k$ *we have* $\mathcal{G}_{k+1}(\tau) = \mathcal{G}_k(\tau)$.

The terms in $\mathsf{L}(\mathcal{C}) = \bigcup_{k\in\mathbb{N}} \mathsf{L}_k$ are called $\mathcal{C}$*-decompositions*. For any such term τ, we write $\mathcal{G}(\tau)$ for $\mathcal{G}_k(\tau)$ when the index is clear. The $\mathcal{C}$*-width* of τ is the smallest k such that $\tau \in \mathsf{L}_k$, and the $\mathcal{C}$-width of a graph G is the smallest k for which there exists $\tau \in \mathsf{L}_k$ with $\mathcal{G}(\tau) \simeq G$.

We denote $\mathcal{C}_k = (\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)$ and define the set of graphs:

$$\mathcal{P}[\mathcal{C}_k] = \mathrm{ISO}(\{\mathcal{G}(\tau) : \tau \in \mathsf{L}_k\}),$$

where $\mathrm{ISO}(S)$ denotes the isomorphism-closure of S. By construction,

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{C}_{k+1}], \quad \text{and} \quad \mathcal{P}[\mathcal{C}] = \bigcup_{k\in\mathbb{N}} \mathcal{P}[\mathcal{C}_k].$$

A graph $G \in \mathcal{P}[\mathcal{C}]$ has finite $\mathcal{C}$-width. If no such decomposition exists, we define $w_{\mathcal{C}}(G) = \infty$.

3.2 Treelike Width Measures

Building on the notion of treelike decomposition classes, we next define the corresponding width parameters, known as treelike width measures. We call a graph invariant *treelike* if it is realized by some treelike decomposition class [de Oliveira Oliveira and Vadiee, 2023].

Definition 13 (Treelike Width Measure)**.** *Let* $\mathcal{P}$ *be a graph property and let* $\mathcal{M} : \mathcal{P} \to \mathbb{N}$ *be a graph invariant. We say that* $\mathcal{M}$ *is a treelike width measure if there exists a treelike decomposition class* $\mathcal{C}$ *such that:*

- $\mathcal{P} = \mathcal{P}[\mathcal{C}]$,
- *for every* $G \in \mathcal{P}$*, we have* $w_{\mathcal{C}}(G) = \mathcal{M}(G)$.

In this case, we say that $\mathcal{C}$ *is a realization of* $\mathcal{M}$.

3.3 Automaton Realizability and Complexity

In applications such as model checking and automated theorem proving, it is important not only to know that a graph class is regular, but also to understand whether its description can be effectively realized by a finite-state automaton of manageable size. For this reason, we associate a sequence of tree automata with each treelike decomposition class and analyze the complexity of this realization.

Definition 14 (Automaton Realizability)**.** *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class of arity r. A recognizer sequence of tree automata for $\mathcal{C}$ is a sequence*

$$\mathcal{A} = \{\mathcal{A}_k\}_{k\in\mathbb{N}}$$

such that for every $k \in \mathbb{N}$, the automaton $\mathcal{A}_k$ accepts the language L_k. That is, $\mathcal{L}(\mathcal{A}_k) = \mathsf{L}_k$.

Definition 15 (Complexity)**.** *We say that a recognizer sequence $\mathcal{A}$ has complexity $f : \mathbb{N} \to \mathbb{N}$ if:*

- *Each automaton $\mathcal{A}_k$ has at most $f(k)$ states,*
- *There exists an algorithm that, given $k \in \mathbb{N}$, constructs $\mathcal{A}_k$ in time $k^{O(1)} \cdot f(k)^{O(r)}$.*

3.4 Instructive Tree Decomposition

We emphasize that ITD is not a new notion of treewidth: it is a DP-friendly term normal form of the standard nice edge-introducing tree decomposition paradigm, with the key extra feature that well-formedness becomes a regular tree-language property validated once by a recognizer automaton [Bodlaender and Kloks, 1996; Cygan et al., 2015].

We now define the symbols used to encode the instructions of an instructive decomposition. These symbols form the alphabet over which decomposition terms are defined.

Definition 16 (Instructive Alphabets)**.** *For each $k \in \mathbb{N}$, the k-instructive alphabet Σ_k is defined as*

$$\begin{aligned}\Sigma_k = {} & \{\texttt{leaf}, \texttt{join}\} \\ & \cup \{\texttt{intro\_}u, \texttt{forget\_}u \;:\; u \in [k+1]\} \\ & \cup \{\texttt{intro\_}u\texttt{\_}v \;:\; u, v \in [k+1],\ u \neq v\}.\end{aligned}$$

The symbols have the following arities:

- `leaf` *has arity* 0,
- `intro_`u, `forget_`u, *and* `intro_`u`_`v *have arity* 1,
- `join` *has arity* 2.

The elements of Σ_k are to be understood as *instructions* for inductively constructing graphs. Each graph under construction maintains a set $\mathbf{b} \subseteq [k+1]$ of *active labels*, which identify the vertices currently available for reference. We will use the word *bag* for such active label sets; formally, for a term τ, its top bag $B(\tau) \subseteq [k+1]$ is a set of labels, and the corresponding set of active vertices in the realized graph is $\theta[\tau](B(\tau)) \subseteq V(\mathcal{G}(\tau))$ (defined later in the term semantics).

The inductive construction proceeds as follows:

- The instruction `leaf` creates an empty graph with an empty set of active labels.
- Given a graph G with active labels $\mathbf{b}$, the instruction `intro_`u (for $u \notin \mathbf{b}$) adds a new vertex labeled u and updates the active label set to $\mathbf{b} \cup \{u\}$.
- The instruction `forget_`u (for $u \in \mathbf{b}$) removes the label u from its vertex and updates the active label set to $\mathbf{b} \setminus \{u\}$. The underlying vertex remains part of the graph, but its label becomes available for reuse.
- The instruction `intro_`u`_`v (for $u, v \in \mathbf{b}$ and $u \neq v$) introduces an edge between the vertices labeled u and v. Multiple edges between the same pair of vertices are permitted.
- Given two graphs G and G' with identical active label sets $\mathbf{b}$, the instruction `join` merges them by identifying, for each $u \in \mathbf{b}$, the vertex labeled u in G with the corresponding vertex in G'.

For the instructive decomposition class, a recognizer automaton $\mathcal{A}_k$ only needs to track the set of active labels (the current bag) and to verify the side conditions of each instruction. Concretely, a state can be taken as a subset $B \subseteq [k+1]$, and the transitions mirror the intended semantics:

$$\begin{aligned} \texttt{leaf} &\to \emptyset, \\ \texttt{intro\_}v(B) &\to B \cup \{v\}, \\ \texttt{forget\_}v(B) &\to B \setminus \{v\}, \\ \texttt{intro\_}u\texttt{\_}v(B) &\to B, \\ \texttt{join}(B, B) &\to B, \end{aligned}$$

Constructor	Precondition on bag B	Bag update	Graph update
`leaf`	none	$\emptyset$	initialize empty graph
`intro_`u	$u \notin B$	$B \cup \{u\}$	add a fresh vertex labeled u
`forget_`u	$u \in B$	$B \setminus \{u\}$	remove label u; keep underlying vertex
`intro_`u`_`v	$u, v \in B$, $u \neq v$	B	add an edge between currently labeled u and v
`join`	child bags are equal to B	B	glue partial graphs along the boundary labels in B

Table 3.1: Instruction cheat sheet for ITD_k.

with each transition permitted precisely when the corresponding admissibility condition (e.g., $v \notin B$ for introduce, $u, v \in B$ for edge, equal bags for join) holds. Therefore $|Q| \leq 2^{k+1}$, and the sequence $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ has complexity $\mathcal{O}(2^k)$. The transition description is polynomial in $|Q|$ and the alphabet arity. Closure and decidability properties follow from the general theory of finite tree automata (Proposition 1).

As a concrete instance, consider a width-one instructive term whose active-label sets evolve as $B_0 = \emptyset$, $B_1 = \{1\}$, $B_2 = \{1, 2\}$, $B_3 = \{2\}$, $B_4 = \{1, 2\}$, $B_5 = \{1\}$, $B_6 = \{1, 2\}$, $B_7 = \{2\}$, $B_8 = \emptyset$.

Consider two subterms whose top bags are both $\{v\}$ and where, below, one subterm introduced an edge incident to v and the other introduced a different edge incident to v. The instruction `join` is admissible because the bags match. It merges the two partial graphs by identifying the two occurrences of v. The resulting term represents the union of these partial graphs glued at v. This illustrates why the join operation requires bag equality and why the automaton checks that side condition.

An example of a k-instructive tree decomposition is presented in Figure 3.1. This figure depicts a 2-instructive tree decomposition term, and Figure 3.2 explicitly illustrates how the graph associated with this decomposition is incrementally built, highlighting active labels and intermediate steps.

Every finite loopless undirected multigraph G with $tw(G) \leq k$ admits a nice edge-introducing tree decomposition $(T, \{X_u\}_{u\in V(T)}, \lambda)$ of width at most k for its underlying simple graph G^{simp} (Definition 6; see also Lemma 1). From such a decomposition one obtains an instructive term $\tau \in \mathsf{ITD}_k$ by first computing a bag-injective label assignment $\alpha : V(G) \to [k + 1]$ that is consistent along the decomposition (labels are preserved while vertices persist in bags), and then replacing each node type by the

corresponding constructor: introduce-vertex nodes by `intro_`$\alpha(v)$, forget-vertex nodes by `forget_`$\alpha(v)$, join nodes by `join`, and an introduce-edge node for $\{u, v\}$ by a unary chain of $m_G(u, v)$ copies of `intro_`$\alpha(u)$`_`$\alpha(v)$, where $m_G(u, v)$ is the multiplicity of $\{u, v\}$ in G. Conversely, starting from a validated instructive term τ, one recovers a nice edge-introducing decomposition of $\mathcal{G}(\tau)^{\text{simp}}$ by reading constructors bottom-up and attaching one decomposition node per constructor, except that repeated `intro_`$\cdot$`_`$\cdot$ steps between the same two currently active vertices do not create new edges in the simple shadow; if an empty root bag is desired, it can be enforced (without changing the realized graph) by appending a chain of forget nodes above the root, as in Remark 1.

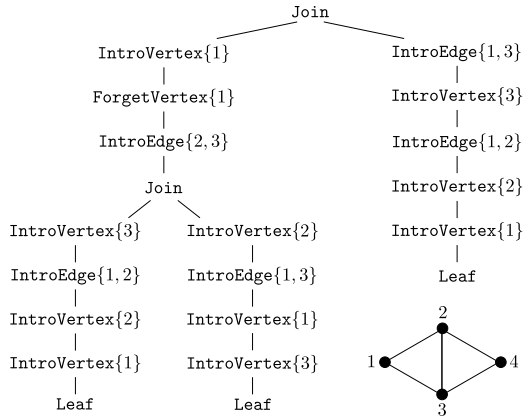


Figure 3.1: A 2-instructive tree decomposition τ, and the graph $\mathcal{G}(\tau)$ associated with τ. Notice that the graph contains four vertices even though only labels $1, 2, 3$ are used in the decomposition. Once a vertex has been forgotten, its label becomes available for reuse.

Each graph constructed in this manner can be encoded by a *term* over the alphabet Σ_k. We denote by ITD_k the set of all such terms, referring to them as k-instructive tree decompositions.

To formalize the notion of k-instructive tree decompositions, we define a tree automaton that recognizes precisely the valid construction terms. We define the tree automaton

$$\mathcal{A}_k = (Q_k, \Sigma_k, F_k, \delta_k)$$

where:

- $Q_k = F_k = \mathcal{P}[k+1]$, the set of all subsets of $[k+1]$,
- the transition relation δ_k is defined as follows:

$$\begin{aligned}\delta_k \;=\;& \{\texttt{leaf} \to \emptyset\} \cup \\ & \{\texttt{intro\_}u(\mathbf{b}) \to \mathbf{b} \cup \{u\} \mid \mathbf{b} \subseteq [k+1],\ u \in [k+1] \setminus \mathbf{b}\} \cup \\ & \{\texttt{forget\_}u(\mathbf{b}) \to \mathbf{b} \setminus \{u\} \mid \mathbf{b} \subseteq [k+1],\ u \in \mathbf{b}\} \cup \\ & \{\texttt{intro\_}u\texttt{\_}v(\mathbf{b}) \to \mathbf{b} \mid \mathbf{b} \subseteq [k+1],\ u, v \in \mathbf{b},\ u \neq v\} \cup \\ & \{\texttt{join}(\mathbf{b}, \mathbf{b}) \to \mathbf{b} \mid \mathbf{b} \subseteq [k+1]\}.\end{aligned}$$

Since $\mathcal{A}_k$ is deterministic on well-formed instructive terms, each $\tau \in \mathsf{ITD}_k$ admits a unique bottom-up run. For any subterm σ of such a term, we write $B(\sigma) \subseteq [k+1]$ for the state assigned by that run to the root of σ (in particular, $B(\tau)$ for the full term). Under the graph semantics below, this coincides with $\mathrm{dom}(\theta[\sigma])$, i.e., the active-label domain at σ.

Root-closed vs. root-open terms. The automaton above accepts terms regardless of the final active-label set (we take $F_k = Q_k$). If one wants to enforce root-closed terms syntactically, it suffices to set $F_k = \{\emptyset\}$. Equivalently, one can post-process a term by forgetting any remaining active labels. Unless explicitly stated otherwise, we work over the full language ITD_k (arbitrary root bag) and use root closure only as a presentation device; see the opening discussion of Chapter 5, in particular Section 5.1.

The states of $\mathcal{A}_k$ represent the sets of active labels during the graph construction process. The transition relation prescribes, for each instruction and current label set, the resulting active label set after applying the instruction. We may thus formalize as follows.

Definition 17 (ITD_k and IPD_k)**.** *The terms in $\mathsf{ITD}_k = L(\mathcal{A}_k)$ are referred to as k-instructive tree decompositions. Those terms that do not use the symbol* `join` *are referred to as k-instructive path decompositions. We denote the set of all k-instructive path decompositions by IPD_k.*

In order to associate a concrete graph with each k-instructive tree decomposition, we introduce the notion of k-boundaried graphs and formalize the graph construction process. Boundaried graphs provide a convenient way to manage active labels during graph construction and to define a join operation that reflects the semantics of the `join` instruction. Throughout Section 3.4, realized objects are finite undirected multigraphs without loops: each edge has two distinct endpoints, and parallel edges may arise (for example by repeated `intro_`$\cdot$`_`$\cdot$ steps or through joins). When we later speak about treewidth, we always mean the treewidth of the underlying simple graph G^{simp} obtained from G by forgetting edge multiplicities. Parallel edges do not affect whether a family of bags covers all vertices and edge endpoints, so this convention is harmless for width arguments. We represent a graph G as (V_G, E_G, ρ_G), where vertices and edges are indexed by $[n]$ and $[m]$, respectively, and ρ_G records the endpoints of each indexed edge. In

particular,

$$tw(G) = tw(G^{\text{simp}})$$

for every finite loopless undirected multigraph G.

Given natural numbers $n_1, n_2 \in \mathbb{N}$ and a subset $Y \subseteq [n_2]$, we define

$$f[n_1, n_2, Y] : [n_2] \setminus Y \to \{n_1 + 1, \ldots, n_1 + n_2 - |Y|\}$$

to be the unique injective function from $[n_2] \setminus Y$ into $\{n_1 + 1, \ldots, n_1 + n_2 - |Y|\}$ such that for all $i, j \in [n_2] \setminus Y$, if $i < j$, then $f(i) < f(j)$. Intuitively, for each $i \in [n_2] \setminus Y$, the value $f[n_1, n_2, Y](i)$ is equal to n_1 plus the rank of i among the elements of $[n_2] \setminus Y$.

We next formalize boundaried graphs and the join operation.

Informally, if (G_1, θ_1) and (G_2, θ_2) share a boundary-label domain B, then $(G_1, \theta_1) \oplus_B (G_2, \theta_2)$ is the disjoint union of the two partial graphs followed by identifying equally labeled boundary vertices. To keep vertex sets in canonical indexed form $[n]$, non-boundary vertices coming from the second operand are injectively renumbered.

Definition 18. *Let $k \in \mathbb{N}$. A k-boundaried graph is a pair (G, θ), where G is a graph and $\theta : B \to V_G$ is an injective map from a subset $B \subseteq [k+1]$ to the vertex set V_G of G. We assume that $V_G = [n]$ and $E_G = [m]$ for some $n, m \in \mathbb{N}$.*

Given two k-boundaried graphs (G_1, θ_1) and (G_2, θ_2) whose boundary maps are defined on a common domain $B \subseteq [k+1]$, we define their *join*, denoted

$$(G_1, \theta_1) \oplus_B (G_2, \theta_2),$$

as the k-boundaried graph (G, θ) obtained by identifying, for each $u \in B$, the vertex $\theta_1(u)$ of G_1 with the vertex $\theta_2(u)$ of G_2.

More precisely, suppose $V_{G_1} = [n_1]$, $E_{G_1} = [m_1]$, $V_{G_2} = [n_2]$, and $E_{G_2} = [m_2]$. Let $f = f[n_1, n_2, \theta_2(B)]$. Then (G, θ) is defined as follows:

1. $V_G = [n_1 + n_2 - |\theta_2(B)|]$,
2. $E_G = [m_1 + m_2]$,
3. $\rho_G = \rho_{G_1} \cup \{(e + m_1, \theta_1(u)) : u \in B,\ (e, \theta_2(u)) \in \rho_{G_2}\} \cup \{(e + m_1, f(x)) : (e, x) \in \rho_{G_2},\ x \notin \theta_2(B)\}$,
4. For each $u \in B$, we set $\theta(u) = \theta_1(u)$.

We now define the process that associates a graph with each instructive decomposition term. The following semantics (incidence-structure graph representation together with an injective boundary map) follows de Oliveira Oliveira and Vadiee [2023, Appendix B].

For each $k \in \mathbb{N}$, let

$$\Theta_k = \{f : \mathbf{b} \to \mathbb{N} \mid \mathbf{b} \subseteq [k+1],\ f \text{ injective}\}$$

be the set of injective functions from subsets of $[k+1]$ to $\mathbb{N}$.

We define two functions:

- $\mathcal{G} : \bigcup_{k\in\mathbb{N}} \mathsf{ITD}_k \to \text{GRAPHS}$,
- $\theta : \bigcup_{k\in\mathbb{N}} \mathsf{ITD}_k \to \bigcup_{k\in\mathbb{N}} \Theta_k$,

where for each k and each k-instructive tree decomposition τ, the pair $(\mathcal{G}(\tau), \theta[\tau])$ forms a k-boundaried graph. The elements of the domain of $\theta[\tau]$ are referred to as active labels.

The functions $\mathcal{G}$ and θ are defined inductively as follows:

1. If $\tau = \texttt{leaf}$, then $\mathcal{G}(\tau) = (\emptyset, \emptyset, \emptyset)$ and $\theta[\tau] = \emptyset$.
2. If $\tau = \texttt{intro\_}u(\sigma)$, then

   $$\mathcal{G}(\tau) = \left(V_{\mathcal{G}(\sigma)} \cup \{|V_{\mathcal{G}(\sigma)}| + 1\},\ E_{\mathcal{G}(\sigma)},\ \rho_{\mathcal{G}(\sigma)}\right),$$

   and

   $$\theta[\tau] = \theta[\sigma] \cup \{(u, |V_{\mathcal{G}(\sigma)}| + 1)\}.$$

3. If $\tau = \texttt{forget\_}u(\sigma)$, then

   $$\mathcal{G}(\tau) = \mathcal{G}(\sigma) \quad \text{and} \quad \theta[\tau] = \theta[\sigma] \setminus \{(u, \theta[\sigma](u))\}.$$

4. If $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$, then

   $$\begin{aligned}\mathcal{G}(\tau) = (&V_{\mathcal{G}(\sigma)},\ E_{\mathcal{G}(\sigma)} \cup \{|E_{\mathcal{G}(\sigma)}| + 1\},\\ &\rho_{\mathcal{G}(\sigma)} \cup \{(|E_{\mathcal{G}(\sigma)}| + 1, \theta[\sigma](u)), (|E_{\mathcal{G}(\sigma)}| + 1, \theta[\sigma](v))\}),\end{aligned}$$

   and $\theta[\tau] = \theta[\sigma]$.
5. If $\tau = \texttt{join}(\sigma_1, \sigma_2)$, then

   $$(\mathcal{G}(\tau), \theta[\tau]) = (\mathcal{G}(\sigma_1), \theta[\sigma_1]) \oplus_B (\mathcal{G}(\sigma_2), \theta[\sigma_2]),$$

where $B = \mathrm{dom}(\theta[\sigma_1]) = \mathrm{dom}(\theta[\sigma_2])$.

For illustration purposes, Figure 3.2 shows a step-by-step construction of the graph associated with the 2-instructive tree decomposition of Figure 3.1.

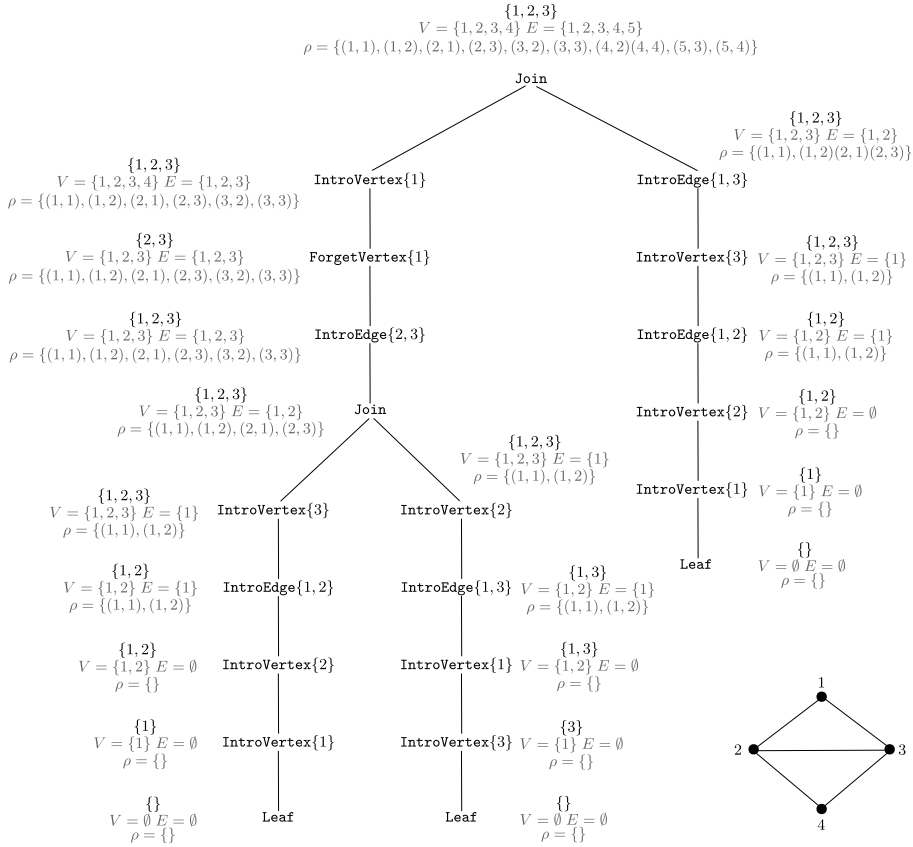


Figure 3.2: Step-by-step construction of the graph associated with the 2-instructive tree decomposition of Figure 3.1. At each node, the current bag B (active labels) and the partial graph of the corresponding subterm are shown. For print-safe rendering, bag annotations are drawn in a dark tone and the (V, E, ρ) tuple lines in a lighter tone, so the two annotation types remain distinguishable in grayscale. Boundary maps are omitted for clarity.

Finally, for each $k \in \mathbb{N}$, let $\mathcal{G}_k$ denote the restriction of $\mathcal{G}$ to ITD_k.

Definition 19 (Instructive tree decomposition class)**.** *The instructive tree decomposition class is the sequence*

$$\mathsf{ITD} = \{(\Sigma_k, \mathsf{ITD}_k, \mathcal{G}_k)\}_{k \in \mathbb{N}},$$

where Σ_k, ITD_k and $\mathcal{G}_k$ are as defined in Section 3.4.

It is readily verified that ITD is a treelike decomposition class of arity 2 in the sense of Definition 12, and that it is automaton-realizable with complexity $\mathcal{O}(2^k)$: each language ITD_k is recognized by the tree automaton $\mathcal{A}_k$ from Section 3.4, which has at most 2^{k+1} states.

3.4.1 Characterizing Treewidth via Instructive Decompositions

In this subsection, we establish that the class of instructive tree decompositions faithfully captures the classical notion of treewidth. That is, a graph has treewidth at most k if and only if it admits a k-instructive tree decomposition. The next lemma states this characterization precisely.

Lemma 2. *Let G be a finite loopless undirected multigraph and $k \in \mathbb{N}$. Then, G has treewidth at most k if and only if there exists a k-instructive tree decomposition τ such that $\mathcal{G}(\tau) \simeq G$.*

To prove this characterization, we show both directions: we first prove that every instructive decomposition encodes a graph of treewidth at most k, and then that every graph of treewidth at most k can be encoded by such a decomposition.

3.4.1.1 From Instructive Decomposition to Treewidth

Assume τ is a k-instructive tree decomposition. By construction (cf. Section 3.4), the graph $\mathcal{G}(\tau)$ is formed by a sequence of graph-building instructions, each involving at most $k+1$ active labels at any given point in the term.

Each application of the `join` operator merges two graphs along a shared boundary of at most $k+1$ vertices. Hence, every intermediate subgraph in the construction process can be viewed as attached to a common bag of size at most $k+1$, and the resulting graph $\mathcal{G}(\tau)$ is glued together through a bounded interface. The following claim formalizes this direction.

Claim 1. *Let τ be a k-instructive tree decomposition. Then the underlying simple graph $\mathcal{G}(\tau)^{\mathrm{simp}}$ admits a tree decomposition of width at most k whose root bag contains precisely the vertices associated with the top active labels $B(\tau)$.*

Proof. We prove the claim by induction on the structure (height) of the term τ.

Base case. If $\tau = \texttt{leaf}$, then $\mathcal{G}(\tau)$ is the empty graph. A decomposition consisting of a single empty bag trivially satisfies the required conditions.

Inductive step. Assume the claim holds for all terms of height at most h, and let τ be a term of height $h+1$.

We distinguish cases based on the topmost instruction in the term:

- If $\tau = \texttt{intro\_}u(\sigma)$, we apply the inductive hypothesis to σ, yielding a tree decomposition (T, X) with root bag X_r. We extend this by adding a new root node r' with bag
$$X_{r'} = X_r \cup \{\theta[\tau](u)\}$$
and child r. This preserves the tree-decomposition axioms: the only new vertex is isolated at this step, every old edge remains covered below, and the bags containing each old vertex stay connected after adjoining a new root bag.

- If $\tau = \texttt{forget\_}u(\sigma)$, then $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$ and
$$\theta[\tau] = \theta[\sigma] \setminus \{(u, \theta[\sigma](u))\}.$$
By the inductive hypothesis, σ has a tree decomposition with root bag $X_r = \theta[\sigma](B(\sigma))$. Let $v := \theta[\sigma](u)$. Add a new root node r' of forget-vertex type with child r and bag $X_{r'} = X_r \setminus \{v\}$. This preserves width and yields the required root bag $\theta[\tau](B(\tau))$. Since forgetting does not change the realized simple graph, every edge remains covered in the child decomposition, and connectedness of the bags containing any remaining vertex is preserved.

- If $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$, let $x := \theta[\sigma](u)$ and $y := \theta[\sigma](v)$. If $\{x, y\}$ is already an edge of the simple graph represented by the child decomposition, we keep that decomposition unchanged. Otherwise, we attach a new root node with the same bag as its child. Because x and y are both active vertices, they both lie in the root bag, so this new root bag covers the new edge $\{x, y\}$. Repeated $\texttt{intro\_}u\texttt{\_}v$ steps therefore create parallel edges in the realized multigraph but do not create new adjacencies in its simple shadow.

- If $\tau = \texttt{join}(\sigma_1, \sigma_2)$, let $B := B(\sigma_1) = B(\sigma_2)$. By induction, each child has a tree decomposition whose root bag realizes its boundary map on B. In the join semantics, boundary vertices with equal labels are identified and non-boundary vertices of the second operand are injectively renamed. Let $V_{\mathcal{G}(\sigma_1)} = [n_1]$, $V_{\mathcal{G}(\sigma_2)} =$

$[n_2]$, and $Y := \theta[\sigma_2](B) \subseteq [n_2]$. Define $\zeta : [n_2] \to V_{\mathcal{G}(\tau)}$ by

$$\zeta(x) = \begin{cases} \theta[\sigma_1](u), & \text{if } x = \theta[\sigma_2](u) \text{ for some } u \in B, \\ f[n_1, n_2, Y](x), & \text{if } x \in [n_2] \setminus Y. \end{cases}$$

Apply ζ uniformly to the right decomposition: replace each bag X_t^2 by $\zeta[X_t^2]$. This yields a tree decomposition of the relabeled right-hand simple graph. After relabeling, the two child roots have the same bag

$$\theta[\sigma_1](B) = \zeta[\theta[\sigma_2](B)],$$

so we attach both roots under a new join root carrying this common bag. Every edge of the joined simple graph is already present in one of the two child simple graphs, hence is covered in one of the two subtrees. For a vertex coming from only one side, the bags containing it remain connected inside that subtree; for a boundary vertex identified across the join, the new common root bag connects its occurrences from the two children. Thus the result is a tree decomposition of the joined simple graph.

In each case, the resulting structure is a tree decomposition of the underlying simple graph, and every bag has size at most $k + 1$. This proves the claim. □

3.4.1.2 From Tree Decomposition to Instructive Decomposition

We now show the converse: if a finite loopless undirected multigraph G has treewidth at most k, then it admits a corresponding k-instructive tree decomposition τ.

Let (T, X, λ) be a nice edge-introducing tree decomposition of G^{simp} (Definition 6) of width at most k; such a decomposition exists by Lemma 1. We define an instructive term τ over the alphabet Σ_k that mirrors the structure of T.

To do this, we must ensure that each vertex can be uniquely identified via a label from $[k+1]$ in each bag. The next definition encodes this requirement through a bag-injective labeling.

Bags. In this section we use the classical tree-decomposition notation and write the bag at a tree node u as X_u. In the term-based representation used later, the corresponding interface bag of a subterm τ is denoted by $B(\tau)$.

Definition 20 (Bag-Injective Labeling)**.** *Let G be a finite graph, and let (T, X, λ) be a nice edge-introducing tree decomposition of G of width at most k. A labeling $\alpha : V_G \to [k+1]$ is bag-injective for (T, X, λ) if its restriction to each bag X_u is injective.*

This is the standard labeling invariant maintained in nice tree decompositions. The next observation spells out the constructive procedure we will rely on later.

Observation 1. *Given a finite graph G and a nice edge-introducing tree decomposition (T, X, λ) of G of width at most k, a bag-injective labeling $\alpha : V_G \to [k+1]$ can be computed in time $O(k \cdot |V(T)|)$.*

Proof. We construct α incrementally by performing a top-down traversal of the decomposition tree T, starting at the root node r.

Let $C(u) = X_u$ denote the current bag at node $u \in V(T)$. For each node u, we ensure the following invariant: *all vertices in $C(u)$ are assigned distinct labels from $[k+1]$, and the labeling is consistent with ancestor nodes.*

We proceed as follows:

1. At the root node r, the bag X_r contains at most $k+1$ vertices. We assign to each vertex in X_r a unique label from $[k+1]$ arbitrarily and store the mapping in α.

2. Now, suppose we have processed a node u and assigned labels to all vertices in X_u so that the invariant holds. Let u' be a child of u. Since the tree decomposition is nice, $X_{u'}$ differs from X_u by at most one vertex. (Because we traverse top-down, the introduce/forget roles are reversed relative to the usual bottom-up DP orientation of Definition 6.) There are three cases:

   - **Introduce-vertex case:** $X_{u'} = X_u \cup \{v\}$ for some $v \in V_G$. Since the decomposition has width at most k, we have $|X_{u'}| \leq k+1$, hence $|X_u| \leq k$. Therefore at least one label in $[k+1]$ is not used in X_u. Assign such a label to v in α and extend the current labeling.
   - **Forget-vertex case:** $X_{u'} = X_u \setminus \{v\}$ for some $v \in V_G$. Since all remaining vertices in $X_{u'}$ are already labeled in X_u, we keep their labels unchanged.
   - **Join case or same bag:** $X_{u'} = X_u$. We propagate the labeling from X_u directly to $X_{u'}$.

3. This process continues recursively for all descendants of r. Each vertex is assigned a label exactly once, namely at the unique node where it is first introduced.

Uniqueness follows from the connectedness axiom for tree decompositions: the set of nodes whose bags contain a fixed vertex forms a connected subtree, so a vertex cannot be introduced independently in two different branches. At a join node both children have the same bag as the parent, so every vertex occurring in both child subtrees is already present in the parent bag and therefore inherits the same label in both branches. Because all label assignments are made from $[k+1]$ and are consistent with existing bag labellings, the resulting labeling is bag-injective. In particular, at every introduce-edge node (which has both endpoints in its bag), the two endpoints receive distinct labels.

It remains to analyze the runtime. Each of the $|V(T)|$ nodes in the tree is visited once. At each node, we perform operations involving sets of size at most $k+1$, such as checking used labels and assigning an unused one. Therefore, the total running time is $O(k \cdot |V(T)|)$.

This concludes the proof. □

Encoding tree decomposition into an instructive term. Let G be a finite loopless undirected multigraph of treewidth at most k, and let (T, X, λ) be a nice edge-introducing tree decomposition of G^{simp} (Definition 6) of width at most k. By Observation 1, we may assume that we are given a bag-injective labeling $\alpha : V_G \to [k+1]$ for (T, X, λ). This labeling allows us to encode the tree decomposition into a k-instructive term by replacing each node in the tree with an instruction over the alphabet Σ_k, where vertex references are replaced by their corresponding label under α.

We define the term τ inductively, using the same tree structure as T, but replacing each node with a symbol as follows:

- If a node $u \in V(T)$ is a **leaf node**, then it becomes `leaf` in the term.
- If u is an **introduce-vertex** node introducing a vertex $v \in V_G$, then we set the corresponding instruction to `intro_`$\alpha(v)$.
- If u is a **forget-vertex** node forgetting vertex v, then the instruction is `forget_`$\alpha(v)$.
- If u is an **introduce-edge** node introducing edge $\{v, w\}$ of the simple shadow, let $m_G(v, w)$ be the number of edges of G with endpoints v and w. We replace u by a unary chain of $m_G(v, w)$ nodes, each carrying the instruction `intro_`$\alpha(v)$`_`$\alpha(w)$. Since v and w appear together in the current bag and α is bag-injective on that bag, we have $\alpha(v) \neq \alpha(w)$ and both labels are active at this point.

- If u is a **join node**, then we assign the instruction `join`.

The result is a well-formed term $\tau \in \mathsf{ITD}_k$. To prove correctness, first expand T by replacing each introduce-edge node with the corresponding unary chain described above. Let τ_u denote the subterm attached to a node u of the expanded tree, and let G_u be the partial multigraph represented by the subtree rooted at u, namely the multigraph with all vertices appearing in that subtree and all edges introduced there with their original multiplicities. We prove by structural induction on u that there exists an isomorphism

$$\varphi_u : \mathcal{G}(\tau_u) \to G_u$$

such that for every vertex $x \in X_u$,

$$\varphi_u\big(\theta[\tau_u](\alpha(x))\big) = x.$$

The leaf, introduce-vertex, and forget-vertex cases follow immediately from the constructor semantics and the bag-injectivity of α on the current bag. In an introduce-edge chain, each application of $\texttt{intro\_}\alpha(v)\texttt{\_}\alpha(w)$ adds one parallel edge between the same two active vertices, so after $m_G(v, w)$ steps the realized multiplicity agrees with G. In the join case, let u be the join node with children u_1, u_2. If a vertex x occurs in both partial graphs G_{u_1} and G_{u_2}, then the set of decomposition nodes whose bags contain x meets both child subtrees. By connectedness of vertex-occurrence subtrees in a tree decomposition, this set must also contain the join node u, hence $x \in X_u$. Therefore the vertex overlap of G_{u_1} and G_{u_2} is exactly the common bag X_u. Moreover, after expanding introduce-edge nodes into unary chains, every edge of G (counted with multiplicity) is still introduced exactly once, at a unique node of the expanded decomposition tree. Hence no edge belongs to both child subtrees of the join: the edge multisets of G_{u_1} and G_{u_2} are disjoint, and only boundary vertices are shared. On that boundary the child isomorphisms agree because both send the active vertex carrying label $\alpha(x)$ to x, and α is injective on X_u. Consequently the two child isomorphisms glue to an isomorphism from the join of the realized child multigraphs to the multigraph union $G_{u_1} \cup G_{u_2}$, with the correct parallel-edge multiplicities preserved. Applying this inductive statement at the root yields $\mathcal{G}(\tau) \simeq G$.

This completes the converse construction.

Proof of Lemma 2. For the forward implication, suppose that $\tau \in \mathsf{ITD}_k$. Claim 1 yields a tree decomposition of the simple shadow $\mathcal{G}(\tau)^{\mathrm{simp}}$ of width at most k. Since treewidth is unchanged by forgetting parallel edges in a loopless multigraph, it follows that $tw(\mathcal{G}(\tau)) \leq k$.

For the converse implication, suppose that G has treewidth at most k. By Lemma 1, the simple shadow G^{simp} admits a nice edge-introducing tree decomposition of width at most k. Observation 1 provides a bag-injective labeling $\alpha : V_G \to [k+1]$. Replacing each tree node by the corresponding instructive constructor, and each introduce-edge node by the unary chain that repeats the same label pair $m_G(v, w)$ times, produces a term τ over Σ_k. The constructor side conditions are satisfied because the edge-introducing decomposition is nice and α is bag-injective on every bag, so $\tau \in \mathsf{ITD}_k$. The inductive argument above then shows that $\mathcal{G}(\tau) \simeq G$. Hence there exists a k-instructive tree decomposition realizing G. □

Theorem 3. *Let* ITD *be the instructive tree decomposition class of Section 3.4. For every finite loopless undirected multigraph G we have*

$$w_{\mathsf{ITD}}(G) \;=\; tw(G).$$

In particular, treewidth is a treelike width measure realized by ITD *in the sense of Definition 13.*

Proof. By Lemma 2, a graph G has treewidth at most k if and only if there exists a k-instructive term $\tau \in \mathsf{ITD}_k$ such that $\mathcal{G}(\tau) \simeq G$. Equivalently, the least k for which such a term exists is exactly $tw(G)$. By Definition 12, this least k is the ITD-width $w_{\mathsf{ITD}}(G)$ of G. The claimed equality follows. □

At this point we have established a uniform term language ITD_k that realizes treewidth, and we have fixed a constructor-wise semantics that maps each instructive term to a (boundaried) graph. This term-and-semantics layer will be the representation layer for all subsequent algorithmic constructions.

Before moving on to DP-cores, we note that the same "graphs as terms" viewpoint extends beyond treewidth. Appendix C shows how clique-width can be presented as a treelike decomposition class with a compatible term semantics, demonstrating that the interface of Definition 12 is not treewidth-specific. The remainder of the thesis, however, only develops the instructive-tree-decomposition instance operationally.

In Chapter 4 we take validated decomposition terms as the input objects for DP-cores, which specify how local witnesses are propagated along constructors (leaf, introduce, forget, edge, join) to decide graph properties or compute invariants.

Chapter 4

Treelike Dynamic Programming Cores

DP-cores provide a disciplined interface for algorithms on bounded-width graphs. Building directly on the instructive decompositions of Chapter 3, a treelike DP-core specifies (i) a witness representation for bag-local information, (ii) constructor-wise transition rules for updating witnesses at introduce/forget/join/edge steps, and (iii) a root-level acceptance condition (and, when needed, a root-level invariant output). Evaluating a core on a validated term is an unambiguous bottom-up computation on finite witness sets, turning the informal dynamic programs that one writes by hand into explicit objects that can be executed, composed, and analyzed.

In what follows we use from Chapter 3 only the instructive tree decomposition language ITD_k and its validation procedure (Section 3.4); the other treelike width measures are included there for generality and will not be used operationally in later chapters.

For orientation, consider the 2-colorability core $\mathcal{C}_2^{\chi}$. A witness at a node is a partition of the current bag B into at most two cells: labels in the same cell are forced to share a color, while color names themselves are ignored. At introduce-vertex steps, witnesses branch by placing the new label into an existing cell or into a fresh singleton cell; at introduce-edge steps, witnesses in which the endpoints of the introduced edge lie in the same cell are removed; at forget-vertex steps, witnesses are restricted to the smaller bag; and at joins, witnesses are combined only when they induce the same partition of the shared bag. These operations mirror the semantics fixed in Definition 16. The labeled total-map presentation used in Chapter 9 is equivalent to this partition presentation; symmetry-aware reductions are treated separately in Chapter 6.

4.1 Treelike DP-cores

We now introduce the notion of a *treelike dynamic programming core* (DP-core).

Definition 21 (Treelike DP-core)**.** *A treelike DP-core is a sequence*

$$\mathcal{D} = \{(\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)\}_{k\in\mathbb{N}}$$

where, for each $k \in \mathbb{N}$:

1. *Σ_k is a ranked alphabet.*
2. *$\mathcal{W}_k$ is a decidable subset of $\{0,1\}^*$.*
3. *$\texttt{Final}_k : \mathcal{P}([k+1]) \times \mathcal{W}_k \to \{0,1\}$.*
4. *Δ_k is a family of transition functions $\{\Delta_{k,a}\}_{a\in\Sigma_k}$ such that for every $a \in \Sigma_k$,*

$$\Delta_{k,a} : \mathcal{P}([k+1]) \times \mathcal{W}_k^{\mathsf{r}(a)} \to \mathcal{P}_{\text{fin}}(\mathcal{W}_k).$$

   (For $\mathsf{r}(a) = 0$ we write $\Delta_{k,a}(B)$ for the resulting finite witness set.)

5. *$\texttt{Clean}_k : \mathcal{P}_{\text{fin}}(\mathcal{W}_k) \to \mathcal{P}_{\text{fin}}(\mathcal{W}_k)$.*
6. *$\texttt{Inv}_k : \mathcal{P}_{\text{fin}}(\mathcal{W}_k) \to \{0,1\}^*$.*

Fix $k \in \mathbb{N}$. The alphabet Σ_k provides the decomposition interface, and a term node labeled by $a \in \Sigma_k$ triggers a local update rule $\Delta_{k,a}$. Witnesses in $\mathcal{W}_k$ encode partial solutions over the current bag $B \subseteq [k+1]$. The witness-level transition may branch, but dynamization aggregates this branching into witness sets and applies $\texttt{Clean}_k$ at each node, yielding a deterministic bottom-up propagation on sets. Acceptance is then tested at the root via $\texttt{Final}_k$; when needed, $\texttt{Inv}_k$ extracts the root output (certificate/invariant) from the final witness set.

In this chapter we treat cleaning abstractly as a postprocessing step on finite witness sets. Additional structural assumptions on cleaning are introduced only when needed later.

We write $\mathcal{D}[k] = (\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)$, and for convenience, we use dot notation to refer to its components, such as $\mathcal{D}[k].\Sigma$ or $\mathcal{D}[k].\texttt{Final}$. When the surrounding context is the instructive decomposition class of Section 3.4, we implicitly take $\mathcal{D}[k].\Sigma = \Sigma_k$ from Definition 16. The extra parameter $B \subseteq [k+1]$ is the current *top bag* (active label set) at the processed term node; in the instructive setting it is

uniquely determined by the term and can be reconstructed by the validation procedure of Section 3.4. We suppress this parameter when it is clear from context or when a core's finality/transition rules do not depend on it. This makes the interface slightly more explicit than in de Oliveira Oliveira and Vadiee [2023], where in the ITD setting the current bag is recovered from the validated term through its unique validation run. For a symbol $a \in \mathcal{D}[k].\Sigma$, we write $\mathcal{D}[k].a$ for the corresponding witness-level transition $\Delta_{k,a}$, and $\mathcal{D}[k].\hat{a}$ for its lifted transition on witness sets (defined next).

4.1.1 Witness-level branching and witness-set propagation

A DP-core packages the dynamic-programming artefact usually presented as "a table per bag together with local update rules". The table rows are *witnesses*: finite objects that encode partial solutions (or partial certificates) over the current bag.

Fix $k \in \mathbb{N}$. At the witness level, for a constructor symbol $a \in \mathcal{D}[k].\Sigma$ of arity r, the local transition $\Delta_{k,a}(B, w_1, \ldots, w_r)$ maps one tuple of child witnesses to a *finite set* of parent witnesses. Allowing a finite set is deliberate: at the witness level, local choices (e.g., assigning colors to a newly introduced vertex, selecting vertices for a set, choosing how to pair endpoints) naturally produce bounded local branching.

During evaluation on a fixed term, this witness-level branching is collected deterministically by lifting transitions to *sets* of witnesses: given child sets $S_1, \ldots, S_r$ we form the union of $\Delta_{k,a}(B, \vec{w})$ over all $\vec{w} \in S_1 \times \cdots \times S_r$, and then apply $\texttt{Clean}_k$ to remove redundancy. Thus, although local transitions may branch on individual witnesses, the bottom-up evaluation is deterministic at the level of witness sets.

We now make the set-level propagation explicit by lifting witness-level transitions to cleaned transitions on finite witness sets; this will be the operation used in the dynamization definition of Section 4.2.

For notational convenience, we lift each witness-level transition $\Delta_{k,a}$ to a function on witness sets and incorporate cleaning. For a symbol $a \in \mathcal{D}[k].\Sigma$ of arity r, a bag $B \subseteq [k+1]$, and witness sets $S_1, \ldots, S_r \subseteq \mathcal{D}[k].\mathcal{W}$, define

$$\mathcal{D}[k].\hat{a}(B, S_1, \ldots, S_r) := \mathcal{D}[k].\texttt{Clean}\left(\bigcup_{(w_1,\ldots,w_r)\in S_1\times\cdots\times S_r} \Delta_{k,a}(B, w_1, \ldots, w_r)\right).$$

For $r = 0$, this agrees with $\mathcal{D}[k].\hat{a}(B) = \mathcal{D}[k].\texttt{Clean}(\Delta_{k,a}(B))$.

The cleaning map $\texttt{Clean}_k$ reduces redundancy (e.g., discarding dominated witnesses in

optimization cores); acceptance and invariants are evaluated on the cleaned witness set output by dynamization.

4.2 Dynamization and Acceptance

Given a treelike DP-core $\mathcal{D}$, the evaluation of a decomposition term τ proceeds via a bottom-up computation called *dynamization*, which assigns witness sets to nodes. Each node aggregates child witnesses, applies its transition rule, and outputs candidates for its own witness set. At the root, we check finality.

Dynamization requires each subterm σ to carry a root bag $B(\sigma) \subseteq [k+1]$. In the instructive setting (Section 3.4), this bag is determined uniquely by the validator automaton run.

Definition 22 (Bag-aware treelike decomposition class)**.** *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class. We call $\mathcal{C}$ bag-aware if for every $k \in \mathbb{N}$ and every term $\tau \in \mathsf{L}_k$, every subterm occurrence σ of τ comes equipped with a root bag $B(\sigma) \subseteq [k+1]$.*

For a bag-aware decomposition class, we fix the following convention. Whenever $\tau \in \mathsf{L}_k$, let $\bar{\tau} \in \mathcal{T}_k^{\mathrm{bag}}$ denote the same term equipped with the bag annotation supplied by the bag-aware realization. Expressions such as $\Gamma[\mathcal{D}, k](\tau)$, $B(\sigma)$, and $\tau \in \mathsf{Acc}(\mathcal{D}[k])$ are shorthand for the corresponding expressions on the annotated representative $\bar{\tau}$. If a term is generated together with an automaton run, $\bar{\tau}$ denotes the run-annotated representative supplied by that run.

Definition 23 (Dynamization)**.** *Let $\mathcal{D}$ be a treelike DP-core. For each $k \in \mathbb{N}$, the k-th dynamization of $\mathcal{D}$ is a function*

$$\Gamma[\mathcal{D}, k] : \mathcal{T}_k^{\mathrm{bag}} \to \mathcal{P}_{\mathrm{fin}}(\mathcal{D}[k].\mathcal{W}),$$

where $\mathcal{T}_k^{\mathrm{bag}} \subseteq \mathrm{Terms}(\mathcal{D}[k].\Sigma)$ denotes the terms all of whose subterms come equipped with a root bag $B(\sigma)$. It is defined inductively as follows:

1. *If $\tau = a$ for some symbol $a \in \mathcal{D}[k].\Sigma$ with arity 0, then*

$$\Gamma[\mathcal{D}, k](\tau) = \mathcal{D}[k].\mathtt{Clean}\big(\Delta_{k,a}(B(\tau))\big).$$

2. *If $\tau = a(\tau_1, \ldots, \tau_r)$, where $a \in \mathcal{D}[k].\Sigma$ has arity r, and $\tau_1, \ldots, \tau_r$ are subterms, then*

we define:

$$\Gamma[\mathcal{D},k](\tau) = \mathcal{D}[k].\texttt{Clean}\left(\bigcup_{(w_1,\dots,w_r)\in\prod_{i=1}^{r}\Gamma[\mathcal{D},k](\tau_i)} \Delta_{k,a}(B(\tau),w_1,\dots,w_r)\right).$$

Transitions that violate semantic constraints return $\emptyset$; this local rejection mechanism ensures global correctness.

Remark 2 (Model checking evaluation on a fixed term). Definition 23 should be read as the model-checking evaluation procedure for a fixed decomposition term. Given a width bound k, a term $\tau \in \mathrm{ITD}_k$, and a DP-core instance $\mathcal{D}[k]$, we: (i) validate that τ is a well-formed width-k term and recover its bag annotation (Section 3.4), (ii) regard the resulting bag-aware representative as the input to $\Gamma[\mathcal{D},k]$ and compute $\Gamma[\mathcal{D},k](\tau)$ bottom-up by applying lifted transitions and cleaning, and (iii) accept iff the root witness set contains at least one final witness. If $\mathcal{D}$ is invariant-valued, the same evaluation produces $\texttt{Inv}_k(\Gamma[\mathcal{D},k](\tau))$ as the output invariant value.

Equivalently, if $\tau = a(\tau_1,\dots,\tau_r)$ has root bag $B = B(\tau)$, then

$$\Gamma[\mathcal{D},k](\tau) = \mathcal{D}[k].\hat{a}\big(B,\Gamma[\mathcal{D},k](\tau_1),\dots,\Gamma[\mathcal{D},k](\tau_r)\big),$$

with the $r = 0$ case agreeing with the nullary definition of $\mathcal{D}[k].\hat{a}$.

Before analyzing running times it is useful to note that dynamization never produces infinite witness sets.

Proposition 2 (Finite Witness Sets)**.** *Let $\mathcal{D}$ be a treelike DP-core. For every $k \in \mathbb{N}$ and every $\tau \in \mathcal{T}_k^{\mathrm{bag}}$, the set $\Gamma[\mathcal{D},k](\tau)$ is a finite subset of $\mathcal{D}[k].\mathcal{W}$, equivalently*

$$\Gamma[\mathcal{D},k](\tau) \in \mathcal{P}_{\mathrm{fin}}(\mathcal{D}[k].\mathcal{W}).$$

Proof. We proceed by structural induction on τ. For a nullary term $\tau = a$, the raw output $\Delta_{k,a}(B(\tau))$ is finite; the lifted nullary value also applies the cleaning map, which maps finite witness sets to finite witness sets by definition. For an internal node $a(\tau_1,\dots,\tau_r)$, the induction hypothesis implies that each $\Gamma[\mathcal{D},k](\tau_i)$ is finite. The Cartesian product of finitely many finite sets is finite, each transition $\Delta_{k,a}$ outputs a finite subset of $\mathcal{D}[k].\mathcal{W}$, and the cleaning map maps finite witness sets to finite witness sets by definition. Consequently $\Gamma[\mathcal{D},k](\tau)$ is finite and wholly contained in $\mathcal{D}[k].\mathcal{W}$. □

A term τ is *accepted* by $\mathcal{D}[k]$ if its root witness set contains a witness w such that

$\texttt{Final}_k(B(\tau), w) = 1$. The set of accepted terms is:

$$\mathsf{Acc}(\mathcal{D}[k]) = \left\{ \tau \in \mathcal{T}_k^{\text{bag}} \mid \exists w \in \Gamma[\mathcal{D}, k](\tau),\ \texttt{Final}_k(B(\tau), w) = 1 \right\}.$$

In this way, a DP-core defines a language of accepted terms. To link this to graph properties formally, we carefully distinguish between the acceptance of a term (witness existence) and the property of the graph it represents.

4.3 Graph Property

The language of accepted terms computed by a DP-core becomes meaningful only once it is interpreted through a decomposition class.

Formally, let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k \in \mathbb{N}}$ be a treelike decomposition class as in Definition 12, where each L_k is a set of decomposition terms of width k, and each $\mathcal{G}_k$ maps such a term to a graph. As in Chapter 3, we simply write $\mathcal{G}(\tau)$ when the relevant index is clear from context.

The core $\mathcal{D}$ then defines a property on graphs by accepting exactly those terms whose underlying graphs satisfy the target condition. We capture this precisely below.

Definition 24 (Graph Property of a DP-core)**.** *Let $\mathcal{C}$ be a bag-aware treelike decomposition class in the sense of Definition 22, and let $\mathcal{D}$ be a treelike DP-core such that $\Sigma_k = \mathcal{D}[k].\Sigma$ for every $k \in \mathbb{N}$. For each $k \in \mathbb{N}$, define:*

$$\mathcal{P}[\mathcal{D}[k], \mathcal{C}] = \mathrm{ISO}(\{\mathcal{G}(\tau) \mid \tau \in \mathsf{L}_k \text{ and } \bar{\tau} \in \mathsf{Acc}(\mathcal{D}[k])\}).$$

The graph property induced by $\mathcal{D}$ is then:

$$\mathcal{P}[\mathcal{D}, \mathcal{C}] = \bigcup_k \mathcal{P}[\mathcal{D}[k], \mathcal{C}].$$

Here $\mathrm{ISO}(S)$ denotes the isomorphism-closure of a set of graphs S.

Remark 3 (Notation vs. prior work). In de Oliveira Oliveira and Vadiee [2023], the induced property is denoted by $\mathsf{G}[\mathcal{D}, \mathcal{C}]$. We write $\mathcal{P}[\mathcal{D}, \mathcal{C}]$ to emphasize that this object is a graph property (a set of graphs closed under isomorphism).

This construction provides a modular way of specifying graph properties. Given a decomposition class and a corresponding treelike DP-core, the framework can express

standard decision problems (such as Vertex Cover, 3-Colorability, or Hamiltonian Path) or optimization problems (such as computing the minimum vertex cover size).

The induced property is meaningful as long as the DP-core processes decomposition terms in a manner consistent with their graph semantics. This idea, called coherence, is the subject of the next section.

4.4 Coherence

When defining algorithms over treelike decompositions, it is often the case that a given graph can be represented by several different decomposition terms, even for the same width bound. To ensure the correctness and consistency of the output, a dynamic programming procedure must behave coherently with respect to all such term representations of the same graph.

This leads us to the central notion of *coherence* for DP-cores, which guarantees that different terms representing isomorphic graphs yield the same acceptance outcome, and (for invariant-valued cores) the same computed invariant. Coherence is a crucial requirement for a DP-core to define a well-behaved graph property independent of the chosen decomposition. For invariant-valued cores, the corresponding invariant must also be decomposition-independent. Coherence is also the key condition that lets Chapter 5 reduce width-bounded graph-property inclusion to term-language inclusion.

As in Remark 2, expressions such as $\tau \in \mathsf{Acc}(\mathcal{D}[k])$ for an unannotated term are read via the validated bag-aware representative $\bar{\tau}$.

Definition 25 (Coherence)**.** *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a bag-aware treelike decomposition class, and let $\mathcal{D}$ be a treelike DP-core. We say that $\mathcal{D}$ is $\mathcal{C}$-coherent if the following hold:*

1. *For every $k \in \mathbb{N}$, we have $\Sigma_k = \mathcal{D}[k].\Sigma$.*
2. *For every pair of indices $k, k' \in \mathbb{N}$, and every pair of decomposition terms $\tau \in \mathsf{L}_k$ and $\tau' \in \mathsf{L}_{k'}$ such that*
$$\mathcal{G}(\tau) \simeq \mathcal{G}(\tau'),$$
the following two conditions hold:
   (a) *$\tau \in \mathsf{Acc}(\mathcal{D}[k])$ if and only if $\tau' \in \mathsf{Acc}(\mathcal{D}[k'])$.*

(b) If the invariant component is intended to induce a graph-level invariant (Definition 26), then the computed invariants agree:

$$\mathcal{D}[k].\texttt{Inv}(\Gamma[\mathcal{D},k](\tau)) = \mathcal{D}[k'].\texttt{Inv}(\Gamma[\mathcal{D},k'](\tau')).$$

Coherence ensures that the behavior of the DP-core is well-defined at the graph level, and does not depend on how a graph is represented internally via its decomposition. In particular, acceptance must be invariant under isomorphism and decomposition choice; for invariant-valued cores, the computed invariant must satisfy the same invariance. The next statement records the catalog-level verification proved later; it is a forward reference to the detailed proofs rather than a consequence of the preceding definitions alone.

Proposition 3 (Coherence of the catalog cores)**.** *All canonical DP-cores in Section 4.8 are* ITD*-coherent in the sense of Definition 25, with one caveat: the clique core* $\mathcal{C}^{\omega}_{\ell}$ *is coherent only on the subclass of instructive terms whose realized graphs are simple. Thus, for any* $k, k' \in \mathbb{N}$ *and any* $\tau \in \mathsf{ITD}_k$*,* $\tau' \in \mathsf{ITD}_{k'}$ *with* $\mathcal{G}(\tau) \simeq \mathcal{G}(\tau')$*, acceptance status coincides for every catalog core other than* $\mathcal{C}^{\omega}_{\ell}$*; for invariant-valued cores, the root invariant value also coincides. For* $\mathcal{C}^{\omega}_{\ell}$ *the same conclusion holds whenever the realized graphs are simple, and an unconditional simple-graph acceptance predicate is obtained by guarding with* $\neg$MultiEdge *as in Appendix D and Chapter 8. Proofs for the individual catalog cores are given in Chapter 9 and Appendix D.*

The following proposition formally captures how coherence ensures correctness in determining whether a graph belongs to the class defined by a DP-core.

Proposition 4. *Let* $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ *be a bag-aware treelike decomposition class, and let* $\mathcal{D}$ *be a* $\mathcal{C}$*-coherent treelike DP-core. Then, for every* $k \in \mathbb{N}$ *and every decomposition term* $\tau \in \mathsf{L}_k$*, we have:*

$$\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D},\mathcal{C}] \quad \textit{if and only if} \quad \bar{\tau} \in \mathsf{Acc}(\mathcal{D}[k]).$$

Proof. Let $k \in \mathbb{N}$ and let $\tau \in \mathsf{L}_k$. Suppose first that $\bar{\tau} \in \mathsf{Acc}(\mathcal{D}[k])$. Then by Definition 24, we have $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D}[k],\mathcal{C}] \subseteq \mathcal{P}[\mathcal{D},\mathcal{C}]$.

For the converse, assume that $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D},\mathcal{C}]$. Then there exists some $k' \in \mathbb{N}$ and a decomposition $\tau' \in \mathsf{L}_{k'}$ with $\bar{\tau'} \in \mathsf{Acc}(\mathcal{D}[k'])$ such that

$$\mathcal{G}(\tau') \simeq \mathcal{G}(\tau).$$

Since $\mathcal{D}$ is $\mathcal{C}$-coherent, it follows from Definition 25 that $\bar{\tau} \in \mathsf{Acc}(\mathcal{D}[k])$. □

This proposition confirms that the graph-level property defined by a coherent DP-core can be fully characterized by examining whether a decomposition term is accepted at its corresponding width.

4.4.1 Graph Invariants via Coherent DP-cores

Beyond Boolean properties, coherent DP-cores can define graph invariants: numerical or symbolic quantities associated with graphs, such as sizes of optimal solutions. The next definition packages this information.

Definition 26 (Invariant of a DP-core)**.** *Let $\mathcal{C}$ be a bag-aware treelike decomposition class and $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core satisfying Condition 2b. The $\mathcal{P}[\mathcal{D},\mathcal{C}]$-invariant defined by $\mathcal{D}$ is the function*

$$\mathcal{I}[\mathcal{D},\mathcal{C}] : \mathcal{P}[\mathcal{D},\mathcal{C}] \to \{0,1\}^*$$

that assigns to each graph $G \in \mathcal{P}[\mathcal{D},\mathcal{C}]$ the string

$$\mathcal{D}[k].\texttt{Inv}(\Gamma[\mathcal{D},k](\tau)),$$

where $k \in \mathbb{N}$ and $\tau \in \mathsf{L}_k$ are chosen so that $\mathcal{G}(\tau) \simeq G$.

Condition 2b of Definition 25 ensures that the invariant is well-defined: it is independent of both the chosen decomposition and the ambient width index k, and is stable under graph isomorphism.

4.5 Complexity Measures

To analyze the computational efficiency of treelike DP-cores, we introduce several complexity measures that quantify the space and time requirements of the dynamic programming process. These measures are defined in terms of the size and structure of the witness sets generated by the core during the evaluation of decomposition terms.

Let $\mathcal{D}$ be a treelike DP-core. We consider the behavior of $\mathcal{D}$ on terms of bounded size and decomposition width, and we define four central measures: bitlength, multiplicity, witness-level state complexity, and witness-set state complexity. They coincide with the quantitative invariants introduced in de Oliveira Oliveira and Vadiee [2023, Section 4]

and will later support the complexity guarantees for our model-checking and automated-proving applications.

Definition 27 (($\mathcal{D}, k, n$)-useful Set)**.** *A set $W \subseteq \mathcal{D}[k].\mathcal{W}$ is called $(\mathcal{D}, k, n)$-useful if there exists a term $\tau \in \mathcal{T}_k^{\text{bag}}$ with $|\tau| \leq n$ such that*

$$\Gamma[\mathcal{D}, k](\tau) = W.$$

A set W is $(\mathcal{D}, k, n)$-useful exactly when it appears as the witness set at some node of a width-k term of size at most n. This definition may be restricted to a specific decomposition language when needed.

Definition 28 (Bitlength)**.** *The bitlength of a DP-core $\mathcal{D}$ is the function*

$$\beta_{\mathcal{D}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

defined by

$$\beta_{\mathcal{D}}(k, n) = \max\left(\{\beta(w) \mid w \in W,\ W \text{ is } (\mathcal{D}, k, n)\text{-useful}\} \cup \{0\}\right).$$

This function measures the size, in bits, of the most complex witness that the core may generate for decompositions of size at most n and width k.

Definition 29 (Multiplicity)**.** *The multiplicity of a DP-core $\mathcal{D}$ is the function*

$$\mu_{\mathcal{D}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

defined by

$$\mu_{\mathcal{D}}(k, n) = \max\left(\{|W| \mid W \text{ is } (\mathcal{D}, k, n)\text{-useful}\} \cup \{0\}\right).$$

This function indicates the maximum number of local witnesses that may be needed to describe the dynamic state at any node of a decomposition. By the displayed convention, both bitlength and multiplicity are 0 when no useful witness set exists at the specified (k, n).

Definition 30 (Witness-level State Complexity)**.** *The witness-level state complexity of a DP-core $\mathcal{D}$ is the function*

$$\nu_{\mathcal{D}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

defined by

$$\nu_{\mathcal{D}}(k, n) = \left|\left\{w \in \mathcal{D}[k].\mathcal{W} \;\middle|\; \exists W \ \left(w \in W \text{ and } W \text{ is } (\mathcal{D}, k, n)\text{-useful}\right)\right\}\right|.$$

This function counts the number of distinct local witnesses that can arise in some reachable dynamic table over terms of bounded size and width.

Definition 31 (Witness-Set State Complexity / Deterministic State Complexity)**.** *The witness-set state complexity (also called the deterministic state complexity) of $\mathcal{D}$ is the function*

$$\delta_{\mathcal{D}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

defined by

$$\delta_{\mathcal{D}}(k, n) = |\{W \mid W \text{ is } (\mathcal{D}, k, n)\text{-useful}\}| .$$

This function counts the number of distinct cleaned witness sets that the DP-core may produce over terms of bounded size and width, and therefore measures the size of the deterministic set-level state space explored by dynamization.

Definition 32 (Finite DP-core)**.** *A treelike DP-core $\mathcal{D}$ is said to be finite if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that for all $k, n \in \mathbb{N}$,*

$$\beta_{\mathcal{D}}(k, n) \leq f(k).$$

This notion is intentionally stronger than merely requiring the reachable witness-set state space to be finite at fixed width. We ask for a width-only bound on witness encodings because that immediately yields width-only bounds on the reachable witness universe, table multiplicity, and deterministic state space used in the runtime corollaries below.

Remark 4 (Non-finite (unbounded) DP-cores). Many DP-cores computing numeric invariants are *not* finite: witness encodings contain counters that grow with input size. In such cases $\beta_{\mathcal{D}}(k, n)$ depends on n (often logarithmically), but the core can still be internally polynomial and coherent.

Lemma 3 (Unbounded invariant range forces non-finiteness). *Let $\mathcal{C}$ be a bag-aware treelike decomposition class, and let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core. Assume that $\mathcal{D}$ is correct for an integer-valued graph invariant I in the following sense: for every k and every term $\tau \in \mathsf{L}_k$, the string $\mathcal{D}[k].\mathtt{Inv}(\Gamma[\mathcal{D}, k](\tau))$ is the fixed binary encoding of $I(\mathcal{G}(\tau))$. If there exists a fixed k and a family $(\tau_n)_{n\geq 1} \subseteq \mathsf{L}_k$ such that the integers $I(\mathcal{G}(\tau_n))$ are pairwise distinct, then $\mathcal{D}$ is non-finite in the sense of Definition 32.*

Proof. Assume for contradiction that $\mathcal{D}$ is finite. Then there exists $f : \mathbb{N} \to \mathbb{N}$ such that $\beta_{\mathcal{D}}(k, n) \leq f(k)$ for all n. Fix k and consider any term $\tau \in \mathsf{L}_k$. Since $\Gamma[\mathcal{D}, k](\tau)$ is $(\mathcal{D}, k, |\tau|)$-useful, every witness w occurring in $\Gamma[\mathcal{D}, k](\tau)$ has bitlength at most $f(k)$ by Definition 28. Hence each root witness set $\Gamma[\mathcal{D}, k](\tau)$ is a subset of the finite set

of binary strings of length at most $f(k)$, so only finitely many root witness sets can occur at width k. Since $\mathcal{D}[k].\texttt{Inv}(\cdot)$ is a deterministic function of the root witness set, it follows that $\mathcal{D}[k].\texttt{Inv}(\Gamma[\mathcal{D},k](\tau))$ can take only finitely many distinct values on L_k-terms, contradicting the existence of $(\tau_n)_{n\geq 1}$ with pairwise distinct integers $I(\mathcal{G}(\tau_n))$. □

Examples include maximum independent set and vertex-counting cores, with $\beta(k,n) = O(k+\log n)$ and $\beta(k,n) = O(\log n)$ respectively (Appendix D). Non-finiteness is harmless for fixed-instance evaluation but motivates bounded-size inclusion testing (Section 5.4) and premise-bounded exploration (Chapter 7).

In a finite DP-core, the bitlength of any witness depends only on the decomposition width and is independent of the term size. For convenience we define the width-only envelopes

$$\begin{aligned}\beta_{\mathcal{D}}(k) &:= \sup_{n\in\mathbb{N}} \beta_{\mathcal{D}}(k,n),\\ \nu_{\mathcal{D}}(k) &:= \sup_{n\in\mathbb{N}} \nu_{\mathcal{D}}(k,n),\\ \mu_{\mathcal{D}}(k) &:= \sup_{n\in\mathbb{N}} \mu_{\mathcal{D}}(k,n),\\ \delta_{\mathcal{D}}(k) &:= \sup_{n\in\mathbb{N}} \delta_{\mathcal{D}}(k,n).\end{aligned}$$

If $\mathcal{D}$ is finite, then $\beta_{\mathcal{D}}(k)$ is finite by definition. Consequently there are only finitely many binary strings of length at most $\beta_{\mathcal{D}}(k)$, and in particular $\nu_{\mathcal{D}}(k)$, $\mu_{\mathcal{D}}(k)$, and $\delta_{\mathcal{D}}(k)$ are also finite.

Throughout we fix, once and for all, a concrete encoding of witnesses as binary strings. Formally, for each $k \in \mathbb{N}$ we view $\mathcal{W}_k$ as a decidable subset of $\{0,1\}^*$, and we measure witness bitlength by string length (i.e., $\beta(w) = |w|$ for a witness encoding w). All complexity bounds in this chapter are stated with respect to this encoding.

4.6 Polynomial DP-cores

We now formalize algorithmic tractability for treelike DP-cores. While $\beta_{\mathcal{D}}$ and $\mu_{\mathcal{D}}$ quantify state growth, *internal polynomiality* ensures that each primitive operation (membership checks, transitions, finality tests, invariant computations) runs in polynomial time. Importantly, this does *not* imply efficient full-term evaluation, which depends on bitlength and multiplicity; it merely ensures *local computations* are efficient.

Definition 33 (Internally Polynomial DP-cores)**.** *Let $\mathcal{D}$ be a treelike DP-core. We say that $\mathcal{D}$ is internally polynomial if the following conditions are satisfied:*

1. *For each $k \in \mathbb{N}$, and each $(\mathcal{D}, k, n)$-useful set W, the bitlength of the output of the invariant function is bounded by a polynomial in the width and the encoded size of the useful witness set:*

$$|\mathcal{D}[k].\texttt{Inv}(W)| \leq (1 + k + \beta_{\mathcal{D}}(k, n) \cdot \max\{1, \mu_{\mathcal{D}}(k, n)\})^{O(1)}.$$

   Here $|\cdot|$ denotes the length of the output string produced by `Inv`*.*

2. *There exists a deterministic algorithm $\mathfrak{A}$ such that the following tasks can be performed in polynomial time (in terms of k and input size):*

   (a) *Given $k \in \mathbb{N}$ and a binary string $w \in \{0,1\}^*$, the algorithm decides whether $w \in \mathcal{D}[k].\mathcal{W}$ in time $(k + |w|)^{O(1)}$.*

   (b) *Given $k \in \mathbb{N}$, a bag $B \subseteq [k+1]$, and a nullary symbol $a \in \Sigma_k$, the algorithm constructs the finite set $\Delta_{k,a}(B)$ in time $(k + \beta_{\mathcal{D}}(k, 1) \cdot \mu_{\mathcal{D}}(k, 1))^{O(1)}$. This is an explicit output-size assumption on the raw nullary transition, before the cleaning map is applied.*

   (c) *Given $k \in \mathbb{N}$, a bag $B \subseteq [k+1]$, a symbol $a \in \Sigma_k$ of arity r, and a tuple $X \in \mathcal{D}[k].\mathcal{W}^r$, the algorithm computes the output set $\Delta_{k,a}(B, X)$ in time polynomial in k and the total encoding length of (B, X).*

   (d) *Given $k \in \mathbb{N}$, a function name* `Function` $\in \{$`Final`, `Clean`, `Inv`$\}$*, and an input X to that function (where $X = (B, w)$ for* `Final` *and X is a witness set for* `Clean` *and* `Inv`*), the algorithm computes $\mathcal{D}[k].$`Function`(X) in time polynomial in k and the total encoding length of X.*

This formalization ensures that, for fixed k, all core functions are accessible via a single deterministic algorithm $\mathfrak{A}$ that operates efficiently. In practice, such cores arise naturally when designing dynamic programming routines where the local solution representations (witnesses) have bounded size, and the transitions and invariant computations involve only combinatorial data like sets, tuples, or small tables.

Internal polynomiality is thus a key requirement for using DP-cores in automated reasoning or algorithmic applications, where tractability depends not only on correctness but also on efficient evaluation of subcomputations.

4.7 DP-core evaluation on a fixed decomposition term

This section concerns the complexity of evaluating a treelike DP-core on a fixed decomposition term—i.e., the standard per-instance dynamic programming mode (model checking and invariant computation). The underlying evaluation pattern is classical in bounded-width algorithms; the role of this section is to make the dependence on the DP-core witness parameters explicit, because the same parameters later control the state space explored in width-bounded automated theorem proving (Chapter 5).

We express explicit running-time bounds in terms of the term size $n = |\tau|$, the maximum constructor arity r, and the witness complexity measures $\beta_{\mathcal{D}}(k, n)$ and $\mu_{\mathcal{D}}(k, n)$.

Since correctness is handled separately via the DP-core proof obligations of Chapter 9, we focus here only on the runtime accounting: running time is governed by the growth and representation cost of witness sets along the term.

This section first records a generic runtime bound for evaluating an internally polynomial DP-core bottom-up on a decomposition term, then specializes it to finite cores and to instructive decompositions, and finally points forward to width-based automated theorem proving.

For terms of size at most n and width k, we write $\beta_{\mathcal{D}}(k, n)$ for the *bitlength* parameter of $\mathcal{D}$ and $\mu_{\mathcal{D}}(k, n)$ for its *multiplicity* parameter (Definition 28 and Definition 29). Concretely, $\beta_{\mathcal{D}}(k, n)$ is the maximum bitlength of a witness appearing in any $(\mathcal{D}, k, n)$-useful set, and $\mu_{\mathcal{D}}(k, n)$ is the maximum size of such a useful set. For finite cores these bounds depend only on k, and we drop the n parameter.

In the instructive setting, a term may end with a non-empty root bag. When convenient, we may close the root bag by appending forget operations for the remaining active labels; this produces a *root-closed* term (empty root bag) and does not change the realized graph, only the interface exposed at the root. For coherent DP-cores, acceptance is independent of such root-bag closure. If the core also satisfies Condition 2b, then the induced invariant value is independent of the chosen root closure as well.

4.7.1 Model Checking and Invariant Computation

Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k \in \mathbb{N}}$ be a bag-aware treelike decomposition class of arity r, and let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core. Given a term $\tau \in \mathsf{L}_k$ of size $n = |\tau|$, together with

the bag annotation of every subterm, let

$$\overline{\tau} \in \mathcal{T}_k^{\mathrm{bag}}$$

denote the resulting validated bag-aware representative. Since Definition 23 is formally stated on bag-aware terms, we write

$$\Gamma[\mathcal{D}, k](\tau) \quad \text{as shorthand for} \quad \Gamma[\mathcal{D}, k](\overline{\tau}).$$

With this convention, the dynamization introduced in Definition 23 computes, for each node of the term, a set of witnesses that summarize the partial solutions represented by the corresponding subterm.

In particular, model checking on the fixed input term τ reduces to computing the root witness set $\Gamma[\mathcal{D}, k](\tau)$ and returning YES iff it contains some witness w with $\texttt{Final}_k(B(\tau), w) = 1$. Invariant computation is identical except that, once the root witness set is available, we output $\texttt{Inv}_k(\Gamma[\mathcal{D}, k](\tau))$. When $\mathcal{D}$ also satisfies Condition 2b and the represented graph lies in the induced property, this root value agrees with the graph-level invariant of Definition 26.

The following lemma isolates the work performed at a single node.

Lemma 4 (Per-node evaluation)**.** *Let $\mathcal{D}$ be an internally polynomial treelike DP-core, and let $a(\tau_1, \ldots, \tau_s)$ be a node labeled a, where $s = \mathsf{r}(a)$ (so $0 \le s \le r$). Assume that the witness sets $\Gamma[\mathcal{D}, k](\tau_i)$ have already been computed and that the bag of this node is available from the input annotation. The cost of producing the witness set at a is bounded by*

$$k^{O(1)} \cdot r^{O(1)} \cdot (1 + \beta_{\mathcal{D}}(k, n))^{O(1)} \cdot (1 + \mu_{\mathcal{D}}(k, n))^{O(\max\{s,1\})}.$$

Proof. By Definition 23, the algorithm considers every tuple of witnesses $(w_1, \ldots, w_s)$ drawn from the child witness sets. When $s = 0$ there is exactly one tuple (the empty tuple). In this nullary case, by internal polynomiality (Definition 33, item 2(b)), the algorithm constructs the raw output in time polynomial in $k + \beta_{\mathcal{D}}(k, 1)\mu_{\mathcal{D}}(k, 1)$, which is bounded by the displayed expression because $n \ge 1$ and the useful-set parameters are monotone in the size bound. Cleaning the raw nullary output runs in time polynomial in its encoded size by item 2(d), so it is covered by the same bound.

Assume now that $s > 0$. There are at most $\mu_{\mathcal{D}}(k, n)^s$ witness tuples. For each tuple we invoke the transition function $\Delta_{k,a}$ once. Internal polynomiality (Definition 33, item 2(c)) guarantees that this call runs in time polynomial in k and the total encoding length of the tuple.

Moreover, each τ_i is a subterm of the input term of size n, hence $|\tau_i| \leq n$ and $\Gamma[\mathcal{D}, k](\tau_i)$ is $(\mathcal{D}, k, n)$-useful. In particular, each w_i lies in a $(\mathcal{D}, k, n)$-useful set, so $\beta(w_i) \leq \beta_{\mathcal{D}}(k, n)$ and hence

$$\sum_i \beta(w_i) \leq s \cdot \beta_{\mathcal{D}}(k, n) \leq r \cdot \beta_{\mathcal{D}}(k, n).$$

Therefore each transition call costs at most

$$k^{O(1)} \cdot r^{O(1)} \cdot (1 + \beta_{\mathcal{D}}(k, n))^{O(1)}.$$

Since an algorithm cannot output more bits than it writes in its running time, the encoded transition output for each tuple has the same polynomial bound on its total size. Concatenating the outputs over all witness tuples therefore produces a raw union of total encoding size at most

$$k^{O(1)} \cdot r^{O(1)} \cdot (1 + \beta_{\mathcal{D}}(k, n))^{O(1)} \cdot (1 + \mu_{\mathcal{D}}(k, n))^{s}.$$

Turning that concatenation into an explicit finite witness set only requires duplicate elimination and set bookkeeping on that representation, hence runs in time polynomial in the same total size. Finally, internal polynomiality (Definition 33, item 2(d)) applies to the cleaning function on this witness-set input, so the cleaning step contributes only an additional polynomial factor in the raw-union size. Since $s > 0$, taking a fixed polynomial power changes this contribution to $(1 + \mu_{\mathcal{D}}(k, n))^{O(s)}$. Absorbing the remaining polynomial factors yields the claimed bound

$$k^{O(1)} \cdot r^{O(1)} \cdot (1 + \beta_{\mathcal{D}}(k, n))^{O(1)} \cdot (1 + \mu_{\mathcal{D}}(k, n))^{O(\max\{s,1\})}.$$

□

With the node-wise bound in hand, we recover the global result. This mirrors Theorem 4.4 in [de Oliveira Oliveira and Vadiee, 2023].

Theorem 4 (Model checking and invariant computation)**.** *Let $\mathcal{C}$ be a bag-aware treelike decomposition class of arity r, and let $\mathcal{D}$ be an internally polynomial, $\mathcal{C}$-coherent treelike DP-core. For every term $\tau \in \mathsf{L}_k$ of size n, together with the bag annotation of each subterm, the following hold.*

1. *One can compute the root witness set $\Gamma[\mathcal{D}, k](\tau)$ and decide whether*

$$\tau \in \mathsf{Acc}(\mathcal{D}[k])$$

in time

$$T(k,n) = n \cdot k^{O(1)} \cdot r^{O(1)} \cdot (1+\beta_{\mathcal{D}}(k,n))^{O(1)} \cdot (1+\mu_{\mathcal{D}}(k,n))^{O(\max\{r,1\})}.$$

Consequently, the same procedure decides whether $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D},\mathcal{C}]$ within the same time bound.

2. *If, in addition, $\mathcal{D}$ satisfies Condition 2b and*

$$\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D},\mathcal{C}],$$

then one can compute the invariant $\mathcal{I}[\mathcal{D},\mathcal{C}](\mathcal{G}(\tau))$ in time

$$T(k,n) + k^{O(1)} \cdot (1+\beta_{\mathcal{D}}(k,n))^{O(1)} \cdot (1+\mu_{\mathcal{D}}(k,n))^{O(1)}.$$

Proof. For fixed-term evaluation, the supplied bag annotation turns τ into the bag-aware representative discussed above, so evaluating $\mathcal{D}[k]$ bottom-up yields the root witness set $\Gamma[\mathcal{D},k](\tau)$ by Definition 23. By the definition of acceptance, $\tau \in \mathsf{Acc}(\mathcal{D}[k])$ holds if and only if this set contains a final witness, proving the first sentence of item 1. If $\mathcal{D}$ is $\mathcal{C}$-coherent, then Proposition 4 upgrades this term-level acceptance test to the graph-level statement

$$\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D},\mathcal{C}] \quad \Longleftrightarrow \quad \tau \in \mathsf{Acc}(\mathcal{D}[k]),$$

which yields the final sentence of item 1. If $\mathcal{D}$ additionally satisfies Condition 2b and $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D},\mathcal{C}]$, then τ itself is a valid representative of $\mathcal{G}(\tau)$ at width k, so Definition 26 gives

$$\mathcal{I}[\mathcal{D},\mathcal{C}](\mathcal{G}(\tau)) = \mathtt{Inv}_k(\Gamma[\mathcal{D},k](\tau)).$$

Here no minimality assumption on k is needed: Definition 26 is independent of the chosen representative width by Condition 2b.

For runtime, there are at most n nodes in the term, each of arity at most r, and the bag annotation is read directly from the input representation; the cost of validation or bag reconstruction is therefore outside the present bound. Applying Lemma 4 to each node and summing the bounds yields the cost of computing the root witness set. Deciding acceptance then scans at most $\mu_{\mathcal{D}}(k,n)$ root witnesses and evaluates $\mathtt{Final}_k(B(\tau),w)$ on each one. By internal polynomiality (Definition 33, item 2(d)), this scan costs

$$k^{O(1)} \cdot (1+\beta_{\mathcal{D}}(k,n))^{O(1)} \cdot (1+\mu_{\mathcal{D}}(k,n))^{O(1)},$$

which is absorbed by $T(k,n)$. This proves item 1. For item 2 we additionally invoke the invariant map $\mathtt{Inv}_k$ at the root. Internal polynomiality bounds this call by $k^{O(1)} \cdot$

$(1+\beta_{\mathcal{D}}(k,n))^{O(1)} \cdot (1+\mu_{\mathcal{D}}(k,n))^{O(1)}$, which we add to $T(k,n)$. □

Intuitively, the factor $(1+\mu_{\mathcal{D}}(k,n))^{O(\max\{r,1\})}$ comes from enumerating witness tuples across children at each node and then applying polynomial-time set bookkeeping and cleaning to the resulting raw union, with the exponent 1 covering nullary transition output size. The dependence on $\beta_{\mathcal{D}}(k,n)$ captures the cost of operating on witness encodings (transition evaluation, membership/comparison work, and cleaning). For instructive decompositions, where $r = 2$ is fixed, the combinatorial contribution is polynomial in the maximal witness-set size.

Theorem 4 makes the dependence on the structural parameters explicit. Two useful specializations appear frequently in practice, starting with the case in which the core itself is finite.

Corollary 1 (Finite DP-cores)**.** *Let $\mathcal{C}$ be a bag-aware treelike decomposition class of arity r, and let $\mathcal{D}$ be a finite (Definition 32), internally polynomial, $\mathcal{C}$-coherent treelike DP-core. Suppose that there exist functions $\beta, \mu : \mathbb{N} \to \mathbb{N}$ with $\beta_{\mathcal{D}}(k,n) \leq \beta(k)$ and $\mu_{\mathcal{D}}(k,n) \leq \mu(k)$ for all n. Then, for every validated term $\tau \in \mathsf{L}_k$ of size n, model checking runs in time*

$$n \cdot k^{O(1)} \cdot r^{O(1)} \cdot (1+\beta(k))^{O(1)} \cdot (1+\mu(k))^{O(\max\{r,1\})}.$$

Proof. Substitute the bounds on bitlength and multiplicity into Theorem 4. □

Remark 5 (Model checking with non-finite cores). Non-finite DP-cores (Remark 4) are still covered by the generic bound of Theorem 4: the dependence on the term size appears only through $\beta_{\mathcal{D}}(k,n)$ and $\mu_{\mathcal{D}}(k,n)$. In particular, when $\beta_{\mathcal{D}}(k,n) = O(\log n)$ and $\mu_{\mathcal{D}}(k,n)$ depends only on k, the per-instance evaluation time is polynomial in n (with the polynomial degree depending only on k and the arity bound). For example, the vertex-counting core satisfies $\beta(k,n) = O(\log n)$ and $\mu(k,n) = 1$ (Theorem 44), and the maximum independent set core satisfies $\beta(k,n) = O(k + \log n)$ and $\mu(k,n) \leq 2^{k+1}$ (Theorem 42), so both admit efficient fixed-instance evaluation at bounded width.

Specializing to instructive decompositions gives the following bound for the main class studied in this thesis.

Corollary 2 (Instructive decompositions)**.** *Let $\mathcal{C} = \mathsf{ITD}$ be the instructive tree decomposition class of Section 3.4 (Definition 19). Then $r = 2$. Consequently, any internally polynomial, ITD-coherent treelike DP-core can be used for graph-level model checking on a validated instructive term (with its bag annotation) in time*

$$n \cdot k^{O(1)} \cdot (1+\beta(k))^{O(1)} \cdot (1+\mu(k))^{O(1)},$$

whenever β and μ bound the witness bitlength and multiplicity, respectively. Without coherence, the same displayed expression still bounds fixed-term computation of the root witness set and term acceptance, but it does not imply a graph-level property decision. The exponent is an absolute constant because instructive decompositions have binary maximum arity. For the chromatic-number cores of de Oliveira Oliveira and Vadiee [2023] we have $\beta(k) = O(k \log k)$ and $\mu(k) = k^{O(k)}$, leading to the $n \cdot k^{O(k)}$ complexity reported in the paper.

Proof. The instructive alphabet contains a nullary leaf, unary instructions, and the binary join, hence its maximum arity is $r = 2$. Substituting this into Theorem 4 and using the width-only bounds $\beta_{\mathcal{D}}(k, n) \leq \beta(k)$ and $\mu_{\mathcal{D}}(k, n) \leq \mu(k)$ for all n yields the stated running time. The bounds on β and μ for the chromatic-number cores come from the witness-encoding analysis developed in [de Oliveira Oliveira and Vadiee, 2023, Section 5]. □

To illustrate these bounds on a concrete instance, we record the behavior of the 2-colorability core on width-one terms.

Lemma 5 (Chromatic-number core on width-one terms)**.** *Let $k \leq 1$ and let $\tau \in \mathrm{ITD}_k$. The evaluation of the partition-presentation core $\mathcal{C}_2^{\chi}$ on τ runs in time $O(|\tau|)$.*

Proof. For $k \leq 1$, every current bag is a subset of $[2]$. A witness of $\mathcal{C}_2^{\chi}$ is therefore a 2-partition of a subset of $[2]$, so only constantly many witnesses can occur (namely the empty partition, the two one-element partitions, the two-element singleton partition, and the one-block partition of $\{1, 2\}$). Hence both the witness bitlength and the multiplicity are bounded by constants. The partition transitions are directly computable in constant time at this width; equivalently, Lemma 31 proves internal polynomiality for the labeled presentation and Lemma 28 with Remark 19 transfers the same constant-width bounds to the partition presentation. For the fixed-term evaluation claimed in this lemma, the fixed-term part of Corollary 2 applies without needing graph-level coherence; if one wants the corresponding graph-level decision statement, Proposition 8 supplies the required acceptance invariance. Substituting these bounds for $k \leq 1$ yields the linear-time bound. □

This section records the generic runtime bound for fixed-term DP-core evaluation (model checking and invariant computation), making the dependence on witness bitlength and multiplicity explicit and yielding clean specializations for finite cores and for instructive decompositions. Chapter 5 turns the perspective around and studies universality and inclusion questions over all bounded-width terms via DP-refutations.

4.8 Canonical DP-core Catalog

To ground the abstract definitions above, we record the canonical treelike DP-cores that underpin the remainder of the thesis and its applications. All cores considered in this section are coherent with the instructive tree decomposition class ITD introduced in Chapter 3, under the graph semantics stated in their specifications. Since ITD terms may realize multigraphs, comparisons with simple-graph problem statements or SAT encodings are made only under the stated simplicity promises or after conjoining the standard no-multiple-edge guard; in particular, the clique core is used in the simple-graph regime noted in Proposition 3.

Catalog scope. We present chromatic number / c-colorability in full detail; remaining cores are listed as a catalog index with correctness proofs in Chapter 9 and Appendix D.

For notational convenience we introduce the following shorthands together with the predicates defined later in the section. For a k-instructive decomposition τ, we continue to write $B(\tau)$ for its top bag and $\theta[\tau] : B(\tau) \to V(\mathcal{G}(\tau))$ for the associated labeling.

Convention 1 (Catalog format (DP-core specification template)). To keep specifications precise and uniform, fully expanded entries in this chapter follow the checklist below. In the current chapter this template is instantiated in detail for $\mathcal{C}_c^{\chi}$; the remaining catalog entries point to Chapter 9 or Appendix D for full transition-level proofs.

1. **Scope:** state whether the core is intended for all terms (arbitrary root bag) or only for *root-closed* terms (empty root bag). Unless explicitly stated otherwise, we follow the prior papers' default convention and work over all root bags; root closure is used only as a presentation/normalization device for unboundaried graphs.
2. **Core interface (6-tuple):** explicitly state the 6-tuple

$$\mathcal{D}[k] = (\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k).$$

   (As in Definition 21, $\texttt{Final}_k$ and $\Delta_{k,a}$ take the current bag $B \subseteq [k{+}1]$ as an explicit argument; we suppress B in specifications when the corresponding rule is bag-independent.)
3. **Witness universe:** define $\mathcal{W}_k$ (sets, maps, tuples), with intended meaning and bitlength bound if applicable. We specify witnesses at a mathematical level, and we assume a fixed binary encoding that matches the interface of Section 4.1.
4. **Finality, cleaning, and invariant:** give closed-form definitions of $\texttt{Final}_k(B, \cdot)$, $\texttt{Clean}_k$, and $\texttt{Inv}_k$ in terms of the witness structure.

5. **Transitions:** for each symbol a in the alphabet, define the transition function $\Delta_{k,a}(B, \cdot)$ as a function of the current bag B and the child witnesses, returning a finite set of witnesses. Specify agreement conditions on the shared bag at joins. We write `LeafSet` for the nullary leaf transition output.
6. **Semantic predicate / interpretation predicate:** define a predicate $\mathrm{Interp}_{\mathcal{D}}(\tau, w)$ characterizing exactly the witnesses produced by $\Gamma[\mathcal{D}, k](\tau)$. For historical continuity, some concrete cores below keep shorter predicate names such as $\mathsf{P}^{\chi}_{c,k}$; these are concrete instances of the same generic interpretation notion.
7. **Correctness:** state the equivalence lemma $w \in \Gamma[\mathcal{D}, k](\cdot) \Leftrightarrow \mathrm{Interp}_{\mathcal{D}}(\cdot)$ and the acceptance theorem relating the core to its graph property or invariant. Proofs are deferred to Chapter 9, with additional catalog instantiations in Appendix D.
8. **Complexity (optional summary):** record bitlength, multiplicity, witness-level state complexity, and witness-set state complexity bounds in terms of k and relevant parameters.

Convention 2 (Transition notation). Throughout the catalog we adopt common notational conventions so that transition maps can be specified succinctly and uniformly.

- **Label universe.** For width $k \in \mathbb{N}$ the active-label universe is $[k{+}1] = \{1, \ldots, k{+}1\}$.
- **Partial maps.** If f is a partial map then $\mathrm{dom}(f)$ denotes its domain, $f|_X$ its restriction to X, and $f[a \mapsto b]$ the map f with the (possibly new) assignment $a \mapsto b$. We write $\emptyset$ for the empty map (the unique map with empty domain). When two partial maps f, g agree on $\mathrm{dom}(f) \cap \mathrm{dom}(g)$, we write $f \cup g$ for their union (pointwise agreement is always stated or enforced explicitly in transitions).
- **Join bag.** At a join node, we write B for the shared bag (the top bag of both children). For two objects x, y, we write $x \equiv_B y$ to mean that x and y agree on the information they carry about B (e.g., equal colorings on B, equal subsets of B).
- **Pairs of labels.** We write $\binom{X}{2}$ for the set of 2-element subsets of X. For $F \subseteq \binom{[k+1]}{2}$ and $u \in [k{+}1]$ we use $F \setminus \{\{u, w\} : w \in [k{+}1]\}$ to delete all pairs incident to u.
- **Agreement guards.** All join transitions include an implicit guard $x \equiv_B y$ and return the empty set otherwise.
- **Adjacency guards.** Phrases such as "u is non-adjacent to every $x \in A$" refer to the current partial graph realized by the subterm. Operationally, this is enforced by the introduced-edge transitions that have been applied so far.

4.8.1 c-Colorability

We define a DP-core that decides whether the realized graph is c-colorable.

4.8.1.1 Core specification

Definition 34 (c-colorability DP-core)**.** *Fix $c \in \mathbb{N}$ with $c \geq 1$. The sequence $\mathcal{C}_c^\chi = \{\mathcal{C}_c^\chi[k]\}_{k\in\mathbb{N}}$ is defined as follows. For each $k \in \mathbb{N}$, the component*

$$\mathcal{C}_c^\chi[k] = (\Sigma_k, \mathcal{W}_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)$$

is given by:

(i) Σ_k *equal to the instructive alphabet of width k (Definition 16).*

(ii) $\mathcal{W}_k$ *the set of c-partitions of subsets of $[k+1]$, i.e. partitions with at most c cells.*

(iii) $\texttt{Final}_k$ *constantly 1 for every current bag (every witness is final).*

(iv) $\texttt{Clean}_k$ *is the identity, and* $\texttt{Inv}_k$ *is the constant map sending every witness set to the empty string ε (this invariant output is unused for the decision core $\mathcal{C}_c^\chi$).*

(v) Δ_k *is the following bag-indexed total family. For a partition $\mathcal{P}$, write* $\mathrm{supp}(\mathcal{P})$ *for the union of its cells and*

$$\mathcal{P}|_X := \{C \cap X \mid C \in \mathcal{P},\ C \cap X \neq \emptyset\}.$$

For labels $u, v \in [k+1]$ and partitions $\mathcal{P}, \mathcal{P}'$, the transition values are:

(a) $\Delta_{k,\mathit{leaf}}(B) = \begin{cases} \{\emptyset\}, & \text{if } B = \emptyset, \\ \emptyset, & \text{otherwise.} \end{cases}$

(b) For introduce-vertex symbols, let

$$\mathrm{Ins}_u(\mathcal{P}) := \{(\mathcal{P} \setminus \{C\}) \cup \{C \cup \{u\}\} \mid C \in \mathcal{P}\} \cup \begin{cases} \{\mathcal{P} \cup \{\{u\}\}\}, & \text{if } |\mathcal{P}| < c, \\ \emptyset, & \text{if } |\mathcal{P}| = c. \end{cases}$$

Then

$$\Delta_{k,\mathit{intro\_u}}(B, \mathcal{P}) = \begin{cases} \mathrm{Ins}_u(\mathcal{P}), & \text{if } u \in B \text{ and } \mathrm{supp}(\mathcal{P}) = B \setminus \{u\}, \\ \emptyset, & \text{otherwise.} \end{cases}$$

(c) $\Delta_{k,\texttt{forget\_}u}(B,\mathcal{P}) = \begin{cases} \{\mathcal{P}|_B\}, & \text{if } u \notin B \text{ and } \operatorname{supp}(\mathcal{P}) = B \cup \{u\}, \\ \emptyset, & \text{otherwise.} \end{cases}$

(d)

$$\Delta_{k,\texttt{intro\_}u\texttt{\_}v}(B,\mathcal{P}) = \begin{cases} \{\mathcal{P}\}, & \text{if } u,v \in B,\ u \neq v,\ \operatorname{supp}(\mathcal{P}) = B, \\ & \text{and } u,v \text{ lie in distinct cells of } \mathcal{P}, \\ \emptyset, & \text{otherwise.} \end{cases}$$

(e) $\Delta_{k,\texttt{join}}(B,\mathcal{P},\mathcal{P}') = \begin{cases} \{\mathcal{P}\}, & \text{if } \operatorname{supp}(\mathcal{P}) = \operatorname{supp}(\mathcal{P}') = B \text{ and } \mathcal{P} = \mathcal{P}', \\ \emptyset, & \text{otherwise,} \end{cases}$

4.8.1.2 Intuition

A witness records only which active labels are forced to share a color, not the names of the colors themselves. Thus witnesses are represented by c-partitions of the current bag. The transitions implement the standard nice-decomposition updates on these partitions: `intro_`u branches over the choice of the color class receiving u (or a new singleton class if fewer than c classes are currently used), `forget_`u removes u from the partition, and `intro_`u`_`v rejects witnesses in which u and v already lie in the same cell. At joins, two witnesses are compatible exactly when they induce the same partition of the shared bag. Since every witness is final, acceptance reduces to checking that the root witness set is non-empty. For the systematic correctness proofs in Chapter 9, we use a labeled total-map formulation of the same chromatic core and write the generic semantic judgment as $\mathsf{Interp}_{\mathcal{C}_c^\chi}$ rather than as a catalog-specific predicate name; Lemma 28 there records the translation between the two witness presentations.

4.8.1.3 Semantic predicate

Definition 35 (Chromatic witnesses)**.** *Let $c,k \in \mathbb{N}$, let $\tau \in \mathsf{ITD}_k$, and let w be a c-partition of a subset of $[k+1]$. We say that $\mathsf{P}^{\chi}_{c,k}(\tau,w)$ holds when the following are true.*

(i) The union of the cells of w is exactly $B(\tau)$.

(ii) There exists a proper c-coloring $\alpha : V(\mathcal{G}(\tau)) \to [c]$ such that, for every $u,v \in B(\tau)$, the labels u and v lie in the same cell of w if and only if $\alpha(\theta[\tau](u)) = \alpha(\theta[\tau](v))$.

Lemma 6. *For every integer $c \geq 1$, every $k \in \mathbb{N}$, every $\tau \in \mathsf{ITD}_k$, and every c-partition*

w of a subset of $[k+1]$, we have

$$w \in \Gamma[\mathcal{C}_c^\chi, k](\tau) \quad \Longleftrightarrow \quad \mathsf{P}_{c,k}^\chi(\tau, w).$$

The proof of Lemma 6 follows from the total-map correctness theorem in Chapter 9, Theorem 20, together with the partition/total-map translation in Lemma 28.

4.8.1.4 Correctness

Theorem 5 (Correctness of $\mathcal{C}_c^\chi$)**.** *Let $c \geq 1$, let $k \in \mathbb{N}$, and let $\tau \in \mathsf{ITD}_k$. Then*

$$\mathcal{G}(\tau) \text{ is } c\text{-colorable} \quad \Longleftrightarrow \quad \tau \in \mathsf{Acc}(\mathcal{C}_c^\chi[k]).$$

The proof of Theorem 5 follows from Lemma 6 and the fact that every chromatic witness is final; equivalently, see Theorem 21 together with Lemma 28 in Chapter 9.

4.8.2 Catalog index (pointers)

Table 4.1 summarizes the additional DP-cores used later in the thesis and points to the chapters/appendices containing their full transition systems and correctness instantiations.

Beyond the fully worked chromatic-number core above, the thesis uses additional canonical DP-cores (degree threshold, clique detection under a simplicity promise, multi-edge detection, vertex cover and feedback vertex set, maximum independent set, vertex counting, perfect matching, and Hamiltonian path/cycle) as reusable components in later chapters. To avoid duplicating substantial catalog material in Chapter 4, we do not restate their transition systems and correctness/coherence instantiations here; Table 4.1 provides the navigation map, and full specifications are collected in Chapter 9 and Appendix D.

Catalog anchors. The labels above provide cross-reference targets for the catalog rows of Table 4.1. They land at the catalog index above and direct the reader to Chapter 9 or Appendix D for the full specification of each component (max-degree, clique under a simplicity promise, multi-edge, vertex cover, feedback vertex set, independent set, vertex count, perfect matching, Hamiltonian path, and Hamiltonian cycle).

DP-core	Property / Invariant	Full specification/proof location
$\mathcal{C}_d^{\Delta}$	Maximum degree at least d	Appendix D.4
$\mathcal{C}_\ell^{\omega}$ (simple-graph regime)	Clique of size at least ℓ	Appendix D.5
$\mathcal{C}^{\mathrm{ME}}$	Contains a multi-edge	Appendix D.6
$\mathcal{C}^{\mathrm{PM}}$	Has a perfect matching	Appendix D.7
$\mathcal{C}^{\mathrm{HP}}$	Has a Hamiltonian path	Appendix D.8
$\mathcal{C}^{\mathrm{HC}}$	Has a Hamiltonian cycle	Appendix D.9
$\mathcal{C}_t^{\mathrm{FVS}}$	Feedback vertex set number at most t	Appendix D.10
$\mathcal{C}_t^{\mathrm{VC}}$	Vertex cover number at most t	Appendix D, Section D.12
$\mathcal{C}^{\alpha}$	Maximum independent-set size	Appendix D, Section D.13
$\mathcal{C}^{\mathrm{VCnt}}$	Number of vertices	Appendix D, Section D.14

Table 4.1: Index of catalog DP-cores beyond the fully worked chromatic core. The clique row applies under a simplicity promise on the realized graph; see Convention 3 in Appendix D.

Part IV

Width-Based Automated Theorem Proving

Chapter 5

Width-Based Automated Theorem Proving and Double-Exponential Upper Bounds

Having analyzed how DP-cores evaluate fixed decomposition terms (Section 4.7), we now address the universal question: whether *all* terms of a decomposition class are accepted, i.e., whether every bounded-width graph satisfies a target property. This is the width-based automated theorem proving problem studied in de Oliveira Oliveira and Vadiee [2023] and refined in de Oliveira Oliveira and Vadiee [2024]. The key notions are language inclusion, DP-refutations, and structural bounds on counterexample size.

The double-exponential bounds proved here are *completeness guarantees* for fixed-width search, not practical runtime predictions. Chapters 6 and 7 explain the search reductions that make the framework usable in practice.

Root bags and closure. In the underlying papers, and throughout the framework unless stated otherwise, we work over the full instructive term language ITD_k and do not impose a closure condition at the root. Equivalently, in the instructive setting we take the ITD_k recognizer automaton to accept in every state (final-state set $F_k = Q_k = \mathcal{P}([k{+}1])$), so refutations and counterexamples may occur at any bag $b \subseteq [k{+}1]$.

When discussing plain (unboundaried) graphs it is sometimes convenient to present terms in a *root-closed* normal form with empty root bag. This is always possible: given any $\tau \in \mathrm{ITD}_k$ with root bag $B(\tau)$, we can append forget constructors at the root to obtain a root-closed term $\mathrm{close}(\tau) \in \mathrm{ITD}_k$ with $B(\mathrm{close}(\tau)) = \emptyset$ and $\mathcal{G}(\mathrm{close}(\tau)) \simeq \mathcal{G}(\tau)$. Since root-bag closure does not change the realized graph, coherence implies that acceptance

and invariant values are preserved under this postprocessing. We therefore state results and algorithms for arbitrary root bags, and use root closure only as a presentation device when exporting concrete counterexamples.

5.1 Inclusion Test

A central task in width-based automated theorem proving is determining whether a graph property $\mathcal{P}$ is included in another property $\mathcal{P}'$, that is, whether $\mathcal{P} \subseteq \mathcal{P}'$. In the remainder we focus on the case where the domain property is the bounded-width class induced by a treelike decomposition class, and the target property is defined by a DP-core (or a composition of DP-cores) over the same decomposition class. Concretely, the property $\mathcal{P}[\mathcal{C}]$ represents all graphs that admit a decomposition term in a decomposition class $\mathcal{C}$, while $\mathcal{P}[\mathcal{D}, \mathcal{C}]$ denotes the set of all graphs accepted by a given DP-core $\mathcal{D}$ when applied over the same class of decompositions.

Testing whether $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$ asks whether *every* width-k graph realizable by $\mathcal{C}$ satisfies the property induced by $\mathcal{D}$. Because $\mathcal{D}$ is coherent with respect to $\mathcal{C}$, this semantic question can be checked at the level of validated terms: one asks whether $\mathcal{D}[k]$ accepts *all* well-formed width-k decomposition terms in L_k.

Because these properties are parameterized by a width bound $k \in \mathbb{N}$, we adopt a width-indexed approach: for each k, we test whether

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}].$$

This task becomes more tractable thanks to a key property of coherent DP-cores. If $\mathcal{D}$ is coherent with respect to $\mathcal{C}$, then the width-k inclusion question reduces to inclusion at the term-language level:

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}] \quad \Leftrightarrow \quad \mathsf{L}_k \subseteq \mathsf{Acc}(\mathcal{D}[k]).$$

The following lemma makes this reduction precise, turning inclusion of graph classes into a language-theoretic question about terms accepted by DP-cores.

Lemma 7. *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k \in \mathbb{N}}$ be a bag-aware treelike decomposition class, and let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core. Then, for every $k \in \mathbb{N}$,*

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}] \quad \textit{if and only if} \quad \mathsf{L}_k \subseteq \mathsf{Acc}(\mathcal{D}[k]).$$

Proof. Fix $k \in \mathbb{N}$. Since $\mathcal{D}$ is $\mathcal{C}$-coherent, Proposition 4 yields, for every $\tau \in \mathsf{L}_k$,

$$\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D}, \mathcal{C}] \quad \text{if and only if} \quad \tau \in \mathsf{Acc}(\mathcal{D}[k]).$$

If $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$ and $\tau \in \mathsf{L}_k$, then $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{C}_k]$ and hence $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D}, \mathcal{C}]$, so $\tau \in \mathsf{Acc}(\mathcal{D}[k])$. Thus $\mathsf{L}_k \subseteq \mathsf{Acc}(\mathcal{D}[k])$. Conversely, if $\mathsf{L}_k \subseteq \mathsf{Acc}(\mathcal{D}[k])$ and $G \in \mathcal{P}[\mathcal{C}_k]$, choose $\tau \in \mathsf{L}_k$ with $\mathcal{G}(\tau) \simeq G$. Then $\tau \in \mathsf{Acc}(\mathcal{D}[k])$ and therefore $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D}, \mathcal{C}]$. Since $\mathcal{P}[\mathcal{D}, \mathcal{C}]$ is a graph property (closed under isomorphism), it follows that $G \in \mathcal{P}[\mathcal{D}, \mathcal{C}]$. □

Remark 6. The coherence assumption is essential: without it, acceptance could depend on properties of the processed decompositions, and inclusion of graph classes would not reduce to language inclusion.

5.2 Dynamic Programming Refutation

As established in the previous section, verifying the inclusion $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$ reduces to checking whether every decomposition term in L_k is accepted by the DP-core $\mathcal{D}[k]$, provided that $\mathcal{D}$ is $\mathcal{C}$-coherent. If this inclusion fails, we certify the failure via a *dynamic programming refutation* (*DP-refutation*): an explicit sequence of partial evaluations tracking automaton states and witness sets until a contradiction is reached—a final state paired with a witness set lacking any final witness.

DP-refutations serve as combinatorial certificates witnessing non-inclusion, analogous to counter-models in classical logic. We define them formally in terms of inconsistent state-witness pairs, then prove that their existence is both necessary and sufficient for inclusion failure.

Before introducing the notion of a refutation, we formalize the key concept of inconsistency between the final states of the automaton and the acceptance condition defined by the DP-core.

Throughout this section we assume that the chosen realization $\mathcal{A} = \{\mathcal{A}_k\}_{k \in \mathbb{N}}$ is *bag-aware*: for every $k \in \mathbb{N}$, every state q of $\mathcal{A}_k$ carries a bag $\mathrm{bag}_k(q) \subseteq [k+1]$ such that for every term τ over Σ_k, every run ρ of $\mathcal{A}_k$ on τ, and every subterm occurrence σ of τ, we have

$$\mathrm{bag}_k(\rho(\sigma)) = B(\sigma).$$

In particular, this applies to accepting runs on terms in L_k. Equivalently, every transition of $\mathcal{A}_k$ preserves the intended bag carried by the corresponding decomposition constructor. For the instructive-tree-decomposition realization this is immediate, since the automaton

state is precisely the current bag.

Definition 36 (Inconsistent Pair)**.** *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class, and let $\mathcal{A} = \{\mathcal{A}_k\}_{k\in\mathbb{N}}$ be an automaton realizing $\mathcal{C}$. Let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core, and let $k \in \mathbb{N}$.*

A pair (q, W), where q is a state of $\mathcal{A}_k$ and $W \subseteq \mathcal{D}[k].\mathcal{W}$, is called $(\mathcal{A}, \mathcal{D}, k)$-inconsistent if:

- *q is a final state of $\mathcal{A}_k$, and*
- *the set W contains no final local witness for $\mathcal{D}[k]$ at the bag carried by q, that is,*

$$\forall w \in W : \ \mathcal{D}[k].\texttt{Final}(\mathrm{bag}_k(q), w) = 0.$$

With this definition in place, we can now define what it means to refute the inclusion $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$ using a derivation that ultimately yields an inconsistency.

Definition 37 (DP-Refutation)**.** *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class with a bag-aware realization $\mathcal{A} = \{\mathcal{A}_k\}_{k\in\mathbb{N}}$, and let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core. For convenience we write $\mathcal{A}_k = \mathcal{A}_k$. Fix $k \in \mathbb{N}$.*

An $(\mathcal{A}, \mathcal{D}, k)$-refutation is a finite sequence of pairs:

$$R \equiv (q_1, W_1), (q_2, W_2), \ldots, (q_m, W_m),$$

satisfying the following two conditions:

1. ***Terminal Inconsistency:*** *The final pair (q_m, W_m) is $(\mathcal{A}, \mathcal{D}, k)$-inconsistent, as defined in Definition 36.*
2. ***Constructibility:*** *For each $i \in [m]$, the pair (q_i, W_i) is either:*
   (a) ***Leaf case:*** *There exists a nullary symbol $a \in \Sigma_k$ such that $a \to q_i$ is a transition in $\mathcal{A}_k$ and:*
   $$W_i = \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q_i)).$$
   (b) ***Composition case:*** *There exists a symbol $a \in \Sigma_k$ of arity $r > 0$, and indices $j_1, \ldots, j_r < i$, such that:*
   $$a(q_{j_1}, \ldots, q_{j_r}) \to q_i$$
   is a transition in $\mathcal{A}_k$, and
   $$W_i = \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q_i), W_{j_1}, \ldots, W_{j_r}).$$

The indices $j_1, \ldots, j_r$ need not be distinct; this allows a constructor to use the same earlier pair in several child positions.

Having formalized the notion of $(\mathcal{A}, \mathcal{D}, k)$-refutations, we now show that such refutations completely characterize failures of inclusion between decomposition-based and DP-core-based graph properties. In particular, whenever the inclusion

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$$

does not hold, there must exist a finite refutation sequence witnessing this fact. This insight is fundamental to width-based automated theorem proving: it reduces the verification of universal statements over bounded-width graphs to the constructive search for small counterexamples.

We formalize this reduction in the following theorem.

Theorem 6 (Refutation Characterization)**.** *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class with bag-aware realization $\mathcal{A} = \{\mathcal{A}_k\}_{k\in\mathbb{N}}$, and let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core. Then, for every $k \in \mathbb{N}$, we have:*

$$\mathcal{P}[\mathcal{C}_k] \not\subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}] \quad \textit{if and only if} \quad \textit{there exists a } (\mathcal{A}, \mathcal{D}, k)\textit{-refutation.}$$

Proof. We divide the proof into two directions.

($\Rightarrow$) Refutation completeness. Suppose $\mathcal{P}[\mathcal{C}_k] \not\subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$. Then, there exists a graph G such that:

$$G \in \mathcal{P}[\mathcal{C}_k] \quad \text{but} \quad G \notin \mathcal{P}[\mathcal{D}, \mathcal{C}].$$

By the definition of $\mathcal{P}[\mathcal{C}_k]$, there exists a term $\tau \in \mathsf{L}_k$ such that $\mathcal{G}(\tau) \simeq G$. Because $G \notin \mathcal{P}[\mathcal{D}, \mathcal{C}]$ and $\mathcal{D}$ is coherent, it must be the case that:

$$\tau \notin \mathsf{Acc}(\mathcal{D}[k]).$$

Since $\tau \in \mathsf{L}_k = L(\mathcal{A}_k)$, there exists an accepting run of $\mathcal{A}_k$ on τ. Fix such a run and denote it by ρ. For every subterm occurrence σ of τ, let $\rho(\sigma)$ denote the state assigned by ρ to the root of that occurrence.

Let $\mathrm{Sub}(\tau)$ denote the set of all subterm occurrences of τ. Since τ is a term over Σ_k, its subterms admit a topological ordering that respects the syntax tree: choose $\sigma_1, \ldots, \sigma_m$ so that $\sigma_m = \tau$ and whenever σ_i is a proper subterm of σ_j, we have $i < j$.

We now define a sequence of pairs (q_i, W_i) for each $i \in [m]$:

- If $\sigma_i = a$ is a nullary symbol, set $q_i := \rho(\sigma_i)$. Since ρ is a run, we have a transition $a \to q_i$ in $\mathcal{A}_k$. Set:
$$W_i = \mathcal{D}[k].\hat{a}(B(\sigma_i)).$$
- If $\sigma_i = a(\sigma_{j_1}, \ldots, \sigma_{j_r})$ for some $a \in \Sigma_k$ of arity $r > 0$, then by the inductive hypothesis we already have $(q_{j_1}, W_{j_1}), \ldots, (q_{j_r}, W_{j_r})$. Set $q_i := \rho(\sigma_i)$. Since ρ is a run, we have a transition
$$a(q_{j_1}, \ldots, q_{j_r}) \to q_i$$
in $\mathcal{A}_k$. Set:
$$W_i = \mathcal{D}[k].\hat{a}(B(\sigma_i), W_{j_1}, \ldots, W_{j_r}).$$

For every $i \in [m]$, bag-awareness gives

$$\mathrm{bag}_k(q_i) = B(\sigma_i).$$

By construction, this sequence therefore satisfies the constructibility condition of Definition 37. Now consider the final pair (q_m, W_m). Since ρ is accepting, $q_m = \rho(\tau)$ is a final state of $\mathcal{A}_k$. Moreover, bag-awareness gives

$$\mathrm{bag}_k(q_m) = B(\tau).$$

Also, $\tau \notin \mathsf{Acc}(\mathcal{D}[k])$, so $W_m = \Gamma[\mathcal{D}, k](\tau)$ contains no final local witness for the bag $B(\tau)$. Hence, this pair is $(\mathcal{A}, \mathcal{D}, k)$-inconsistent.

Therefore, the sequence:

$$(q_1, W_1), \ldots, (q_m, W_m)$$

is a valid $(\mathcal{A}, \mathcal{D}, k)$-refutation.

($\Leftarrow$) Refutation soundness. Suppose a $(\mathcal{A}, \mathcal{D}, k)$-refutation:

$$R \equiv (q_1, W_1), \ldots, (q_m, W_m)$$

exists. We will construct terms $\sigma_1, \ldots, \sigma_m$ such that for each $i \in [m]$,

$$\begin{aligned} W_i &= \Gamma[\mathcal{D}, k](\sigma_i), \\ q_i &\text{ is the root state of some run of } \mathcal{A}_k \text{ on } \sigma_i, \\ \mathrm{bag}_k(q_i) &= B(\sigma_i). \end{aligned}$$

The construction proceeds inductively, based on the structure of each pair:

- If (q_i, W_i) is introduced via a nullary symbol a (i.e., a leaf case), we define:

  $$\sigma_i = a.$$

  Then the displayed transition $a \to q_i$ is a run of $\mathcal{A}_k$ on σ_i, so bag-awareness gives $\mathrm{bag}_k(q_i) = B(\sigma_i)$. By the leaf clause of Definition 37,

  $$W_i = \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q_i)) = \mathcal{D}[k].\hat{a}(B(\sigma_i)) = \Gamma[\mathcal{D}, k](\sigma_i).$$

- If (q_i, W_i) arises from a transition of the form:

  $$a(q_{j_1}, \ldots, q_{j_r}) \to q_i \quad \text{and} \quad W_i = \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q_i), W_{j_1}, \ldots, W_{j_r}),$$

  then we let:

  $$\sigma_i = a(\sigma_{j_1}, \ldots, \sigma_{j_r}).$$

  By induction, each σ_{j_t} admits a run whose root state is q_{j_t}, so the displayed transition yields a run of $\mathcal{A}_k$ on σ_i with root state q_i. Again bag-awareness gives $\mathrm{bag}_k(q_i) = B(\sigma_i)$, and therefore

  $$W_i = \mathcal{D}[k].\hat{a}(B(\sigma_i), \Gamma[\mathcal{D}, k](\sigma_{j_1}), \ldots, \Gamma[\mathcal{D}, k](\sigma_{j_r})) = \Gamma[\mathcal{D}, k](\sigma_i).$$

This inductive construction guarantees that:

$$\begin{aligned} W_i &= \Gamma[\mathcal{D}, k](\sigma_i), \\ q_i &\text{ is the root state of some run of } \mathcal{A}_k \text{ on } \sigma_i, \qquad \text{for all } i \in [m]. \\ \mathrm{bag}_k(q_i) &= B(\sigma_i) \end{aligned}$$

In particular, the final pair (q_m, W_m) is inconsistent, so q_m is a final state. Therefore the run witnessing reachability of q_m at the root of σ_m is accepting, i.e. $\sigma_m \in \mathsf{L}_k$. On the other hand, inconsistency says precisely that W_m has no final local witness for the bag $\mathrm{bag}_k(q_m) = B(\sigma_m)$, hence $\sigma_m \notin \mathsf{Acc}(\mathcal{D}[k])$. Since $\sigma_m \in \mathsf{L}_k$, Proposition 4 gives $\mathcal{G}(\sigma_m) \notin \mathcal{P}[\mathcal{D}, \mathcal{C}]$. At the same time, $\sigma_m \in \mathsf{L}_k$ implies $\mathcal{G}(\sigma_m) \in \mathcal{P}[\mathcal{C}_k]$ by definition. Therefore

$$\mathcal{G}(\sigma_m) \in \mathcal{P}[\mathcal{C}_k] \setminus \mathcal{P}[\mathcal{D}, \mathcal{C}].$$

Therefore, the inclusion $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$ fails. □

Example 1 (Non-inclusion at width 2: bipartiteness)**.** *Let* $\mathcal{C} = \mathsf{ITD}$ *and* $\mathcal{D} = \mathcal{C}_2^X$*. At*

width $k = 2$, the triangle K_3 has treewidth 2 and is represented by the root-open instructive term

$$\tau_{\triangle} = \texttt{intro\_1\_3}\Big(\texttt{intro\_2\_3}\big(\texttt{intro\_1\_2}\big(\texttt{intro\_3}\big(\texttt{intro\_2}\big(\texttt{intro\_1}(\texttt{leaf})\big)\big)\big)\big)\Big).$$

Evaluating $\mathcal{C}_2^{\chi}[2]$ bottom-up yields witness sets of partial proper 2-colorings over the current bag. After all three edge-introduction steps, there is no proper 2-coloring of $\{1,2,3\}$, so the root witness set is empty. In the root-open convention fixed in the paragraph "Root bags and closure" above, every bag state of the ITD recognizer is final, hence the product state $(B = \{1,2,3\}, W = \emptyset)$ is inconsistent. Therefore $\mathsf{L}_2 \nsubseteq \mathsf{Acc}(\mathcal{C}_2^{\chi}[2])$, and $\mathcal{G}_2(\tau_{\triangle}) \simeq K_3$ is a concrete counterexample in $\mathcal{P}[\mathcal{C}_2] \setminus \mathcal{P}[\mathcal{C}_2^{\chi}, \mathcal{C}]$.

If a root-closed presentation is preferred, append `forget_1`*,* `forget_2`*, and* `forget_3` *at the root. This does not change the realized graph, and by coherence it does not affect whether the decomposition is accepted by the core.*

Theorem 6 established that the failure of an inclusion

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$$

is witnessed by the existence of a finite $(\mathcal{A}, \mathcal{D}, k)$-refutation. This result yields a forward-chaining algorithmic strategy: by systematically constructing all reachable $(\mathcal{A}, \mathcal{D}, k)$-pairs and checking for inconsistency, we can determine whether the inclusion holds. The next theorem states the bound in our notation and explains the combinatorial mechanism behind it.

Theorem 7 (Bounded Refutation Length)**.** *Let $\mathcal{C}$ be a treelike decomposition class with bag-aware realization $\mathcal{A} = \{\mathcal{A}_k\}_{k\in\mathbb{N}}$, and let $\mathcal{D}$ be a finite, internally polynomial, $\mathcal{C}$-coherent treelike DP-core. For each $k \in \mathbb{N}$, let s_k be the number of states of $\mathcal{A}_k$, and write $\beta_k = \beta_{\mathcal{D}}(k)$ and $\mu_k = \mu_{\mathcal{D}}(k)$. If a $(\mathcal{A}, \mathcal{D}, k)$-refutation exists, then there exists one of length at most*

$$H(k) = s_k \cdot \left(2^{\beta_k}\right)^{O(\mu_k)}.$$

Proof. There are only s_k possible automaton states. Moreover, the soundness construction in the proof of Theorem 6 applies to every pair in any refutation: for each position i, there is a term σ_i whose automaton run reaches q_i and whose root DP table is exactly W_i. Thus every witness set appearing in a refutation is useful. Because $\mathcal{D}$ is finite, every witness in such a set can be encoded using at most β_k bits, and each such witness set contains at most μ_k witnesses. Hence the number of distinct witness sets that can appear is bounded by $\left(2^{\beta_k}\right)^{O(\mu_k)}$. The product $H(k)$ therefore bounds the total number of distinct combinations of an automaton state and a witness set.

Suppose a $(\mathcal{A}, \mathcal{D}, k)$-refutation exists, and choose one of minimal length. If its length exceeded $H(k)$, two different positions $i < j$ would contain the same pair $(q_i, W_i) = (q_j, W_j)$. If $j < m$, we form a shorter sequence by deleting the entry at j, renumbering every later position $p > j$ to $p-1$, and redirecting all subsequent dependencies: whenever a constructibility justification for a later position cites j, we replace that citation with i. Because $(q_i, W_i) = (q_j, W_j)$, the transition and witness computation at the affected position remain valid. Moreover, predecessor order is preserved: if $p > j$ and j was cited at position p, then after renumbering the new position is $p-1$, and still $i < p-1$ because $i < j < p$. All other predecessor citations remain earlier than the renumbered position as well. This yields a valid refutation of strictly shorter length, contradicting minimality. If instead $j = m$, then the pair at position i is already inconsistent, because it is equal to the final inconsistent pair. Truncating the refutation after position i therefore yields a shorter valid refutation, again contradicting minimality. Therefore every minimal refutation has length at most $H(k)$, and in particular some refutation of length at most $H(k)$ exists whenever a refutation exists. □

When the DP-core is both finite and internally polynomial, and the decomposition class is defined by an automaton, the corresponding search space is effectively bounded. The following theorem formalizes this insight by providing an explicit upper bound on the time required to decide inclusion.

Theorem 8 (Inclusion Test)**.** *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class with bag-aware automaton $\mathcal{A}$, and let $\mathcal{D}$ be a finite, internally polynomial $\mathcal{C}$-coherent treelike DP-core. Let r be the arity of $\mathcal{C}$ and $f(k)$ be the complexity of $\mathcal{A}$. Assume moreover that, once $\mathcal{A}_k$ is constructed, its transitions of arity at most r can be enumerated and queried within the same polynomial-over-$f(k)$ bounds. Then, one can determine whether $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$ in time*

$$k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{O(r\cdot\beta_{\mathcal{D}}(k)\cdot\mu_{\mathcal{D}}(k))} \leq k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{r\cdot 2^{O(\beta_{\mathcal{D}}(k))}}.$$

Proof. Using a worklist enumeration, we successively enumerate the $(\mathcal{A}, \mathcal{D}, k)$-pairs that can be derived using the rule 2b of Definition 37. We may assume that the first traversed pairs are those in the set

$$\{(q, \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q))) \mid \mathsf{r}(a) = 0,\ a \to q \text{ is a transition of } \mathcal{A}_k\},$$

which correspond to symbols of arity 0. We run this search until either an inconsistent $(\mathcal{A}, \mathcal{D}, k)$-pair has been reached or there is no $(\mathcal{A}, \mathcal{D}, k)$-pair left to be enumerated. In

the first case, the obtained list of pairs

$$R = (q_1, W_1)(q_2, W_2) \dots (q_m, W_m) \tag{5.1}$$

is, by construction, an $(\mathcal{A}, \mathcal{D}, k)$-refutation since (q_m, W_m) is inconsistent, the initial worklist entries satisfy the nullary clause 2a of Definition 37, and every later pair added during the loop is obtained by applying the rule 2b. In the second case, no such refutation exists. More specifically, at the beginning of the process, R is the empty list, and the pairs $\{(q, \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q))) : \ \mathsf{r}(a) = 0,\ a \to q \text{ is a transition of } \mathcal{A}_k\}$ are added to a worklist Y. While Y is non-empty, we delete an arbitrary pair (q, W) from Y and append this pair to R. If the pair is inconsistent, we have constructed a refutation R, and therefore, we return R. Otherwise, for each (q', W') that can be obtained from (q, W) using the rule 2b of Definition 37 (together with the required predecessor pairs from R), we insert (q', W') into the worklist Y provided this pair has not already been discovered, i.e., it is not in $\hat{R}$ (with Y represented as a set). We repeat this process until either R has been returned or Y is empty. In the latter case, we conclude that $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$, and return *Inclusion Holds*. This construction is detailed in Algorithm 1. The key invariant is that every pair in $Y \cup \hat{R}$ is derivable from the nullary seeds by the constructibility clauses of Definition 37, and conversely every derivable pair eventually enters $Y \cup \hat{R}$. The forward direction is immediate from the initialization and the update rule. For the converse, we argue by induction on the length of a derivation of a pair (q, W): nullary derivations are placed into Y initially, and if (q, W) is obtained compositionally from predecessor pairs, then by the induction hypothesis all predecessors eventually belong to $\hat{R}$; when the last missing predecessor is processed, the algorithm enumerates the required constructor and transition and inserts (q, W) into Y unless it has already been discovered. Hence, when the worklist empties, $\hat{R}$ is exactly the least constructibly closed set of derivable pairs.

We next record the realization claim needed both for correctness and for the state bound: every generated pair $(q, W) \in Y \cup \hat{R}$ is realized by some decomposition term τ such that a run of $\mathcal{A}_k$ reaches q at the root of τ,

$$\mathrm{bag}_k(q) = B(\tau),$$

and

$$\Gamma[\mathcal{D}, k](\tau) = W.$$

This is proved by induction on the discovery order. For an initial seed, choose the corresponding nullary term a; then $a \to q$ is a run transition of $\mathcal{A}_k$, and bag-awareness gives $\mathrm{bag}_k(q) = B(a)$. Moreover,

$$W = \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q)) = \Gamma[\mathcal{D}, k](a).$$

For a newly inserted pair obtained from already realized predecessor pairs $(q_1, W_1), \ldots, (q_t, W_t)$ by a constructor a, let $\tau_1, \ldots, \tau_t$ be realizing terms for those predecessors and set $\tau := a(\tau_1, \ldots, \tau_t)$. The corresponding automaton transition $a(q_1, \ldots, q_t) \to q$ yields a run with root state q. Bag-awareness gives $\mathrm{bag}_k(q) = B(\tau)$, and the definition of dynamization gives $\Gamma[\mathcal{D}, k](\tau) = W$. Consequently, if the algorithm encounters an inconsistent generated pair (q, W), then the realizing term τ is accepted by the automaton, hence $\tau \in \mathsf{L}_k$, and so $\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{C}_k]$ by the definition of $\mathcal{P}[\mathcal{C}_k]$. Moreover, inconsistency gives $\tau \notin \mathsf{Acc}(\mathcal{D}[k])$. By Proposition 4, $\mathcal{G}(\tau) \notin \mathcal{P}[\mathcal{D}, \mathcal{C}]$. Conversely, by Theorem 6, any failure of the inclusion gives a refutation and hence an inconsistent derivable pair. Therefore an inconsistent pair exists if and only if the algorithm encounters one.

We now establish an upper bound on the running time of the algorithm. By the realization claim, every generated witness set is useful. Since $\mathcal{A}$ has complexity $f(k)$, the worklist enumeration constructs at most

$$f(k) \cdot \delta_{\mathcal{D}}(k)$$

pairs of the form (q, W), where q is a state of $\mathcal{A}_k$ and $W \subseteq \mathcal{D}[k].\mathcal{W}$. Furthermore, every constructor of $\mathcal{C}$ has arity at most r, so the creation of a new pair may require the analysis of at most

$$f(k)^r \cdot \delta_{\mathcal{D}}(k)^r$$

tuples of previously created pairs. For each such tuple $((q_1, W_1), \ldots, (q_t, W_t))$ with $t \leq r$, the computation of successor states from $(q_1, \ldots, q_t)$ involves iterating over the transitions of $\mathcal{A}_k$, taking time $f(k)^{O(1)}$. Moreover, as argued in the proof of Theorem 4, the computation of the transition value

$$W' = \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q'), W_1, \ldots, W_t)$$

takes time

$$k^{O(1)} \cdot r^{O(1)} \cdot \beta_{\mathcal{D}}(k)^{O(1)} \cdot \mu_{\mathcal{D}}(k)^{O(r)}.$$

Therefore, the whole process takes time at most

$$k^{O(1)} \cdot r^{O(1)} \cdot \beta_{\mathcal{D}}(k)^{O(1)} \cdot \mu_{\mathcal{D}}(k)^{O(r)} \cdot f(k)^{r+O(1)} \cdot \delta_{\mathcal{D}}(k)^{r+O(1)}. \tag{5.2}$$

Additionally, under the fixed binary witness encoding, the number of possible useful witnesses of bitlength at most $\beta_{\mathcal{D}}(k)$ is $2^{O(\beta_{\mathcal{D}}(k))}$. Hence $\mu_{\mathcal{D}}(k) \leq 2^{O(\beta_{\mathcal{D}}(k))}$ and $\delta_{\mathcal{D}}(k) \leq$

$2^{O(\beta_{\mathcal{D}}(k)\cdot\mu_{\mathcal{D}}(k))}$. Therefore, Expression 5.2 can be simplified to

$$k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{O(r\cdot\beta_{\mathcal{D}}(k)\cdot\mu_{\mathcal{D}}(k))} \leq k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{r\cdot 2^{O(\beta_{\mathcal{D}}(k))}}.$$

□

Input : A bag-aware automaton $\mathcal{A}$ for $\mathcal{C}$, a finite, $\mathcal{C}$-coherent treelike DP-core $\mathcal{D}$, and an integer $k \in \mathbb{N}$.
Output: An $(\mathcal{A}, \mathcal{D}, k)$-refutation R if $\mathcal{P}[\mathcal{C}_k] \not\subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$, and "Inclusion Holds", otherwise.

$R \leftarrow [\,]$; `/* [ ] is the empty list. */`
$\hat{R} \leftarrow \{\,\}$; `/* { } is the empty set. */`
$Y \leftarrow \{(q, \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q))) : \mathsf{r}(a) = 0,\ a \to q \text{ is a transition of } \mathcal{A}_k\}$;
while $Y \neq \emptyset$ **do**
  Remove some pair (q, W) from Y, append it to R, and insert it in $\hat{R}$;
  if (q, W) *is an inconsistent* $(\mathcal{A}, \mathcal{D}, k)$*-pair,* **then**
    return R ; `/* In this case, R is a (A,D,k)-refutation. */`
  else
    foreach $a \in \Sigma_k$ **do**
      foreach *sequence* $(q_1, W_1) \ldots (q_{\mathsf{r}(a)}, W_{\mathsf{r}(a)})$ *of pairs from* $\hat{R}^{\mathsf{r}(a)}$ *having at least one occurrence of* (q, W) **do**
        foreach *transition* $a(q_1, \ldots, q_{\mathsf{r}(a)}) \to q'$ *in* $\mathcal{A}_k$ **do**
          $W' \leftarrow \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q'), W_1, \ldots, W_{\mathsf{r}(a)})$;
          if (q', W') *is not in* $\hat{R}$ **then** $Y \leftarrow Y \cup \{(q', W')\}$;
        end
      end
    end
  end
end
return "Inclusion Holds."

Algorithm 1: Forward-Chaining Inclusion Test

Conceptually, Algorithm 1 computes the least closed set of reachable product states (q, W) in the transition system $\mathcal{A}_k \times \mathcal{D}[k]$. In implementations, storing one predecessor tuple (plus constructor label) for each newly discovered state is enough to reconstruct an explicit refutation and a counterexample term once inconsistency is reached. Algorithm 1 is presented as a *completeness reference*: it is the simplest forward-chaining procedure whose correctness follows directly from the refutation characterization. Its worst-case behavior is governed by the number of reachable product states (automaton configuration + witness-set state), which is finite under the hypotheses of Theorem 7.

In practice, however, naive exploration can still be infeasible even at small widths because of symmetries and dead-ends encountered early. For this reason, Chapters 6 and 7 introduce two completeness-preserving reductions: (i) symmetry reduction by canonizing

intermediate DP states modulo bag relabelings, and (ii) premise-driven pruning for implication checking under monotonicity assumptions. The conceptual role of the present chapter is to establish the finite search space and the evidence objects; the role of the next two chapters is to reduce the size of the explored subset of that space without losing refutations or proofs.

5.3 Counterexample Size Bounds

When the inclusion $\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$ fails, Theorem 6 guarantees a counterexample term $\tau \in \mathsf{L}_k$. We now bound the height and size of such counterexamples in terms of refutation length.

Corollary 3. *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class of arity r with bag-aware automaton $\mathcal{A}$, let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core, and let*

$$R = (q_1, W_1), \ldots, (q_m, W_m)$$

be a $(\mathcal{A}, \mathcal{D}, k)$-refutation. Then there exists a decomposition term $\tau \in \mathsf{L}_k$ such that:

1. *The term τ has height at most $m - 1$.*
2. *The graph $\mathcal{G}_k(\tau)$ belongs to $\mathcal{P}[\mathcal{C}_k] \setminus \mathcal{P}[\mathcal{D}, \mathcal{C}]$.*
3. *The size of τ is at most m if $r = 1$, and at most $\frac{r^m-1}{r-1}$ if $r > 1$.*

Proof. In the second half of the proof of Theorem 6, the refutation sequence is converted into a topologically ordered family of terms $(\tau_i)_{i=1}^m$ whose final element is the desired counterexample term. We claim by induction on i that

$$\textsc{Height}(\tau_i) \leq i - 1.$$

For $i \geq 1$, if the pair (q_i, W_i) arises from a nullary constructor, then τ_i is a leaf and $\textsc{Height}(\tau_i) = 0 \leq i - 1$. Otherwise, τ_i is obtained by applying one constructor to predecessor terms τ_{j_t} with $j_t < i$. By the induction hypothesis,

$$\textsc{Height}(\tau_i) \leq 1 + \max_t \textsc{Height}(\tau_{j_t}) \leq 1 + \max_t (j_t - 1) \leq i - 1.$$

Applying this to $i = m$ gives item 1.

For the size bound, a unary term of height at most $m-1$ has at most m nodes. If $r > 1$, a rooted tree of height at most $m-1$ and branching factor at most r has at most

$$1 + r + \cdots + r^{m-1} = \frac{r^m - 1}{r - 1}$$

nodes. Since τ has arity at most r, this yields item 3. Item 2 is exactly the graph-theoretic conclusion established in Theorem 6. □

Remark 7 (Binary instructive case). For instructive decompositions, the arity is $r = 2$. Hence a refutation of length m yields a counterexample term of height at most $m-1$ and size at most $2^m - 1$. In particular, any fixed-width bound on refutation height immediately gives a singly exponential bound on the size of minimal counterexample terms.

We now relate the size of counterexamples to the size of the search space used in the inclusion testing algorithm from Theorem 8. That algorithm performs a forward-chaining enumeration of all $(\mathcal{A}, \mathcal{D}, k)$-pairs. Since the number of such pairs is finite and computable when $\mathcal{D}$ is finite, the length of the shortest possible refutation is also bounded.

Corollary 4. *Let $\mathcal{C} = \{(\Sigma_k, \mathsf{L}_k, \mathcal{G}_k)\}_{k\in\mathbb{N}}$ be a treelike decomposition class of arity r with a bag-aware automaton $\mathcal{A}$ of complexity $f(k)$, and let $\mathcal{D}$ be a finite, $\mathcal{C}$-coherent treelike DP-core. If the inclusion*

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$$

does not hold, then there exists a decomposition term $\tau \in \mathsf{L}_k$ such that:

1. *$\mathcal{G}_k(\tau) \in \mathcal{P}[\mathcal{C}_k] \setminus \mathcal{P}[\mathcal{D}, \mathcal{C}]$.*
2. *The height of τ is at most $f(k) \cdot \delta_{\mathcal{D}}(k) - 1$.*
3. *The total number of nodes in τ is at most:*
   - *$f(k) \cdot \delta_{\mathcal{D}}(k)$ if $r = 1$.*
   - *$r^{f(k)\cdot\delta_{\mathcal{D}}(k)}$ if $r > 1$.*

Proof. Use the bag-aware automaton $\mathcal{A}$ witnessing the complexity $f(k)$ of $\mathcal{C}$. If the inclusion fails, Theorem 6 yields a finite $(\mathcal{A}, \mathcal{D}, k)$-refutation. Choose one of minimum length m.

For every position $i \in [m]$, the soundness direction in the proof of Theorem 6 constructs a term σ_i such that

$$W_i = \Gamma[\mathcal{D}, k](\sigma_i).$$

Hence each W_i is a width-k useful witness set in the sense of Definition 27. There are at most $f(k)$ automaton states and at most $\delta_{\mathcal{D}}(k)$ useful witness sets at width k. Therefore the refutation can contain at most

$$f(k) \cdot \delta_{\mathcal{D}}(k)$$

distinct pairs (q_i, W_i).

If m were larger than this bound, then two positions $i < j$ in the minimal refutation would contain the same pair. Applying the shortening argument from the proof of Theorem 7, we obtain a shorter valid refutation: either $j < m$ and the later occurrence can be deleted, later positions renumbered, and every citation of j redirected to i; because $i < j < p$ for every affected later position p, the predecessor-order condition remains valid after renumbering. Or $j = m$, in which case the refutation can be truncated at position i, which is already inconsistent. This contradicts minimality. Therefore there exists a refutation of length at most $f(k) \cdot \delta_{\mathcal{D}}(k)$.

Applying Corollary 3 with this value of m yields the stated height bound. For $r = 1$ the size bound is m, and for $r > 1$ we simplify $\frac{r^m-1}{r-1}$ to the coarser upper bound r^m. □

5.4 Bounded-Size Inclusion Test

Theorem 8 depends on the DP-core $\mathcal{D}$ being finite. Many useful DP-cores (e.g., for optimization problems) are *non-finite* (Definition 32): the witness bitlength $\beta_{\mathcal{D}}(k, n)$ grows with n. Non-finiteness is benign for fixed-instance evaluation (Section 4.7) but problematic for universality checking, where the algorithm ranges over *all* width-k terms. Nevertheless, inclusion testing remains feasible when restricted to decompositions of bounded size n.

Corollary 5 (Bounded-Size Inclusion Test)**.** *Let $\mathcal{C}$ be a treelike decomposition class of arity r and complexity $f(k)$ with a bag-aware automaton $\mathcal{A}$, and let $\mathcal{D}$ be an internally polynomial, $\mathcal{C}$-coherent treelike DP-core (not necessarily finite). Then, for any width parameter $k \in \mathbb{N}$ and size bound $n \in \mathbb{N}$, one can determine in time*

$$k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{O(r \cdot \beta_{\mathcal{D}}(k,n) \cdot \mu_{\mathcal{D}}(k,n))} \cdot n^{O(r)}$$

whether every graph corresponding to a $\mathcal{C}$-decomposition of width at most k and size at most n belongs to $\mathcal{P}[\mathcal{D}, \mathcal{C}]$.

Proof. The argument is similar to that of Theorem 8, but the search space is limited to

decomposition terms of size at most n. We perform a worklist enumeration over extended triples (q, W, s), where q is a state of $\mathcal{A}_k$, $W \subseteq \mathcal{D}[k].\mathcal{W}$, and $s \in \mathbb{N}$ tracks the size. For leaves $a \to q$, we initialise with $(q, \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q)), 1)$. For a transition $a(q_1, \ldots, q_r) \to q'$ and predecessors (q_i, W_i, s_i), we generate $(q', \mathcal{D}[k].\hat{a}(\mathrm{bag}_k(q'), W_1, \ldots, W_r), 1 + \sum s_i)$ provided $1 + \sum s_i \leq n$. By induction on the derivation length, a triple (q, W, s) is generated if and only if there exists a term τ of size s such that some run of $\mathcal{A}_k$ reaches q at the root of τ and $\Gamma[\mathcal{D}, k](\tau) = W$. The base case is exactly the leaf initialization, and the inductive step matches the term constructor $a(\tau_1, \ldots, \tau_r)$ with the update rule $1 + \sum s_i$. Therefore the algorithm encounters a triple (q, W, s) with q final and W non-final if and only if there exists a width-k decomposition term τ of size at most n with $\tau \in \mathsf{L}_k \setminus \mathsf{Acc}(\mathcal{D}[k])$. For such a term, Proposition 4 yields

$$\mathcal{G}(\tau) \in \mathcal{P}[\mathcal{C}_k] \setminus \mathcal{P}[\mathcal{D}, \mathcal{C}].$$

Conversely, if a graph in that set difference is represented by some width-k decomposition term of size at most n, then the same proposition shows that the representing term lies in $\mathsf{L}_k \setminus \mathsf{Acc}(\mathcal{D}[k])$, so the corresponding triple is eventually generated. Write

$$\beta := \beta_{\mathcal{D}}(k, n) \qquad \text{and} \qquad \mu := \mu_{\mathcal{D}}(k, n).$$

There are at most $f(k) \cdot \delta_{\mathcal{D}}(k, n) \cdot n$ distinct triples (q, W, s), and under the fixed binary witness encoding $\delta_{\mathcal{D}}(k, n) \leq 2^{O(\beta\mu)}$. Successor generation uses the same local DP-core computations as in the proof of Theorem 8, with the additional enumeration of up to r size coordinates and the arithmetic check $1+\sum s_i \leq n$, so the size component contributes an $n^{O(r)}$ factor. Therefore the total running time is

$$k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{O(r\beta\mu)} \cdot n^{O(r)},$$

as claimed. □

The main insight is that when $\beta_{\mathcal{D}}(k, n) = O(\log n)$ and $\mu_{\mathcal{D}}(k, n) = h(k)$ for some function h, the runtime becomes polynomial in n, with the degree depending only on k. In such cases, the bounded-size inclusion test provides a highly efficient tool for experimental verification of conjectures and small counterexample search.

Examples. The vertex-counting core has $\beta(k, n) = O(\log n)$ and $\mu(k, n) = 1$ (Theorem 44), and the maximum-independent-set core has $\beta(k, n) = O(k + \log n)$ and $\mu(k, n) \leq 2^{k+1}$ (Theorem 42). Both are therefore non-finite (Remark 4) but fall into the bounded-size regime above.

5.5 Combinators and Combinations

Many natural graph properties can be defined by combining simpler properties or by relating associated numerical invariants. For instance, classes of graphs whose chromatic number equals their clique number or whose independence number is greater than their domination number involve logical and arithmetic relationships between multiple parameters. In the context of treelike DP-cores, such combined properties can be represented via a formal mechanism we call *combinators*.

5.5.1 Combinators: Syntax and Semantics

Let $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ be graph properties, and let $\perp$ be a distinguished symbol not belonging to $\{0,1\}^*$. We define a *combinator* to be a function that, given Boolean indicators of property membership together with invariant values (or $\perp$ when the corresponding invariant is unavailable), returns a Boolean outcome. To avoid confusion with the decomposition-class symbol $\mathcal{C}$, we denote combinators by Φ.

Definition 38 (Combinator)**.** *An ℓ-combinator is a function*

$$\Phi : \{0,1\}^\ell \times (\{0,1\}^* \cup \{\perp\})^\ell \to \{0,1\}.$$

Definition 39 (Combined Graph Property)**.** *Given graph properties $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ and functions*

$$\mathcal{I}_1, \ldots, \mathcal{I}_\ell : \mathrm{GRAPHS} \to \{0,1\}^* \cup \{\perp\},$$

we define the combined graph property

$$\hat{\Phi}(\mathcal{P}_1, \ldots, \mathcal{P}_\ell, \mathcal{I}_1, \ldots, \mathcal{I}_\ell)$$

to consist of all graphs G such that

$$\Phi\left(\mathcal{P}_1(G), \ldots, \mathcal{P}_\ell(G), \mathcal{I}_1(G), \ldots, \mathcal{I}_\ell(G)\right) = 1.$$

Definition 40 (Totalized Invariant)**.** *Let $\mathcal{C}$ be a decomposition class, and let $\mathcal{D}$ be a $\mathcal{C}$-coherent treelike DP-core satisfying Condition 2b. We define the totalized invariant*

$$\mathrm{Inv}^{\perp}_{\mathcal{D},\mathcal{C}} : \mathrm{GRAPHS} \to \{0,1\}^* \cup \{\perp\}$$

by

$$\mathrm{Inv}^{\perp}_{\mathcal{D},\mathcal{C}}(G) = \begin{cases} \mathcal{I}[\mathcal{D},\mathcal{C}](G), & \text{if } G \in \mathcal{P}[\mathcal{D},\mathcal{C}], \\ \perp, & \text{otherwise.} \end{cases}$$

A combinator is said to be *polynomial* if it can be evaluated in time polynomial in the total input size (with the exponent independent of ℓ).

Lemma 8 (Computing lifted transitions from the DP-core interface)**.** *Let $\mathcal{D}$ be an internally polynomial treelike DP-core, let $k \in \mathbb{N}$, let $B \subseteq [k+1]$, and let $a \in \Sigma_k$ have arity r. If $r > 0$ and $W_1, \ldots, W_r \subseteq \mathcal{D}[k].\mathcal{W}$ are explicit finite witness sets, then the lifted transition*

$$\mathcal{D}[k].\hat{a}(B, W_1, \ldots, W_r)$$

can be computed in time polynomial in k and the total encoding length of $(B, W_1, \ldots, W_r)$, with polynomial degree allowed to depend on the fixed arity r; equivalently, under a standard product encoding the bound is

$$\big(k + |(B, W_1, \ldots, W_r)|\big)^{O(r)}.$$

If $r = 0$, then the lifted transition $\mathcal{D}[k].\hat{a}(B)$ can be computed in time polynomial in

$$k + \beta_{\mathcal{D}}(k,1) \cdot \mu_{\mathcal{D}}(k,1).$$

Proof. If $r = 0$, Definition 33(2b) constructs the raw nullary transition $\Delta_{k,a}(B)$ in time polynomial in

$$k + \beta_{\mathcal{D}}(k,1) \cdot \mu_{\mathcal{D}}(k,1).$$

The same condition is an explicit output-size assumption for this raw set, and Definition 33(2d) computes the cleaning map in time polynomial in the encoded size of that raw set. Thus the cleaned output $\mathcal{D}[k].\hat{a}(B)$ satisfies the claimed nullary bound.

Now assume $r > 0$. Enumerate all tuples $(w_1, \ldots, w_r) \in W_1 \times \cdots \times W_r$. The number of tuples is at most the rth power of the total input size. For each tuple, Definition 33(2c) computes the raw transition value $\Delta_{k,a}(B, w_1, \ldots, w_r)$ in time polynomial in k and the encoding length of $(B, w_1, \ldots, w_r)$. Taking the union of these outputs and removing duplicates requires only polynomial overhead in the total size of the enumerated family because the arity r is fixed by the constructor a. Applying the cleaning map to that explicit union is polynomial by Definition 33(2d). By definition of $\hat{a}$, the resulting cleaned union is exactly $\mathcal{D}[k].\hat{a}(B, W_1, \ldots, W_r)$. □

5.5.2 Constructing Combined DP-cores

The next theorem shows that if we have ℓ $\mathcal{C}$-coherent treelike DP-cores whose invariant components define graph-level invariants, then we can construct a single $\mathcal{C}$-coherent DP-core that realizes, on graphs represented by the decomposition class, the combined property defined by any polynomial combinator. When a constituent core is purely Boolean and has no meaningful graph-level invariant, we view it as carrying the dummy invariant that returns the empty string on accepted inputs. Its totalized invariant is then the empty string on accepted graphs and $\perp$ otherwise, and combinators that do not use invariant information simply ignore these coordinates.

Theorem 9 (Combinator Construction)**.** *Let Φ be a polynomial ℓ-combinator, let $\mathcal{C}$ be a treelike decomposition class, and let $\mathcal{D}_1, \ldots, \mathcal{D}_\ell$ be $\mathcal{C}$-coherent treelike DP-cores satisfying Condition 2b. Then there exists a $\mathcal{C}$-coherent treelike DP-core*

$$\mathcal{D} = \mathcal{D}(\Phi, \mathcal{D}_1, \ldots, \mathcal{D}_\ell)$$

such that:

1. *For every $k \in \mathbb{N}$ and every graph $G \in \mathcal{P}[\mathcal{C}_k]$,*

$$G \in \mathcal{P}[\mathcal{D}, \mathcal{C}] \quad \Longleftrightarrow \quad G \in \hat{\Phi}(\mathcal{P}[\mathcal{D}_1, \mathcal{C}], \ldots, \mathcal{P}[\mathcal{D}_\ell, \mathcal{C}], \mathrm{Inv}^{\perp}_{\mathcal{D}_1, \mathcal{C}}, \ldots, \mathrm{Inv}^{\perp}_{\mathcal{D}_\ell, \mathcal{C}}).$$

2. *Under a standard self-delimiting encoding of tuples of finite witness sets, the bitlength of $\mathcal{D}$ satisfies*

$$\beta_{\mathcal{D}}(k, n) = O\left(\sum_{i=1}^{\ell} \beta_{\mathcal{D}_i}(k, n) \cdot \mu_{\mathcal{D}_i}(k, n) \right).$$

3. *The multiplicity of $\mathcal{D}$ is 1: $\mu_{\mathcal{D}}(k, n) = 1$.*

4. *The deterministic state complexity of $\mathcal{D}$ satisfies*

$$\delta_{\mathcal{D}}(k, n) \leq \prod_{i=1}^{\ell} \delta_{\mathcal{D}_i}(k, n).$$

Proof. We construct $\mathcal{D}[k]$ pointwise from the components $\mathcal{D}_1[k], \ldots, \mathcal{D}_\ell[k]$. A witness of the combined core is a tuple of witness sets from the component cores:

$$\mathcal{D}[k].\mathcal{W} = \mathcal{P}_{\mathrm{fin}}(\mathcal{D}_1[k].\mathcal{W}) \times \cdots \times \mathcal{P}_{\mathrm{fin}}(\mathcal{D}_\ell[k].\mathcal{W}).$$

By coherence of each $\mathcal{D}_i$, we have $\mathcal{D}_i[k].\Sigma = \Sigma_k$, so we may set

$$\mathcal{D}[k].\Sigma := \Sigma_k.$$

We also set $\mathcal{D}[k].\texttt{Clean}$ to be the identity map on $\mathcal{P}_{\text{fin}}(\mathcal{D}[k].\mathcal{W})$.

For a constructor $a \in \Sigma_k$ of arity r, define the witness-level transition of the combined core by

$$\Delta_{k,a}\big(B, (W_1^1, \dots, W_1^\ell), \dots, (W_r^1, \dots, W_r^\ell)\big) = \Big\{(\mathcal{D}_1[k].\hat{a}(B, W_1^1, \dots, W_r^1), \dots, \mathcal{D}_\ell[k].\hat{a}(B, W_1^\ell, \dots, W_r^\ell))\Big\}.$$

Since $\mathcal{D}[k].\texttt{Clean}$ is the identity, the lifted transition $\mathcal{D}[k].\hat{a}$ is obtained by taking the union of these singleton outputs over all tuples of child witnesses. In particular, whenever each child input witness set is itself a singleton, the lifted transition also returns a singleton witness set. This is exactly the regime reached by dynamization, as shown in the induction below.

For a bag $B \subseteq [k+1]$ and a witness

$$X = (W_1, \dots, W_\ell) \in \mathcal{D}[k].\mathcal{W},$$

define

$$b_i(B, X) = 1 \quad \Longleftrightarrow \quad \exists w \in W_i \text{ with } \mathcal{D}_i[k].\texttt{Final}(B, w) = 1,$$

and

$$v_i(B, X) = \begin{cases} \mathcal{D}_i[k].\texttt{Inv}(W_i), & \text{if } b_i(B, X) = 1, \\ \bot, & \text{if } b_i(B, X) = 0. \end{cases}$$

We declare X to be final for $\mathcal{D}[k]$ at bag B exactly when

$$\Phi\big(b_1(B, X), \dots, b_\ell(B, X), v_1(B, X), \dots, v_\ell(B, X)\big) = 1.$$

Finally, the invariant component of $\mathcal{D}[k]$ may be taken to be the constant map sending every witness set to the empty string.

For each term τ, write

$$W_i(\tau) := \Gamma[\mathcal{D}_i, k](\tau).$$

We claim that

$$\Gamma[\mathcal{D}, k](\tau) = \big\{\big(W_1(\tau), \dots, W_\ell(\tau)\big)\big\}$$

for every decomposition term τ. This follows by structural induction on τ: for a leaf a, the claim is immediate from the definition of $\mathcal{D}[k].\hat{a}(B(\tau))$; for $\tau = a(\tau_1, \dots, \tau_r)$, the induction hypothesis for the children and the componentwise transition rule yield the stated singleton.

Now fix $k \in \mathbb{N}$, let $\tau \in \mathsf{L}_k$, let $B := B(\tau)$, and write

$$X(\tau) := \big(W_1(\tau), \dots, W_\ell(\tau)\big).$$

Let $G := \mathcal{G}(\tau)$. For each $i \in [\ell]$, Proposition 4 gives

$$b_i(B, X(\tau)) = 1 \quad \Longleftrightarrow \quad G \in \mathcal{P}[\mathcal{D}_i, \mathcal{C}].$$

If this common truth value is 1, then Definition 26 and Condition 2b imply

$$v_i(B, X(\tau)) = \mathcal{I}[\mathcal{D}_i, \mathcal{C}](G) = \mathrm{Inv}^{\perp}_{\mathcal{D}_i, \mathcal{C}}(G).$$

If the common truth value is 0, then by definition

$$v_i(B, X(\tau)) = \perp = \mathrm{Inv}^{\perp}_{\mathcal{D}_i, \mathcal{C}}(G).$$

By the singleton claim above,

$$\begin{aligned} \tau \in \mathsf{Acc}(\mathcal{D}[k]) \quad \Longleftrightarrow \quad & \Phi\big(b_1(B, X(\tau)), \dots, b_\ell(B, X(\tau)), \\ & \qquad v_1(B, X(\tau)), \dots, v_\ell(B, X(\tau))\big) = 1. \end{aligned}$$

Substituting the graph-level interpretations of the b_i and v_i yields

$$\tau \in \mathsf{Acc}(\mathcal{D}[k]) \quad \Longleftrightarrow \quad G \in \hat{\Phi}(\mathcal{P}[\mathcal{D}_1, \mathcal{C}], \dots, \mathcal{P}[\mathcal{D}_\ell, \mathcal{C}], \mathrm{Inv}^{\perp}_{\mathcal{D}_1, \mathcal{C}}, \dots, \mathrm{Inv}^{\perp}_{\mathcal{D}_\ell, \mathcal{C}}). \qquad (*)$$

The right-hand side depends only on the graph G, so if $\tau \in \mathsf{L}_k$ and $\tau' \in \mathsf{L}_{k'}$ realize isomorphic graphs, then $(*)$ gives

$$\tau \in \mathsf{Acc}(\mathcal{D}[k]) \quad \Longleftrightarrow \quad \tau' \in \mathsf{Acc}(\mathcal{D}[k']).$$

Thus the acceptance part of coherence holds for the combined core. Its invariant component is constant, so Definition 25(2) is automatic. Since $\mathcal{D}[k].\Sigma = \Sigma_k$ by construction, the combined core is $\mathcal{C}$-coherent.

Finally, let $G \in \mathcal{P}[\mathcal{C}_k]$ and choose $\tau \in \mathsf{L}_k$ with $\mathcal{G}(\tau) \simeq G$. By $(*)$ and the isomorphism-

invariance of the right-hand side,

$$G \in \mathcal{P}[\mathcal{D}, \mathcal{C}] \quad \Longleftrightarrow \quad G \in \hat{\Phi}(\mathcal{P}[\mathcal{D}_1, \mathcal{C}], \ldots, \mathcal{P}[\mathcal{D}_\ell, \mathcal{C}], \mathrm{Inv}^{\perp}_{\mathcal{D}_1, \mathcal{C}}, \ldots, \mathrm{Inv}^{\perp}_{\mathcal{D}_\ell, \mathcal{C}}),$$

which proves item 1.

For item 2, fix any standard self-delimiting encoding of tuples of finite witness sets, for instance by encoding each component witness set as a delimited list of its witnesses and then concatenating the components with self-delimiting length markers. Encoding the i-th component then requires $O(\beta_{\mathcal{D}_i}(k, n) \cdot \mu_{\mathcal{D}_i}(k, n))$ bits, and the delimiters contribute only linear overhead in ℓ. Hence the total bitlength satisfies the stated $O(\cdot)$ bound. Item 3 holds because every combined transition returns a singleton witness set. Finally, item 4 follows because every useful witness set of the combined core is determined uniquely by a tuple of useful witness sets of the component cores, yielding the stated product bound. □

We refer to the resulting DP-core as the Φ-*combination* of $\mathcal{D}_1, \ldots, \mathcal{D}_\ell$.

5.5.3 Algorithmic Implications

We now consider the algorithmic implications of the combinator construction for model checking and inclusion testing. If all constituent DP-cores are finite and internally polynomial, then so is the Φ-combination: finiteness follows from Theorem 9(2), Lemma 8 supplies the positive-arity lifted-transition step in the combined local simulation, and nullary component transitions are computed using Definition 33(2b). The remaining work is componentwise, together with one additional polynomial-time evaluation of Φ on the component acceptance bits and totalized invariants.

Theorem 10 (Inclusion Test for Combinations)**.** *Let $\mathcal{C}$ be a treelike decomposition class of arity r with a bag-aware automaton $\mathcal{A}$ of complexity $f(k)$, and assume that, once $\mathcal{A}_k$ is constructed, its transitions of arity at most r can be enumerated and queried within the same polynomial-over-$f(k)$ bounds. Let $\mathcal{D}_1, \ldots, \mathcal{D}_\ell$ be finite, internally polynomial, $\mathcal{C}$-coherent treelike DP-cores satisfying Condition 2b. Let Φ be a polynomial ℓ-combinator, and let*

$$\mathcal{D} = \mathcal{D}(\Phi, \mathcal{D}_1, \ldots, \mathcal{D}_\ell).$$

Let $\beta(k) = \max_i \beta_{\mathcal{D}_i}(k)$ and $\mu(k) = \max_i \mu_{\mathcal{D}_i}(k)$. Then one can determine whether

$$\mathcal{P}[\mathcal{C}_k] \subseteq \mathcal{P}[\mathcal{D}, \mathcal{C}]$$

in time

$$k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{O(\ell \cdot r \cdot \beta(k) \cdot \mu(k))} \leq k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{\ell \cdot r \cdot 2^{O(\beta(k))}}.$$

Proof. By Theorem 9, the combined core $\mathcal{D}$ is $\mathcal{C}$-coherent. Under the encoding convention fixed in Theorem 9(2), its witness bitlength satisfies

$$\beta_{\mathcal{D}}(k) = O\left(\sum_{i=1}^{\ell} \beta_{\mathcal{D}_i}(k) \cdot \mu_{\mathcal{D}_i}(k)\right) = O(\ell \cdot \beta(k) \cdot \mu(k)),$$

and its multiplicity is 1. The paragraph above shows that $\mathcal{D}$ is also finite and internally polynomial whenever all constituents are: membership in the combined witness space is checked componentwise; for positive-arity constructors, component lifted transitions are computed by applying Lemma 8 to each component; for nullary constructors, component raw outputs are computed using Definition 33(2b) and then cleaned as in the nullary part of Lemma 8. Finality of the combined core requires only those component outputs together with one polynomial-time evaluation of Φ. Applying Theorem 8 to $\mathcal{D}$ therefore gives

$$k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{O(r \cdot \beta_{\mathcal{D}}(k) \cdot \mu_{\mathcal{D}}(k))} \leq k^{O(1)} \cdot f(k)^{O(r)} \cdot 2^{O(\ell \cdot r \cdot \beta(k) \cdot \mu(k))},$$

and the final displayed inequality follows from $\mu(k) \leq 2^{\beta(k)}$. □

The first bound in Theorem 10 is often the more useful one in mixed combinations: it keeps the factor $\beta(k)\mu(k)$ visible and can exploit cores whose witness bitlength is large but whose multiplicity is small, such as the simplicity core with $\beta(k) = O(k^2)$ and $\mu(k) = 1$. The looser right-hand bound applies the uniform estimate $\mu(k) \leq 2^{O(\beta(k))}$, which is convenient but may overcount in such mixed profiles.

The result shows that combinator constructions preserve inclusion testability under mild complexity assumptions. In typical applications, both r and ℓ are constants, and $\mu(k)$ is moderate, so the exponential dependence on $\beta(k)$ remains the dominant term.

Counterexample Bounds. If the inclusion fails, then a counterexample graph $G \in \mathcal{P}[\mathcal{C}_k] \setminus \mathcal{P}[\mathcal{D}, \mathcal{C}]$ exists and can be represented by a $\mathcal{C}$-decomposition of height at most

$$f(k) \cdot \delta_{\mathcal{D}}(k) \leq f(k) \cdot \prod_{i=1}^{\ell} \delta_{\mathcal{D}_i}(k),$$

by Corollary 4. Hence, the combinator construction provides not only theoretical guarantees but also bounds on minimal counterexamples.

5.6 Applications to Classical Conjectures

The purpose of this section is twofold. First, we catalog families of DP-cores that satisfy the hypotheses of Theorem 10. Second, we quantify the algorithmic consequences for long-standing conjectures that concern graphs of bounded treewidth. These applications mirror the discussion in prior work [de Oliveira Oliveira and Vadiee, 2024], but are expanded here to make explicit how the abstract bounds translate into concrete verification procedures.

Catalog of Treewidth DP-cores

For the instructive decomposition class ITD (Section 3.4), de Oliveira Oliveira and Vadiee [2023, 2024] provide explicit finite, internally polynomial DP-cores for a wide collection of graph properties. Table 5.1 summarizes the asymptotic parameters relevant to Theorem 10. Each entry lists the bitlength $\beta(k)$, multiplicity $\mu(k)$, witness-level state complexity $\nu(k)$, and deterministic witness-set state complexity $\delta_{\mathrm{D}}(k)$ of a core tailored to property $\mathcal{P}$ when evaluated on width-k instructive decompositions. Table 5.1 reports asymptotic upper bounds; some rows use coarse summaries even when sharper bounds are available (notably `Conn`, where Appendix D.1 records a deterministic realization with $\mu(k) = 1$). The Hamiltonian row below refers specifically to the Hamiltonian-cycle (Hamiltonicity) property.

Property $\mathcal{P}$	$\beta(k)$	$\mu(k)$	$\nu(k)$	$\delta_{\mathrm{D}}(k)$
`Simple` ($= \neg$`HasMultipleEdges`)	$O(k^2)$	1	$2^{O(\beta(k))}$	$2^{O(\beta(k))}$
$\texttt{MaxDeg}_{\geq c}$ / $\texttt{MinDeg}_{\leq c}$	$O(k \log c)$	1	$2^{O(\beta(k))}$	$2^{O(\beta(k))}$
`ChromaticNumber`(c)	$O(k \log c)$	$2^{O(\beta(k))}$	$2^{O(\beta(k))}$	$2^{2^{O(\beta(k))}}$
`Conn`	$O(k \log k)$	$2^{O(\beta(k))}$	$2^{O(\beta(k))}$	$2^{2^{O(\beta(k))}}$
$\texttt{VConn}_{\leq c}$ / $\texttt{EConn}_{\leq c}$	$O(\log c + k \log k)$	$2^{O(\beta(k))}$	$2^{O(\beta(k))}$	$2^{2^{O(\beta(k))}}$
`HamiltonianCycle`	$O(k \log k)$	$2^{O(k)}$	$2^{O(\beta(k))}$	$2^{2^{O(k)}}$
`NZFlow`$(\mathbb{Z}_c)$	$O(k \log c)$	$2^{O(\beta(k))}$	$2^{O(\beta(k))}$	$2^{2^{O(\beta(k))}}$
`Minor`(H)	$O(k \log k + \lvert V(H)\rvert + \lvert E(H)\rvert)$	$2^{O(\beta(k))}$	$2^{O(\beta(k))}$	$2^{2^{O(\beta(k))}}$

Table 5.1: Asymptotic complexity parameters for selected ITD-coherent DP-cores. Here $\nu(k)$ denotes the witness-level state complexity (number of useful witnesses), and $\delta_{\mathrm{D}}(k)$ denotes the deterministic witness-set state complexity (number of useful witness sets). The underlying constructions are collected from [Bodlaender et al., 2015; de Oliveira Oliveira and Vadiee, 2023, 2024].

Legend. At width k, $\beta(k)$ is the maximum bitlength of a useful witness, $\mu(k)$ is the maximum size of a useful witness set, $\nu(k)$ is the number of useful witnesses that can appear in some useful set, and $\delta_{\mathrm{D}}(k)$ is the number of useful witness sets. Appendix D.1

explains the witness encoding behind every row, while Sections D.2 and D.3 and the surrounding appendix sections supply the corresponding correctness proofs.

The Hamiltonicity core is noteworthy: thanks to the rank-based compression of Bodlaender et al. [2015], its multiplicity is $2^{O(k)}$, rather than the naive $2^{O(k \log k)}$. This small improvement propagates to the deterministic witness-set state complexity and, consequently, to the runtime of inclusion tests involving Hamiltonicity.

Algorithmic Verification of Conjectures

Combining Table 5.1 with Theorems 7 and 10 yields concrete double-exponential time bounds for checking deep graph-theoretic statements on graphs of bounded treewidth. The following case studies illustrate the workflow.

Hadwiger's Conjecture. A *clique* is a set of pairwise adjacent vertices, and a graph H is a *minor* of a graph G if H can be obtained from a subgraph of G by contracting edges (Appendix B). Hadwiger's conjecture states that for every $c \in \mathbb{N}$, every graph with no K_{c+1} minor is c-colorable. Since its original formulation by Hadwiger [1943], it has been one of the central open problems relating graph minors to coloring. This makes it a natural benchmark for the combination theorem: the premise is an excluded-minor condition and the conclusion is a coloring property.

Corollary 6 (Width-k verification of Hadwiger's conjecture)**.** *For each fixed $c \in \mathbb{N}$, one can decide in time*

$$2^{2^{O(k \log k + c^2)}}$$

whether every graph of treewidth at most k that has no K_{c+1} minor is c-colorable.

Proof. Use the constituent acceptance bit m for $\texttt{Minor}(K_{c+1})$ and the constituent acceptance bit q for $\texttt{ChromaticNumber}(c)$. The implication from the no-minor premise to the coloring conclusion is encoded by the binary combinator

$$\Phi_{\Rightarrow}(m, q, u, v) := m \vee q,$$

which is equivalent to $\neg(\neg m) \vee q$. The coordinates u, v are the totalized invariant coordinates, which are ignored. Since $|V(K_{c+1})| + |E(K_{c+1})| = O(c^2)$, the minor row in Table 5.1 contributes $O(k \log k + c^2)$ to the witness encoding, while the chromatic row contributes $O(k \log c)$. Applying Theorem 10 to the resulting Boolean combination, with the constant value $\ell = 2$, yields the stated runtime bound. □

Compared with a direct MSO encoding via Courcelle's theorem, our bound avoids the tower-type bounds that usually arise from formula translation. It also sharpens the four-level power-tower estimate $p^{p^{p^p}}$ of Kawarabayashi and Reed [2009], where $p = (k+1)^{c-1}$. The gain comes from working with DP-core encodings rather than formula encodings.

Tutte's Flow Conjectures. A nowhere-zero $\mathbb{Z}_c$-flow on a graph is an orientation of the edges together with a nonzero element of $\mathbb{Z}_c$ on each edge such that the net flow at every vertex is zero modulo c (Appendix B). A graph is *bridgeless* if it has no edge whose deletion disconnects the graph. Tutte's 5-flow conjecture, for example, asserts that every bridgeless graph has a nowhere-zero $\mathbb{Z}_5$-flow. The corresponding 4- and 3-flow statements studied in the ATP setting ask whether stronger structural hypotheses force nowhere-zero flows over $\mathbb{Z}_4$ and $\mathbb{Z}_3$, respectively. They form a good ATP testbed because, since the foundational work of Tutte [1954, 1966] and the later flow-theoretic development surveyed by Zhang [1997], they have served as standard benchmark conjectures for nowhere-zero flow methods. Their premises mix connectivity and obstruction conditions, while the conclusion is the single flow predicate $\texttt{NZFlow}(\mathbb{Z}_c)$. Here, a fixed-pattern flow encoding means a concrete Boolean combination, in the premise, of edge-connectivity bounds and finitely many fixed minor predicates, with the conclusion $\texttt{NZFlow}(\mathbb{Z}_c)$ for a fixed $c \in \{3, 4, 5\}$.

Corollary 7 (Width-k verification of Tutte flow conjectures)**.** *For each of the fixed-pattern* 5*-,* 4*-, and* 3*-flow conjecture encodings considered here, one can decide in time*

$$2^{2^{O(k \log k)}}$$

whether the conjecture holds for all graphs of treewidth at most k.

Proof. Each flow conjecture is encoded here as a Boolean combination of edge-connectivity lower bounds, finitely many fixed minor predicates $\texttt{Minor}(P)$, and the flow conclusion $\texttt{NZFlow}(\mathbb{Z}_c)$ for a fixed value of $c \in \{3, 4, 5\}$. For a fixed-pattern encoding, let $\Psi(x_1, \ldots, x_s)$ be the fixed Boolean formula expressing the premise from its s constituent acceptance bits, and use the implication combinator

$$\Phi(x_1, \ldots, x_s, y, \vec{u}) := \neg\Psi(x_1, \ldots, x_s) \vee y,$$

where y is the flow-conclusion bit and $\vec{u}$ collects the ignored totalized invariant coordinates. Here s and hence $\ell = s+1$ are constants depending only on the chosen conjecture encoding. By Table 5.1, the constituent rows used here have $\beta(k) = O(k \log k)$ and $\mu(k) \leq 2^{O(k \log k)}$, hence $\beta(k)\mu(k) \leq 2^{O(k \log k)}$. Equivalently, their deterministic witness-

set state complexities are bounded by $2^{O(\beta(k)\mu(k))} \leq 2^{2^{O(k \log k)}}$. Theorem 10 therefore yields the same asymptotic runtime for the combined property. □

Barnette's Conjecture. Here cubic means 3-regular. A graph is 3-connected if deleting fewer than three vertices leaves it connected. It is bipartite if its vertex set can be partitioned into two independent sets, planar if it admits a crossing-free drawing in the plane, and Hamiltonian if it contains a cycle through all vertices. Barnette's conjecture concerns simple, cubic, 3-connected, bipartite planar graphs and asserts that every such graph is Hamiltonian. This makes it a useful stress test for the combination theorem, because its premise simultaneously mixes degree, connectivity, bipartiteness, planarity, and simplicity constraints, while the conclusion is Hamiltonicity. Since its proposal by Barnette [1969], it has been a standard Hamiltonicity test case at the intersection of planar and bipartite graph structure.

Corollary 8 (Width-k verification of Barnette's conjecture)**.** *One can decide in time*

$$2^{2^{O(k \log k)}}$$

whether every graph of treewidth at most k satisfying Barnette's premise is Hamiltonian.

Proof. Since the ambient instructive-tree-decomposition semantics allows multigraphs, we include the explicit simplicity guard $\texttt{Simple} = \neg\texttt{HasMultipleEdges}$ in addition to the usual premise components: cubicity via $\texttt{MaxDeg}_{\geq 3} \wedge \neg\texttt{MaxDeg}_{\geq 4} \wedge \neg\texttt{MinDeg}_{\leq 2}$, 3-connectedness via $\neg\texttt{VConn}_{\leq 2}$, bipartiteness via $\texttt{ChromaticNumber}(2)$, and planarity via $\neg\texttt{Minor}(K_5) \wedge \neg\texttt{Minor}(K_{3,3})$. Let Ψ denote this fixed Boolean formula for Barnette's premise over the corresponding constituent acceptance bits, and combine it with the Hamiltonicity bit h using the implication combinator

$$\Phi(\vec{x}, h, \vec{u}) := \neg\Psi(\vec{x}) \vee h,$$

again ignoring the totalized invariant coordinates $\vec{u}$. The number of constituent cores is constant. By Table 5.1, each constituent has deterministic witness-set state complexity at most $2^{2^{O(k \log k)}}$, with the Hamiltonicity row contributing the sharper rank-based bound $2^{2^{O(k)}}$. For the first bound in Theorem 10, the same rows satisfy $\beta(k)\mu(k) \leq 2^{O(k \log k)}$: the simplicity guard has $\beta = O(k^2)$ but $\mu = 1$, while the higher-multiplicity rows have $\beta = O(k \log k)$. Applying Theorem 10 to the combined property therefore gives the stated $2^{2^{O(k \log k)}}$ runtime. □

In all three case studies, the width-bounded ATP framework yields more than a yes/no verdict. The resulting corollaries replace generic logic-to-automaton upper bounds by

explicit DP-core-driven double-exponential algorithms on each width-k slice. If the corresponding inclusion fails on that slice, the search produces an explicit DP-refutation together with a concrete bounded-width counterexample term. These bounds should therefore be read as fixed-width completeness guarantees, not as practical runtime predictions; they offer a more concrete state-of-the-art account of what can be certified algorithmically for these conjecture families on bounded-treewidth graphs. Chapter 7 explains how premise-guided exploration is used to mine such witnesses in practice.

Chapter 6

Symmetry Breaking

The refutation engines of Chapter 5 rely on exhaustive exploration of witness sets. In practice many of these states are isomorphic under relabelings of the active bag, leading to a significant amount of redundant work. The state-canonization work [de Oliveira Oliveira and Vadiee, 2024] tackles this issue by introducing symmetry-breaking operators that collapse equivalent states. In this chapter, we develop that line of work in the present framework: we formalize the witness actions induced by relabelings, show that *relabeled refutations* still witness counterexamples, and describe the canonization procedure that is exposed in TreeWidzard. Semi-relabeled refutations appear only as an inductive intermediate (Lemma 15).

Two ideas underpin the construction. Witness actions describe how relabelings operate on the witness space, and canonical forms assign a representative to each relabeling-equivalence class of states. Together they guarantee that restricting to canonical representatives does not lose deductive power (Theorem 12) while substantially reducing the explored state space in practice.

The canonization developed here identifies states only up to permutations of the active bag labels. It does *not* quotient by full graph isomorphism of the realized partial graphs. This distinction matters: bag-label canonization preserves completeness and removes a large redundancy class, but it does not eliminate all duplicate searches.

6.1 Witness Actions and Relabeled Refutations

We now introduce the technical foundation for our symmetry-breaking approach in treewidth-based ATP. In particular, we describe how relabeling functions act on wit-

nesses within a DP-core, how these actions extend to refutations, and why working modulo relabeling does not lose any counterexamples. We begin by formalizing this action, and then show how this leads to a relabeled notion of refutation where symmetry is broken at each step. This generalizes the instructive $(k, \mathcal{D})$-refutation notion of Definition 46.

Throughout this chapter we specialize to the instructive decomposition class ITD of Section 3.4. Thus, whenever we speak of a DP-core in this chapter, we mean an *instructive* DP-core, and coherence is always understood with respect to ITD. In the width-based ATP regime we allow root-open instructive terms, so the recognizer state at a subterm is just its current bag $B \subseteq [k+1]$, and every such bag state is treated as final. Accordingly, the generic Chapter 5 refutation states (q, S) can here be read simply as bag-witness pairs (B, S).

To keep notation readable, we suppress the explicit bag parameter carried by the Chapter 4 transition interface. Thus expressions such as $\mathcal{D}[k].\widehat{\mathsf{intro}}_u(W)$, $\mathcal{D}[k].\widehat{\mathsf{forget}}_u(W)$, $\mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W)$, and $\mathcal{D}[k].\widehat{\mathsf{join}}(W, W')$ are always read in the bag-aware sense determined by the surrounding state pair(s). Equivalently, one may read this chapter in the bag-annotated style made explicit in Chapter 9. In particular, join expressions are considered only when the two operand states have the same current bag (or have first been aligned to the same bag by a relabeling).

6.1.1 Witness Actions

We begin by defining how relabeling functions act on the local witness structure of an instructive DP-core. These actions formalize the symmetries induced by renaming active bag labels and are the key ingredient behind the canonical-state constructions of Section 6.2.

We fix a treewidth parameter $k \in \mathbb{N}$, and consider relabeling functions over the universe $[k+1]$. These are injective maps from subsets of $[k+1]$ into $[k+1]$, denoted $\mathcal{R}_k$. Since every $f \in \mathcal{R}_k$ is injective, it has a partial inverse

$$f^{-1} : \mathrm{im}(f) \to \mathrm{dom}(f).$$

Let $\mathcal{D}$ be an instructive DP-core and let $w \in \mathcal{D}[k].\mathcal{W}$. We write $\mathrm{supp}(w) \subseteq [k+1]$ for the set of labels occurring in w.

Formally, the DP-core interface of Chapter 4 evaluates finality relative to the current bag. Accordingly, in the formal statements below we write $\mathcal{D}[k].\texttt{Final}(\mathbf{b}, w)$, where

$\mathbf{b} \subseteq [k+1]$ is the current bag. When a bag $\mathbf{b}$ is fixed by context, we may abbreviate $\mathcal{D}[k].\texttt{Final}(\mathbf{b}, w)$ to $\mathcal{D}[k].\texttt{Final}(w)$.

Definition 41 (Witness Action)**.** *Let $\mathcal{D}$ be an instructive DP-core and $k \in \mathbb{N}$. A partial function*

$$\rho^k_{\mathcal{D}} : \{\, (f, w) \in \mathcal{R}_k \times \mathcal{D}[k].\mathcal{W} \mid \operatorname{supp}(w) \subseteq \operatorname{dom}(f) \,\} \to \mathcal{D}[k].\mathcal{W}$$

is said to be a witness action for $\mathcal{D}[k]$ if the following conditions are satisfied.

1. ***Finality invariance:*** *for all bags $\mathbf{b} \subseteq [k+1]$, all $w \in \mathcal{D}[k].\mathcal{W}$, and all $f \in \mathcal{R}_k$ with*

   $$\operatorname{supp}(w) \subseteq \mathbf{b} \subseteq \operatorname{dom}(f),$$

   we have

   $$\mathcal{D}[k].\texttt{Final}(\mathbf{b}, w) = 1 \iff \mathcal{D}[k].\texttt{Final}(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, w)) = 1.$$

2. ***Inversion:*** *for all $w \in \mathcal{D}[k].\mathcal{W}$ and all $f \in \mathcal{R}_k$ with $\operatorname{supp}(w) \subseteq \operatorname{dom}(f)$,*

   $$\rho^k_{\mathcal{D}}\big(f^{-1}, \rho^k_{\mathcal{D}}(f, w)\big) = w.$$

3. ***Composition:*** *for all $w \in \mathcal{D}[k].\mathcal{W}$ and all $f, f' \in \mathcal{R}_k$ such that*

   $$\operatorname{supp}(w) \subseteq \operatorname{dom}(f') \qquad \text{and} \qquad f'(\operatorname{supp}(w)) \subseteq \operatorname{dom}(f),$$

   we have

   $$\rho^k_{\mathcal{D}}(f \circ f', w) = \rho^k_{\mathcal{D}}\big(f, \rho^k_{\mathcal{D}}(f', w)\big).$$

4. ***Label support transport:*** *for all $w \in \mathcal{D}[k].\mathcal{W}$ and all $f \in \mathcal{R}_k$ with $\operatorname{supp}(w) \subseteq \operatorname{dom}(f)$,*

   $$\operatorname{supp}(\rho^k_{\mathcal{D}}(f, w)) = f(\operatorname{supp}(w)).$$

5. ***Extension invariance:*** *if $f_0 \in \mathcal{R}_k$ extends f (i.e. $\operatorname{dom}(f) \subseteq \operatorname{dom}(f_0)$ and $f_0|_{\operatorname{dom}(f)} = f$), then for every $w \in \mathcal{D}[k].\mathcal{W}$ with $\operatorname{supp}(w) \subseteq \operatorname{dom}(f)$,*

   $$\rho^k_{\mathcal{D}}(f_0, w) = \rho^k_{\mathcal{D}}(f, w).$$

Definition 42 (Pointwise extension to witness sets)**.** *Given $f \in \mathcal{R}_k$ and a finite witness set $W \subseteq \mathcal{D}[k].\mathcal{W}$ such that*

$$\forall w \in W :\ \operatorname{supp}(w) \subseteq \operatorname{dom}(f) \qquad \textit{(equivalently, } \bigcup_{w \in W} \operatorname{supp}(w) \subseteq \operatorname{dom}(f)\textit{)},$$

define

$$\rho^k_{\mathcal{D}}(f, W) = \{\, \rho^k_{\mathcal{D}}(f, w) \mid w \in W \,\}.$$

For a finite witness set W, write

$$\operatorname{supp}(W) := \bigcup_{w \in W} \operatorname{supp}(w).$$

A pair $(\mathbf{b}, W)$ is called *well-formed* if every witness $w \in W$ satisfies $\operatorname{supp}(w) \subseteq \mathbf{b}$.

Lemma 9 (Set-level witness-action laws)**.** *Let $\mathcal{D}$ be equipped with a witness action. Let $f, g \in \mathcal{R}_k$, and let $W \subseteq \mathcal{D}[k].\mathcal{W}$ be a finite witness set. Then the clauses of Definition 41 lift pointwise to W. In particular:*

1. *If* $\operatorname{supp}(W) \subseteq \operatorname{dom}(f)$*, then*

$$\operatorname{supp}\big(\rho^k_{\mathcal{D}}(f, W)\big) = f(\operatorname{supp}(W)).$$

2. *If* $\operatorname{supp}(W) \subseteq \operatorname{dom}(f)$*, then*

$$\rho^k_{\mathcal{D}}\big(f^{-1}, \rho^k_{\mathcal{D}}(f, W)\big) = W.$$

3. *If* $\operatorname{supp}(W) \subseteq \operatorname{dom}(g)$ *and* $g(\operatorname{supp}(W)) \subseteq \operatorname{dom}(f)$*, then*

$$\rho^k_{\mathcal{D}}(f \circ g, W) = \rho^k_{\mathcal{D}}\big(f, \rho^k_{\mathcal{D}}(g, W)\big).$$

4. *If $f_0 \in \mathcal{R}_k$ extends f and* $\operatorname{supp}(W) \subseteq \operatorname{dom}(f)$*, then*

$$\rho^k_{\mathcal{D}}(f_0, W) = \rho^k_{\mathcal{D}}(f, W).$$

Proof. Each displayed statement follows by applying the corresponding clause of Definition 41 to every $w \in W$, using the pointwise definition of Definition 42. Equality is extensional as equality of finite sets of witnesses. □

Definition 43 (Compatibility with lifted DP-core operations)**.** *A witness action $\rho^k_{\mathcal{D}}$ is compatible with the lifted DP-core operations if the following bag-aware equalities hold whenever both sides are well-defined. Here $\mathcal{D}[k].\hat{a}(B, \dots)$ denotes the lifted transition with current top bag B, as in Chapter 4.*

1. *For every well-formed $(\mathbf{b}, W)$, every $u \in [k+1] \setminus \mathbf{b}$, and every $f : \mathbf{b} \cup \{u\} \hookrightarrow [k+1]$,*

$$\rho^k_{\mathcal{D}}\big(f, \mathcal{D}[k].\widehat{\mathsf{intro}}_u(\mathbf{b} \cup \{u\}, W)\big) = \mathcal{D}[k].\widehat{\mathsf{intro}}_{f(u)}\big(f(\mathbf{b} \cup \{u\}), \rho^k_{\mathcal{D}}(f|_{\mathbf{b}}, W)\big).$$

2. *For every well-formed* $(\mathbf{b}, W)$*, every* $u \in \mathbf{b}$*, and every* $f : \mathbf{b} \hookrightarrow [k+1]$*,*

$$\rho^k_{\mathcal{D}}\big(f|_{\mathbf{b}\setminus\{u\}}, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(\mathbf{b}\setminus\{u\}, W)\big) = \mathcal{D}[k].\widehat{\mathsf{forget}}_{f(u)}\big(f(\mathbf{b}\setminus\{u\}), \rho^k_{\mathcal{D}}(f, W)\big).$$

3. *For every well-formed* $(\mathbf{b}, W)$*, all distinct* $u, v \in \mathbf{b}$*, and every* $f : \mathbf{b} \hookrightarrow [k+1]$*,*

$$\rho^k_{\mathcal{D}}\big(f, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(\mathbf{b}, W)\big) = \mathcal{D}[k].\widehat{\mathsf{intro}}_{f(u),f(v)}\big(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, W)\big).$$

4. *For all well-formed pairs* $(\mathbf{b}, W)$ *and* $(\mathbf{b}, W')$ *over the same bag, and every* $f : \mathbf{b} \hookrightarrow [k+1]$*,*

$$\rho^k_{\mathcal{D}}\big(f, \mathcal{D}[k].\widehat{\mathsf{join}}(\mathbf{b}, W, W')\big) = \mathcal{D}[k].\widehat{\mathsf{join}}\big(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, W), \rho^k_{\mathcal{D}}(f, W')\big).$$

5. *For every* $f \in \mathcal{R}_k$*,*

$$\rho^k_{\mathcal{D}}\big(f, \mathcal{D}[k].\widehat{\mathsf{leaf}}(\emptyset)\big) = \mathcal{D}[k].\widehat{\mathsf{leaf}}(\emptyset).$$

Remark 8. In the remainder, when we say that $\mathcal{D}[k]$ is *equipped with a witness action*, we mean a witness action in the sense of Definition 41, extended pointwise to witness sets as in Definition 42, and compatible with the lifted DP-core operations in the sense of Definition 43.

Remark 9. The results below are therefore conditional on this witness-action package. For a concrete DP-core, one must separately define the relabeling action on witnesses and verify Definitions 41–43 before invoking the symmetry-breaking theorems of this chapter. These core-specific verification obligations are not automatic consequences of the abstract framework.

Remark 10 (Bridge to the concrete catalog). For the concrete cores used later in the thesis, the relevant witness action is always the relabeling obtained by renaming the bag-indexed witness data. Chapter 9 records this explicitly for the chromatic and vertex-cover cores, and Appendix D does the same for the remaining catalog cores used in Chapters 8 and 10. Thus the symmetry-breaking results of the present chapter apply only after consulting those core-specific relabeling/equivariance paragraphs and checking that they provide the witness-level action required here.

Lemma 10 (From permutation actions to partial relabelings)**.** *Fix* $k \in \mathbb{N}$*. Assume that, at width* k*, the witness space* $\mathcal{D}[k].\mathcal{W}$ *is equipped with:*

1. *a support map* $\mathrm{supp} : \mathcal{D}[k].\mathcal{W} \to \mathcal{P}_{\mathrm{fin}}([k+1])$*,*

2. *an action of the permutation group* S_{k+1} *on witnesses, written* $(\rho, w) \mapsto \rho \cdot w$*,*

3. *support transport and support locality:*

$$\mathrm{supp}(\rho \cdot w) = \rho(\mathrm{supp}(w)),$$

and whenever $\rho, \pi \in S_{k+1}$ *agree on* $\mathrm{supp}(w)$*, we have* $\rho \cdot w = \pi \cdot w$*,*

4. *permutation-equivariance of finality and of the cleaned lifted DP-core operations: relabeling the bag, the constructor parameters, and all witnesses by a permutation has the same effect as first applying the cleaned lifted DP-core operation and then relabeling the result.*

Then every partial injective relabeling $f \in \mathcal{R}_k$ *induces a well-defined witness action on witnesses with support contained in* $\mathrm{dom}(f)$ *by*

$$\rho^k_{\mathcal{D}}(f, w) := \rho \cdot w,$$

where $\rho \in S_{k+1}$ *is any permutation extending* f *on* $\mathrm{supp}(w)$*. This action satisfies Definitions 41–43.*

Proof. Because f is an injective map between finite subsets of $[k+1]$, it extends to some permutation $\rho \in S_{k+1}$. If ρ and π are two such extensions agreeing with f on $\mathrm{supp}(w)$, then support locality gives $\rho \cdot w = \pi \cdot w$, so the definition is independent of the chosen extension.

Support transport is inherited directly from the permutation action:

$$\mathrm{supp}(\rho^k_{\mathcal{D}}(f, w)) = \mathrm{supp}(\rho \cdot w) = \rho(\mathrm{supp}(w)) = f(\mathrm{supp}(w)).$$

Finality invariance follows in the same way, because for every bag $\mathbf{b} \supseteq \mathrm{supp}(w)$ and every permutation ρ extending f on $\mathbf{b}$,

$$\mathcal{D}[k].\texttt{Final}(\mathbf{b}, w) \iff \mathcal{D}[k].\texttt{Final}(\rho(\mathbf{b}), \rho \cdot w) \iff \mathcal{D}[k].\texttt{Final}(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, w)).$$

For inversion and composition, choose permutation extensions compatible with the relevant supports; the group identities $\rho^{-1}\rho = \mathrm{id}$ and $(\rho \circ \pi) \cdot w = \rho \cdot (\pi \cdot w)$, together with support locality, yield the corresponding clauses for partial relabelings. Extension invariance is immediate from well-definedness: if f_0 extends f, then any permutation extending f_0 also extends f on $\mathrm{supp}(w)$.

It remains to check compatibility with lifted operations. Whenever one of the equalities in Definition 43 is well-defined, choose a permutation $\rho \in S_{k+1}$ extending the relevant

partial relabeling on the current bag, on the constructor labels, and on the supports of all witnesses involved. By the assumed permutation-equivariance of the corresponding cleaned lifted operation, applying ρ after the operation equals first relabeling every argument by ρ and then applying the relabeled operation. Replacing ρ by the induced partial map using well-definedness gives exactly the required compatibility clause. The leaf clause is the nullary instance of the same argument. □

Remark 11. Lemma 10 is the formal bridge between the permutation-equivariance paragraphs proved later for concrete cores and the partial relabeling framework of the present chapter. In each concrete instantiation one must still verify the support-locality requirement, i.e. that a permutation fixing all labels in the support of a witness fixes the witness itself.

The clauses above ensure that relabeling functions act consistently on the witness space. To preclude any ambiguity regarding domains and ranges, we record the following typing property, which justifies the well-definedness of these compositions.

Lemma 11 (Typing and Well-Definedness)**.** *Assume $\mathcal{D}[k]$ is equipped with a witness action. Let $f \in \mathcal{R}_k$ and $w \in \mathcal{D}[k].\mathcal{W}$. If $\mathrm{supp}(w) \subseteq \mathrm{dom}(f)$, then $\rho^k_{\mathcal{D}}(f, w)$ is well-defined and satisfies*

$$\mathrm{supp}(\rho^k_{\mathcal{D}}(f, w)) = f(\mathrm{supp}(w)) \subseteq \mathrm{im}(f).$$

Consequently, $\rho^k_{\mathcal{D}}(f^{-1}, \rho^k_{\mathcal{D}}(f, w))$ is well-defined. Moreover, if $f_0 \in \mathcal{R}_k$ extends f (i.e. $\mathrm{dom}(f) \subseteq \mathrm{dom}(f_0)$ and $f_0|_{\mathrm{dom}(f)} = f$), then

$$\rho^k_{\mathcal{D}}(f_0, w) = \rho^k_{\mathcal{D}}(f, w).$$

Finally, the composition $\rho^k_{\mathcal{D}}(f \circ g, w)$ is well-defined whenever

$$\mathrm{supp}(w) \subseteq \mathrm{dom}(g) \qquad \text{and} \qquad g(\mathrm{supp}(w)) \subseteq \mathrm{dom}(f).$$

Proof. The first statement is immediate from the hypotheses and the Label Support Transport axiom in Definition 41: since $\mathrm{supp}(w) \subseteq \mathrm{dom}(f)$, the witness $\rho^k_{\mathcal{D}}(f, w)$ is defined and satisfies

$$\mathrm{supp}(\rho^k_{\mathcal{D}}(f, w)) = f(\mathrm{supp}(w)) \subseteq \mathrm{im}(f).$$

Because f is injective, $\mathrm{im}(f) = \mathrm{dom}(f^{-1})$, so $\rho^k_{\mathcal{D}}(f^{-1}, \rho^k_{\mathcal{D}}(f, w))$ is well-defined.

If f_0 extends f, the displayed equality $\rho^k_{\mathcal{D}}(f_0, w) = \rho^k_{\mathcal{D}}(f, w)$ is exactly the Extension Invariance axiom.

Finally, if $\mathrm{supp}(w) \subseteq \mathrm{dom}(g)$ and $g(\mathrm{supp}(w)) \subseteq \mathrm{dom}(f)$, then Label Support Transport gives

$$\mathrm{supp}(\rho^k_{\mathcal{D}}(g, w)) = g(\mathrm{supp}(w)) \subseteq \mathrm{dom}(f),$$

so $\rho^k_{\mathcal{D}}(f, \rho^k_{\mathcal{D}}(g, w))$ is well-defined. The Composition axiom then shows that $\rho^k_{\mathcal{D}}(f \circ g, w)$ is also well-defined and equal to that composition. □

Lemma 12 (Identity relabeling acts trivially)**.** *Assume $\mathcal{D}[k]$ is equipped with a witness action. Let $A \subseteq [k+1]$, let $\mathrm{id}_A : A \to A$ be the identity relabeling, and let $w \in \mathcal{D}[k].\mathcal{W}$ satisfy $\mathrm{supp}(w) \subseteq A$. Then*

$$\rho^k_{\mathcal{D}}(\mathrm{id}_A, w) = w.$$

Consequently, for every finite witness set $W \subseteq \mathcal{D}[k].\mathcal{W}$ such that $\mathrm{supp}(w) \subseteq A$ for all $w \in W$,

$$\rho^k_{\mathcal{D}}(\mathrm{id}_A, W) = W.$$

Proof. Since $\mathrm{id}_A^{-1} = \mathrm{id}_A$ and $\mathrm{id}_A \circ \mathrm{id}_A = \mathrm{id}_A$, the Composition clause of Definition 41 gives

$$\rho^k_{\mathcal{D}}(\mathrm{id}_A, w) = \rho^k_{\mathcal{D}}\big(\mathrm{id}_A, \rho^k_{\mathcal{D}}(\mathrm{id}_A, w)\big).$$

By Label Support Transport,

$$\mathrm{supp}(\rho^k_{\mathcal{D}}(\mathrm{id}_A, w)) = \mathrm{id}_A(\mathrm{supp}(w)) \subseteq A,$$

so the outer action in the displayed equality is well-defined. Since $\mathrm{supp}(w) \subseteq A = \mathrm{dom}(\mathrm{id}_A)$, the Inversion clause with $f = \mathrm{id}_A$ yields

$$\rho^k_{\mathcal{D}}\big(\mathrm{id}_A, \rho^k_{\mathcal{D}}(\mathrm{id}_A, w)\big) = w.$$

Combining the two displayed equalities gives $\rho^k_{\mathcal{D}}(\mathrm{id}_A, w) = w$. The witness-set statement follows immediately from the pointwise definition of $\rho^k_{\mathcal{D}}(\mathrm{id}_A, W)$. □

Definition 44 (Well-formedness preservation)**.** *Let $\mathcal{D}$ be an instructive DP-core and let $k \in \mathbb{N}$. We say that $\mathcal{D}[k]$ preserves well-formedness if:*

1. *the initial pair $(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$ is well-formed,*
2. *whenever $(\mathbf{b}, W)$ is well-formed and $u \notin \mathbf{b}$, the pair*

$$(\mathbf{b} \cup \{u\}, \mathcal{D}[k].\widehat{\mathsf{intro}}_u(W))$$

is well-formed,

3. *whenever* $(\mathbf{b}, W)$ *is well-formed and* $u \in \mathbf{b}$*, the pair*

$$(\mathbf{b} \setminus \{u\}, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(W))$$

is well-formed,

4. *whenever* $(\mathbf{b}, W)$ *is well-formed and* $u, v \in \mathbf{b}$ *are distinct, the pair*

$$(\mathbf{b}, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W))$$

is well-formed,

5. *whenever* $(\mathbf{b}, W)$ *and* $(\mathbf{b}, W')$ *are well-formed, the pair*

$$(\mathbf{b}, \mathcal{D}[k].\widehat{\mathsf{join}}(W, W'))$$

is well-formed.

We say that $\mathcal{D}$ *preserves well-formedness if* $\mathcal{D}[k]$ *does so for every* $k \in \mathbb{N}$*.*

Lemma 13 (Relabelings preserve well-formedness)**.** *Assume* $\mathcal{D}[k]$ *is equipped with a witness action. Let* $(\mathbf{b}, W)$ *be a well-formed pair, and let* $f \in \mathcal{R}_k$ *satisfy* $\mathrm{dom}(f) = \mathbf{b}$*. Then*

$$(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, W))$$

is well-formed.

Proof. For every $w \in W$, well-formedness gives $\mathrm{supp}(w) \subseteq \mathbf{b} = \mathrm{dom}(f)$. By Label Support Transport,

$$\mathrm{supp}(\rho^k_{\mathcal{D}}(f, w)) = f(\mathrm{supp}(w)) \subseteq f(\mathbf{b}).$$

Applying this pointwise to all $w \in W$ proves the claim. □

Remark 12. Chapter 4 does not force well-formedness preservation at the abstract interface level. In the present chapter we therefore assume it explicitly whenever witness actions or canonization are applied to newly generated DP states. For the catalog cores, this closure property is recorded as a core-specific verification obligation in Chapter 9 and Appendix D.

Example 2. *Let* D *denote the* 2*-colorability DP-core from Chapter 4. On the bag* $\mathbf{b} = \{1, 2\}$*, a witness is a proper* 2*-coloring of the current bag, written as an ordered pair of color classes. After the edge between labels* 1 *and* 2 *has been introduced, the surviving witness set is*

$$W = \big\{(\{1\}, \{2\}),\ (\{2\}, \{1\})\big\}.$$

Let $f : \mathbf{b} \to \mathbf{b}$ be the relabeling defined by $f(1) = 2$ and $f(2) = 1$. Then

$$\rho_D^1\big(f, (\{1\}, \{2\})\big) = (\{2\}, \{1\}), \qquad \rho_D^1\big(f, (\{2\}, \{1\})\big) = (\{1\}, \{2\}).$$

Hence

$$\rho_D^1(f, W) = W.$$

So the relabeling does not change the coloring information; it only renames the active bag labels appearing in the witnesses. This is exactly the symmetry exploited later by canonization: states that differ only by a renaming of the active bag labels should be identified during search.

Definition 45 ($(k, \mathcal{D})$-inconsistent pair)**.** *Let $\mathcal{D}$ be an instructive DP-core and $k \in \mathbb{N}$. A well-formed pair $(\mathbf{b}, W)$, where $\mathbf{b} \subseteq [k+1]$ and $W \subseteq \mathcal{D}[k].\mathcal{W}$ is finite, is called $(k, \mathcal{D})$-inconsistent if*

$$\forall w \in W : \ \mathcal{D}[k].\texttt{Final}(\mathbf{b}, w) = 0.$$

Definition 46 ($(k, \mathcal{D})$-refutation)**.** *Let $\mathcal{D}$ be an instructive DP-core and $k \in \mathbb{N}$. A $(k, \mathcal{D})$-refutation is a finite sequence of well-formed pairs*

$$R \equiv (\mathbf{b}_0, W_0)(\mathbf{b}_1, W_1) \dots (\mathbf{b}_m, W_m)$$

satisfying the following conditions:

1. ***Initialization:***
$$(\mathbf{b}_0, W_0) = (\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}()).$$
2. ***Refutation:*** *The final pair $(\mathbf{b}_m, W_m)$ is $(k, \mathcal{D})$-inconsistent.*
3. ***Progression:*** *For each $i \in [m]$, there exists $j < i$ such that $(\mathbf{b}_i, W_i)$ is equal to one of the following pairs:*
   (a) $(\mathbf{b}_j \cup \{u\}, \mathcal{D}[k].\widehat{\mathsf{intro}}_u(W_j))$ *for some $u \notin \mathbf{b}_j$.*
   (b) $(\mathbf{b}_j \setminus \{u\}, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(W_j))$ *for some $u \in \mathbf{b}_j$.*
   (c) $(\mathbf{b}_j, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W_j))$ *for some distinct $u, v \in \mathbf{b}_j$.*
   (d) $(\mathbf{b}_j, \mathcal{D}[k].\widehat{\mathsf{join}}(W_j, W_\ell))$ *for some $\ell < i$ with $\mathbf{b}_\ell = \mathbf{b}_j$, so the join is well-defined on a common current bag.*

Lemma 14 (Leaf normalization of specialized generic refutations)**.** *Let $\mathcal{D}$ be an instructive DP-core and let $k \in \mathbb{N}$. Let*

$$R \equiv (\mathbf{b}_1, W_1), \dots, (\mathbf{b}_m, W_m)$$

be a sequence that satisfies Definition 37 after specializing $\mathcal{C}$ *to* ITD *and identifying each recognizer state with the current bag. Then there exists another such sequence with the same final pair such that:*

1. *the first pair is* $(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$*, and*
2. *no later pair is introduced by the leaf clause.*

Proof. In the specialized generic definition, the first pair cannot satisfy the composition clause because there are no earlier predecessors. Hence it must satisfy the leaf clause. The instructive alphabet has exactly one nullary symbol, `leaf`, and its unique recognizer transition produces the empty bag. Therefore

$$(\mathbf{b}_1, W_1) = (\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}()).$$

Now let $i > 1$ be any later index introduced by the leaf clause. The same uniqueness argument shows

$$(\mathbf{b}_i, W_i) = (\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}()) = (\mathbf{b}_1, W_1).$$

If the final pair is itself such a later leaf occurrence, then the one-state sequence consisting only of $(\mathbf{b}_1, W_1)$ has the same final pair and satisfies the desired normalization. We may therefore assume that no later leaf occurrence is final. Replacing every reference to index i inside later composition justifications by the first index 1 preserves constructibility, because the referenced pair is unchanged. After deleting all such redundant later leaf occurrences, we obtain the required normalized sequence with the same final pair. □

The generic ATP refutation theorem from Chapter 5 now specializes cleanly to instructive bag-states.

Corollary 9 (Instructive Refutation Characterization)**.** *Let* $\mathcal{D}$ *be an* ITD*-coherent instructive DP-core that preserves well-formedness. Then, for every* $k \in \mathbb{N}$*, there exists a* $(k, \mathcal{D})$*-refutation if and only if there exists a graph of treewidth at most* k *that does not belong to* $\mathcal{P}[\mathcal{D}, \mathsf{ITD}]$*.*

Proof. Apply Theorem 6 with $\mathcal{C} = \mathsf{ITD}$ and with the recognizer sequence $\{\mathcal{A}_k\}_{k\in\mathbb{N}}$ from Section 3.4. For instructive terms, a state of $\mathcal{A}_k$ can be identified with the current bag $B \subseteq [k+1]$ [de Oliveira Oliveira and Vadiee, 2023, Section 3.3]; under the root-open convention fixed above, every such bag state is final. Under this identification, the generic inconsistency notion of Definition 36 becomes exactly the bag-based inconsistency notion introduced above. The only remaining mismatch is that the generic Chapter 5 definition

permits redundant later leaf-clause entries, whereas Definition 46 fixes the unique leaf pair as the initial state. By Lemma 14, every specialized generic refutation normalizes to one whose first state is exactly that initial pair and whose remaining steps use only constructor clauses. Here the only nullary instructive symbol is `leaf`, whose unique recognizer transition produces the bag $\emptyset$, while the remaining clauses are exactly the four instructive constructors. Hence the normalized specialized refutations are exactly the $(k, \mathcal{D})$-refutations of Definition 46.

The graph class $\mathcal{P}[\mathsf{ITD}_k]$ is exactly the class of graphs of treewidth at most k by Lemma 2. Therefore the specialized version of Theorem 6 yields the stated equivalence, once we note that the specialized witness-set sequence is well-formed: the initial state $(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$ is well-formed by Definition 44(1), and every later instructive constructor step preserves well-formedness by Definition 44(2)–(5). Hence the generic specialized refutations are exactly the $(k, \mathcal{D})$-refutations defined above. □

6.1.2 Relabeled Refutations

Having defined witness actions, we now show how they can be used to compactify refutations via symmetry-aware reasoning. The key idea is to allow each step in a width-based refutation to be followed by an appropriate relabeling. This lets us reuse previous states under symmetry and focus only on canonical representatives. The result is a *relabeled refutation*, which generalizes the instructive $(k, \mathcal{D})$-refutation from Definition 46.

In a relabeled refutation, each transition may apply a relabeling function that renames the bag and transforms the associated witness set using a witness action. These relabelings do not alter the semantics of the proof; they preserve soundness while reducing the search space by collapsing symmetric paths.

Definition 47 (F-Relabeled Refutation)**.** *Let* $\mathrm{F} = (f_1, \ldots, f_m)$ *be a sequence of* k*-relabeling functions, and let* $\mathcal{D}$ *be an instructive DP-core. A sequence of well-formed pairs*

$$R \equiv (\mathbf{b}_0, W_0)(\mathbf{b}_1, W_1) \ldots (\mathbf{b}_m, W_m)$$

is an F*-relabeled* $(k, \mathcal{D})$*-refutation if the following conditions hold:*

1. ***Initialization:*** $(\mathbf{b}_0, W_0) = (\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$.
2. ***Refutation:*** *The final state* $(\mathbf{b}_m, W_m)$ *is* $(k, \mathcal{D})$*-inconsistent. Equivalently,*

$$\forall w \in W_m : \ \mathcal{D}[k].\mathtt{Final}(\mathbf{b}_m, w) = 0.$$

3. ***Progression via relabeling:*** *For each $i \in [m]$, the state $(\mathbf{b}_i, W_i)$ arises from some earlier state $(\mathbf{b}_j, W_j)$ for $j < i$ via a valid DP-core operation followed by a relabeling f_i. Specifically, one of the following must hold:*

   (a) $(\mathbf{b}_i, W_i) = (f_i(\mathbf{b}_j \cup \{u\}), \rho^k_{\mathcal{D}}(f_i, \mathcal{D}[k].\widehat{\mathsf{intro}}_u(W_j)))$*, for some $u \notin \mathbf{b}_j$ and some k-relabeling function $f_i : \mathbf{b}_j \cup \{u\} \to [k+1]$ with image $\mathbf{b}_i$.*

   (b) $(\mathbf{b}_i, W_i) = (f_i(\mathbf{b}_j \setminus \{u\}), \rho^k_{\mathcal{D}}(f_i, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(W_j)))$*, for some $u \in \mathbf{b}_j$ and some k-relabeling function $f_i : \mathbf{b}_j \setminus \{u\} \to [k+1]$ with image $\mathbf{b}_i$.*

   (c) $(\mathbf{b}_i, W_i) = (f_i(\mathbf{b}_j), \rho^k_{\mathcal{D}}(f_i, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W_j)))$*, for some distinct $u, v \in \mathbf{b}_j$ and some k-relabeling function $f_i : \mathbf{b}_j \to [k+1]$ with image $\mathbf{b}_i$.*

   (d) $(\mathbf{b}_i, W_i) = (f_i(\mathbf{b}_j), \rho^k_{\mathcal{D}}(f_i, \mathcal{D}[k].\widehat{\mathsf{join}}(W_j, \rho^k_{\mathcal{D}}(h, W_l))))$*, for some $j, l < i$, some k-relabeling function $f_i : \mathbf{b}_j \to [k+1]$ with image $\mathbf{b}_i$, and some relabeling function $h : \mathbf{b}_l \to \mathbf{b}_j$ with $h(\mathbf{b}_l) = \mathbf{b}_j$. In particular, h is a bijection between the two predecessor bags, so the ordered join is well-typed.*

We refer to the sequence F as the *relabeling trace* of the refutation. If a sequence satisfies the initialization and progression conditions (1 and 3), but not necessarily the inconsistency condition (2), we refer to it as a *semi-*F*-relabeled refutation*. These play an important role in inductive constructions and the proof of soundness.

Relabeled refutations allow us to simulate inference steps under symmetry, facilitating state-space pruning. Later, in Section 6.2, we will use this concept to define and reason about *canonical refutations*, where only one representative per equivalence class is retained. Lemma 15 below shows that every semi-relabeled refutation can be realized by an actual tree decomposition with the expected witness dynamics.

Lemma 15. *Let $\mathcal{D}$ be an instructive DP-core equipped with a witness action in the sense of Definitions 41–43, let $\mathrm{F} = (f_1, \ldots, f_m)$ be a sequence of k-relabeling functions, and let*

$$R = (\mathbf{b}_0, W_0)(\mathbf{b}_1, W_1) \ldots (\mathbf{b}_m, W_m)$$

*be a semi-*F*-relabeled $(k, \mathcal{D})$-refutation. Further, let $g : \mathbf{b}_m \to [k+1]$ be a k-relabeling function.*

Then there exists a k-instructive tree decomposition $\tau_R \in \mathsf{ITD}_k$, and a map

$$T : \mathrm{Sub}(\tau_R) \to \mathcal{P}_{\mathrm{fin}}(\mathcal{D}[k].\mathcal{W})$$

where $\mathrm{Sub}(\tau_R)$ denotes the set of subterm occurrences of τ_R. Formally, occurrences are positions in the term tree. Thus, for a join term $\mathtt{join}(\sigma_1, \sigma_2)$, the occurrence set is the

disjoint union of the root occurrence, the left-tagged occurrences of σ_1, and the right-tagged occurrences of σ_2. In the clauses below we write an occurrence by the rooted subterm it induces when the occurrence tag is clear from context. With this convention, the following conditions are satisfied for every subterm occurrence τ of τ_R:

1. *If $\tau = \tau_R$, then $B(\tau) = g(\mathbf{b}_m)$ and $T(\tau) = \rho^k_{\mathcal{D}}(g, W_m)$.*
2. *If $\tau =$ `leaf`, then $T(\tau) = \mathcal{D}[k].\widehat{\mathsf{leaf}}()$.*
3. *If $\tau =$ `intro_`$u(\tau_1)$, then $T(\tau) = \mathcal{D}[k].\widehat{\mathsf{intro}}_u(T(\tau_1))$.*
4. *If $\tau =$ `forget_`$u(\tau_1)$, then $T(\tau) = \mathcal{D}[k].\widehat{\mathsf{forget}}_u(T(\tau_1))$.*
5. *If $\tau =$ `intro_`u`_`$v(\tau_1)$, then $T(\tau) = \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(T(\tau_1))$.*
6. *If $\tau =$ `join`(τ_1, τ_2), then $T(\tau) = \mathcal{D}[k].\widehat{\mathsf{join}}(T(\tau_1), T(\tau_2))$.*

The role of g is to allow an arbitrary final renaming of the last bag. In particular, one may take g to be the identity. This lemma asserts that every semi-relabeled refutation can be embedded into an actual tree decomposition. The action-based witness propagation remains structurally valid throughout the decomposition tree. This is a key technical bridge from symbolic refutations to tree-based reasoning.

Proof. We proceed by induction on the length m of the semi-relabeled refutation. Because every pair in R is well-formed and each progression clause specifies the domain of the applied relabeling, every witness action appearing below is well-defined.

Base case ($m = 0$): The sequence is a single state $R = (\mathbf{b}_0, W_0)$. Since every semi-relabeled refutation begins with $(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$, we define:

$$\tau_R := \mathtt{leaf}, \quad T(\mathtt{leaf}) := \mathcal{D}[k].\widehat{\mathsf{leaf}}().$$

Since $\mathbf{b}_0 = \emptyset$, we have $B(\mathtt{leaf}) = \emptyset = g(\mathbf{b}_0)$. Moreover, by Definition 43, item 5,

$$\rho^k_{\mathcal{D}}(g, W_0) = \rho^k_{\mathcal{D}}(g, \mathcal{D}[k].\widehat{\mathsf{leaf}}()) = \mathcal{D}[k].\widehat{\mathsf{leaf}}() = T(\mathtt{leaf}),$$

so condition 1 holds as well. Condition 2 is true by definition of T, and the remaining clauses are vacuous.

Inductive step: Assume the lemma holds for all semi-relabeled refutations of length less than m. Consider a semi-relabeled refutation of length m:

$$R = (\mathbf{b}_0, W_0)(\mathbf{b}_1, W_1) \dots (\mathbf{b}_m, W_m),$$

with relabeling trace $\mathrm{F} = (f_1, \ldots, f_m)$. Let the last transition be from some $i < m$. We distinguish cases based on the type of DP-core operation used.

In Cases 3 and 4, let $f = g \circ f_m$ and apply the inductive hypothesis to obtain a tree decomposition τ_i^f and labeling T_i^f for the prefix ending at state i.

Case 1: $(\mathbf{b}_m, W_m) = (f_m(\mathbf{b}_i \cup \{u\}), \rho_{\mathcal{D}}^k(f_m, \mathcal{D}[k].\widehat{\mathsf{intro}}_u(W_i)))$. Let

$$f_0 := (g \circ f_m)|_{\mathbf{b}_i} \qquad \text{and} \qquad f := g \circ f_m : \mathbf{b}_i \cup \{u\} \to [k+1].$$

Apply the inductive hypothesis to the prefix ending at state i with relabeling f_0, obtaining a tree decomposition $\tau_i^{f_0}$ and labeling $T_i^{f_0}$. Define:

$$\tau_R := \mathtt{intro\_}f(u)(\tau_i^{f_0}), \quad T(\tau) := \begin{cases} T_i^{f_0}(\tau), & \text{if } \tau \in \mathrm{Sub}(\tau_i^{f_0}), \\ \mathcal{D}[k].\widehat{\mathsf{intro}}_{f(u)}(T_i^{f_0}(\tau_i^{f_0})), & \text{if } \tau = \tau_R. \end{cases}$$

Since f_m is injective on the relevant bag and g is injective on $\mathbf{b}_m$, both f_0 and f are valid k-relabelings, and f extends f_0. By the inductive hypothesis and extension invariance,

$$T_i^{f_0}(\tau_i^{f_0}) = \rho_{\mathcal{D}}^k(f_0, W_i) = \rho_{\mathcal{D}}^k(f, W_i).$$

Using compatibility (Definition 43, item 1), extension invariance, and composition (Definition 41, item 3), we get:

$$T(\tau_R) = \rho_{\mathcal{D}}^k(f, \mathcal{D}[k].\widehat{\mathsf{intro}}_u(W_i)) = \rho_{\mathcal{D}}^k(g, W_m),$$

establishing the witness-set part of condition 1. Moreover, by the inductive hypothesis $B(\tau_i^{f_0}) = f_0(\mathbf{b}_i)$, and introduce-vertex updates the top bag by adding $f(u)$, so

$$B(\tau_R) = f_0(\mathbf{b}_i) \cup \{f(u)\} = f(\mathbf{b}_i \cup \{u\}) = g(\mathbf{b}_m).$$

All other conditions follow from the inductive hypothesis.

Case 2: $(\mathbf{b}_m, W_m) = (f_m(\mathbf{b}_i \setminus \{u\}), \rho_{\mathcal{D}}^k(f_m, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(W_i)))$. Let

$$h_0 := g \circ f_m : \mathbf{b}_i \setminus \{u\} \to [k+1].$$

Since h_0 is injective and $|\mathbf{b}_i \setminus \{u\}| < k+1$, choose $x \in [k+1] \setminus h_0(\mathbf{b}_i \setminus \{u\})$, and define the injective extension $\bar{h} : \mathbf{b}_i \to [k+1]$ by

$$\bar{h}|_{\mathbf{b}_i \setminus \{u\}} = h_0, \qquad \bar{h}(u) = x.$$

Apply the inductive hypothesis to the prefix ending at state i with relabeling $\bar{h}$, obtaining $\tau_i^{\bar{h}}$ and $T_i^{\bar{h}}$. Define:

$$\tau_R := \texttt{forget\_}x(\tau_i^{\bar{h}}), \quad T(\tau) := \begin{cases} T_i^{\bar{h}}(\tau), & \text{if } \tau \in \mathrm{Sub}(\tau_i^{\bar{h}}), \\ \mathcal{D}[k].\widehat{\mathsf{forget}}_x(T_i^{\bar{h}}(\tau_i^{\bar{h}})), & \text{if } \tau = \tau_R. \end{cases}$$

By the inductive hypothesis,

$$T_i^{\bar{h}}(\tau_i^{\bar{h}}) = \rho_{\mathcal{D}}^k(\bar{h}, W_i).$$

Using compatibility (Definition 43, item 2), extension invariance (Definition 41, item 5), and composition (Definition 41, item 3), we obtain

$$\begin{aligned} T(\tau_R) &= \rho_{\mathcal{D}}^k(\bar{h}, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(W_i)) \\ &= \rho_{\mathcal{D}}^k(h_0, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(W_i)) \\ &= \rho_{\mathcal{D}}^k\big(g, \rho_{\mathcal{D}}^k(f_m, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(W_i))\big) \\ &= \rho_{\mathcal{D}}^k(g, W_m). \end{aligned}$$

Moreover, by the inductive hypothesis $B(\tau_i^{\bar{h}}) = \bar{h}(\mathbf{b}_i)$, and forget removes x from the top bag. Therefore

$$B(\tau_R) = \bar{h}(\mathbf{b}_i) \setminus \{x\} = \bar{h}(\mathbf{b}_i \setminus \{u\}) = h_0(\mathbf{b}_i \setminus \{u\}) = g(\mathbf{b}_m).$$

All other conditions follow from the inductive hypothesis.

Case 3: $(\mathbf{b}_m, W_m) = (f_m(\mathbf{b}_i), \rho_{\mathcal{D}}^k(f_m, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W_i)))$. Define:

$$\tau_R := \texttt{intro\_}f(u)\texttt{\_}f(v)(\tau_i^f), \quad T(\tau) := \begin{cases} T_i^f(\tau), & \text{if } \tau \in \mathrm{Sub}(\tau_i^f), \\ \mathcal{D}[k].\widehat{\mathsf{intro}}_{f(u),f(v)}(T_i^f(\tau_i^f)), & \text{if } \tau = \tau_R. \end{cases}$$

Using compatibility (Definition 43, item 3):

$$T(\tau_R) = \rho_{\mathcal{D}}^k(f, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W_i)) = \rho_{\mathcal{D}}^k(g, W_m).$$

Moreover, an introduce-edge step does not change the top bag, so $B(\tau_R) = B(\tau_i^f) = f(\mathbf{b}_i) = g(\mathbf{b}_m)$ by the inductive hypothesis.

Case 4: $(\mathbf{b}_m, W_m) = (f_m(\mathbf{b}_i), \rho_{\mathcal{D}}^k(f_m, \mathcal{D}[k].\widehat{\mathsf{join}}(W_i, \rho_{\mathcal{D}}^k(h, W_j))))$, for some $i, j < m$ and some relabeling $h : \mathbf{b}_j \to \mathbf{b}_i$ with $h(\mathbf{b}_j) = \mathbf{b}_i$.

The map h is therefore a bijection between the two predecessor bags, so the ordered join

is well-typed. Define

$$f := g \circ f_m : \mathbf{b}_i \to [k+1] \qquad \text{and} \qquad f' := f \circ h = g \circ f_m \circ h : \mathbf{b}_j \to [k+1].$$

By induction, obtain τ_i^f, $\tau_j^{f'}$, and corresponding $T_i^f, T_j^{f'}$. Define:

$$\tau_R := \mathtt{join}(\tau_i^f, \tau_j^{f'}).$$

Writing $L\sigma$ and $R\sigma$ for the left- and right-tagged occurrences, set

$$T(L\sigma) := T_i^f(\sigma) \quad (\sigma \in \mathrm{Sub}(\tau_i^f)),$$

$$T(R\sigma) := T_j^{f'}(\sigma) \quad (\sigma \in \mathrm{Sub}(\tau_j^{f'})),$$

and

$$T(\tau_R) := \mathcal{D}[k].\widehat{\mathsf{join}}(T_i^f(\tau_i^f), T_j^{f'}(\tau_j^{f'})).$$

Then, by compatibility (Definition 43, item 4) and composition (Definition 41, item 3):

$$T(\tau_R) = \rho_{\mathcal{D}}^k(f, \mathcal{D}[k].\widehat{\mathsf{join}}(W_i, \rho_{\mathcal{D}}^k(h, W_j))) = \rho_{\mathcal{D}}^k(g, W_m).$$

Moreover, by the inductive hypothesis $B(\tau_i^f) = f(\mathbf{b}_i)$ and $B(\tau_j^{f'}) = f'(\mathbf{b}_j) = f(\mathbf{b}_i)$, so the join is well-formed and $B(\tau_R) = f(\mathbf{b}_i) = g(\mathbf{b}_m)$.

Conclusion: In each case, we have constructed τ_R and T satisfying all required conditions, completing the inductive step. □

Corollary 10 (Realized states agree with dynamization)**.** *In the setting of Lemma 15, every subterm occurrence τ of τ_R satisfies*

$$T(\tau) = \Gamma[\mathcal{D}, k](\tau).$$

Proof. We argue by structural induction on the subterm occurrence τ. For the leaf occurrence, clause 2 of Lemma 15 matches the nullary clause of Definition 23. For an internal occurrence, clauses 3–6 state exactly the corresponding lifted DP-core recurrence, so the inductive hypothesis and Definition 23 yield the claim. □

Lemma 15 shows that every semi-relabeled $(k, \mathcal{D})$-refutation can be realized by a concrete k-instructive tree decomposition whose root state is the relabeled final pair. When that final pair is $(k, \mathcal{D})$-inconsistent, Theorem 11 yields that the resulting decomposition is not accepted by $\mathcal{D}[k]$, and hence, by coherence, its realized graph does not belong to $\mathcal{P}[\mathcal{D}, \mathsf{ITD}]$.

This reasoning is formalized in Theorem 11 below, which generalizes Corollary 9 from ordinary instructive refutations to the symmetry-breaking setting.

Theorem 11. *Let $\mathcal{D}$ be an* ITD*-coherent instructive DP-core equipped with a witness action in the sense of Definitions 41–43, and preserving well-formedness, let $k \in \mathbb{N}$, and let $\mathrm{F} = (f_1, \ldots, f_m)$ be a sequence of k-relabeling functions. If there exists an F-relabeled $(k, \mathcal{D})$-refutation, then there exists a k-instructive tree decomposition $\tau \in \mathsf{ITD}_k$ such that:*

$$\mathcal{G}(\tau) \notin \mathcal{P}[\mathcal{D}, \mathsf{ITD}].$$

This result shows that symmetry-aware relabeled refutations are sufficient to prove counterexamples. In other words, relabeling preserves the ability to refute membership in a graph property. The theorem generalizes Corollary 9, which deals with ordinary (non-relabeled) instructive refutations.

The canonization argument below relies on inductive extensions of relabelings. To ensure that fresh labels are always available for introduced vertices, we rely on the following extension property.

Lemma 16 (Fresh-Label Extension)**.** *If $B \subseteq [k+1]$, $|B| < k+1$, $f : B \hookrightarrow [k+1]$ is injective, and $u \in [k+1] \setminus B$, then there exists an injective extension $\bar{f} : B \cup \{u\} \hookrightarrow [k+1]$ such that $\bar{f}|_B = f$ and $\bar{f}(u) \notin f(B)$.*

Proof. Since f is injective, $|f(B)| = |B| < k+1$, so the complement $[k+1] \setminus f(B)$ is nonempty. Choose $x \in [k+1] \setminus f(B)$, define $\bar{f}|_B = f$, and set $\bar{f}(u) = x$. Then $\bar{f}$ is injective, extends f, and satisfies $\bar{f}(u) \notin f(B)$. □

Proof of Theorem 11. Apply Lemma 15 to the underlying semi-relabeled refutation, and take g to be the identity on $\mathbf{b}_m$. This yields a k-instructive decomposition τ whose root bag is $g(\mathbf{b}_m) = \mathbf{b}_m$ and whose root witness set is $\rho^k_{\mathcal{D}}(g, W_m)$. By Corollary 10, this root witness set is exactly

$$\Gamma[\mathcal{D}, k](\tau) = \rho^k_{\mathcal{D}}(g, W_m).$$

Since $(\mathbf{b}_m, W_m)$ is $(k, \mathcal{D})$-inconsistent, we have

$$\forall w \in W_m : \ \mathcal{D}[k].\texttt{Final}(\mathbf{b}_m, w) = 0.$$

Let $w' \in \rho^k_{\mathcal{D}}(g, W_m)$. By Definition 42, there exists $w \in W_m$ with $w' = \rho^k_{\mathcal{D}}(g, w)$. By Finality Invariance (Definition 41, item 1),

$$\mathcal{D}[k].\texttt{Final}(g(\mathbf{b}_m), w') = \mathcal{D}[k].\texttt{Final}(g(\mathbf{b}_m), \rho^k_{\mathcal{D}}(g, w)) = \mathcal{D}[k].\texttt{Final}(\mathbf{b}_m, w) = 0.$$

Hence the root witness set $\rho^k_{\mathcal{D}}(g, W_m)$ contains no final witness relative to the root bag $g(\mathbf{b}_m)$, so $\tau \notin \mathsf{Acc}(\mathcal{D}[k])$. Finally, Proposition 4, specialized to $\mathcal{C} = \mathsf{ITD}$, gives

$$\tau \in \mathsf{Acc}(\mathcal{D}[k]) \iff \mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D}, \mathsf{ITD}].$$

Since $\tau \notin \mathsf{Acc}(\mathcal{D}[k])$, the underlying graph $\mathcal{G}(\tau)$ cannot lie in $\mathcal{P}[\mathcal{D}, \mathsf{ITD}]$. □

6.2 Canonical Refutations

We now define a refined approach to symmetry breaking based on canonical representatives of states. The goal is to retain only one representative for each class of relabelings in a refutation, thereby reducing redundancy without sacrificing completeness. This leads to the notion of a *canonized refutation.*

Let U be an ordered set of elements, and let $X \subseteq U$. We let $\mathrm{vec}(X)$ denote the vector obtained by ordering the elements of X increasingly. For instance, if $X = \{2, 3, 5\}$, then $\mathrm{vec}(X) = (2, 3, 5)$. Given two such subsets $X, X' \subseteq U$, we say that $X < X'$ if $\mathrm{vec}(X)$ is lexicographically smaller than $\mathrm{vec}(X')$.

Let $\mathcal{D}$ be an instructive DP-core and fix an arbitrary total order on the set $\mathcal{D}[k].\mathcal{W}$ (for example, the lexicographic order on encodings of witnesses). For algorithmic applications this order is taken to be computable; equivalently, Theorem 13 assumes a computable canonization procedure directly. We compare pairs $(\mathbf{b}, W)$ and $(\mathbf{b}', W')$ lexicographically by their ordered vector representations:

$$(\mathbf{b}, W) < (\mathbf{b}', W') \iff (\mathrm{vec}(\mathbf{b}), \mathrm{vec}(W)) < (\mathrm{vec}(\mathbf{b}'), \mathrm{vec}(W')).$$

Throughout this section we apply witness actions and canonization only to *well-formed* pairs $(\mathbf{b}, W)$ in which every witness $w \in W$ satisfies $\mathrm{supp}(w) \subseteq \mathbf{b}$. Under Definition 44, the raw DP states produced by the constructions above stay in this well-formed region, and Lemma 13 shows that subsequent relabelings and canonization do so as well.

Definition 48 (Canonical Pair)**.** *Let $(\mathbf{b}, W)$ be a finite well-formed pair; that is, $\mathbf{b} \subseteq [k+1]$, $W \subseteq \mathcal{D}[k].\mathcal{W}$ is finite, and $\mathrm{supp}(W) \subseteq \mathbf{b}$. The canonical form of the pair $(\mathbf{b}, W)$ is defined as:*

$$\mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W) = \min\{(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, W)) \mid f \in \mathcal{R}_k,\ \mathrm{dom}(f) = \mathbf{b}\}.$$

The minimum is taken over the finite nonempty set of injections $\mathbf{b} \hookrightarrow [k+1]$, ordered

by the fixed total order on encoded pairs. We call any relabeling function $f : \mathbf{b} \to [k+1]$ that achieves this minimum a canonical relabeling of $(\mathbf{b}, W)$.

Lemma 17 (Canonization is orbit-invariant and idempotent)**.** *Assume $\mathcal{D}[k]$ is equipped with a witness action. Let $(\mathbf{b}, W)$ be a finite well-formed pair, and let $f \in \mathcal{R}_k$ have domain $\mathbf{b}$. Set*

$$(\mathbf{b}', W') := \big(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, W)\big).$$

Then

$$\mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}', W') = \mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W).$$

Consequently,

$$\mathrm{CAN}^k_{\mathcal{D}}\big(\mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W)\big) = \mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W).$$

Proof. By Lemma 13, $(\mathbf{b}', W')$ is well-formed. The candidates minimized over in Definition 48 for $(\mathbf{b}', W')$ are exactly the candidates for $(\mathbf{b}, W)$. Indeed, if $h : \mathbf{b}' \hookrightarrow [k+1]$, then $h \circ f : \mathbf{b} \hookrightarrow [k+1]$, and by Lemma 9,

$$\big(h(\mathbf{b}'), \rho^k_{\mathcal{D}}(h, W')\big) = \big((h \circ f)(\mathbf{b}), \rho^k_{\mathcal{D}}(h \circ f, W)\big).$$

Conversely, every candidate for $(\mathbf{b}, W)$ arises this way by taking $h = s \circ f^{-1}$ for a relabeling $s : \mathbf{b} \hookrightarrow [k+1]$. Thus the two finite candidate sets are equal, and their minima are equal.

For idempotence, let

$$(\mathbf{b}_c, W_c) = \mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W).$$

By definition, $(\mathbf{b}_c, W_c)$ is obtained from $(\mathbf{b}, W)$ by some canonical relabeling. The orbit-invariance just proved gives

$$\mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}_c, W_c) = \mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W) = (\mathbf{b}_c, W_c).$$

□

Corollary 11. *If $\mathbf{b}$ has size t, then the bag of the canonical form $\mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W)$ is $\{1, \ldots, t\}$. This holds because the lexicographic order on pairs compares bags first.*

Example 3 (Canonization on a two-vertex bag)**.** *Fix $k \geq 4$. Let $\mathbf{b} = \{2, 5\}$ and let $W \subseteq \mathcal{D}[k].\mathcal{W}$ be well-formed with $\mathrm{supp}(w) \subseteq \mathbf{b}$ for all $w \in W$. There are two bijections $f, f' \in \mathcal{R}_k$ from $\mathbf{b}$ to $\{1, 2\}$, namely $f(2) = 1, f(5) = 2$ and $f'(2) = 2, f'(5) = 1$. The two candidates $(f(\mathbf{b}), \rho^k_{\mathcal{D}}(f, W))$ and $(f'(\mathbf{b}), \rho^k_{\mathcal{D}}(f', W))$ only differ by swapping labels 1 and 2 in the relabeled witness set. Canonization picks the lexicographically smaller of the two, so any two states that only differ by swapping the labels 2 and 5 map to the same canonical representative.*

Intuitively, the canonical form $\mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W)$ is the lexicographically smallest pair obtainable by applying a relabeling function to $\mathbf{b}$ and simultaneously relabeling each element of W using a witness action. This operation identifies semantically equivalent states and ensures that only one representative, the canonical one, is used in the refutation process.

Definition 49 (Canonized Refutation)**.** *Let $\mathcal{D}$ be an ITD-coherent instructive DP-core, $k \in \mathbb{N}$, and let $F = (f_1, \ldots, f_m)$ be a sequence of relabeling functions. Suppose*

$$R = (\mathbf{b}_0, W_0)(\mathbf{b}_1, W_1) \ldots (\mathbf{b}_m, W_m)$$

is an F-relabeled $(k, \mathcal{D})$-refutation. Then R is canonized if for every $i \in \{0, \ldots, m\}$, the pair $(\mathbf{b}_i, W_i)$ is equal to its canonical form.

The purpose of canonization is to systematically remove syntactic symmetries in the dynamic programming process. Relabelings allow us to navigate equivalence classes of states efficiently, and canonization ensures that only one representative is used in the refutation. This enables more efficient implementations and avoids redundant work in automated reasoning.

The next theorem guarantees that canonical refutations are as powerful as arbitrary ones: the existence of a canonized refutation is equivalent to the existence of a counterexample graph of bounded treewidth.

Theorem 12. *Let $\mathcal{D}$ be an ITD-coherent instructive DP-core equipped with a witness action in the sense of Definitions 41–43, preserving well-formedness, and let $k \in \mathbb{N}$. Then there exists a canonized $(k, \mathcal{D})$-refutation if and only if there exists a graph of treewidth at most k that does not belong to $\mathcal{P}[\mathcal{D}, \mathsf{ITD}]$.*

Proof. Suppose G is a graph of treewidth at most k such that

$$G \notin \mathcal{P}[\mathcal{D}, \mathsf{ITD}].$$

Then, by Corollary 9, there exists a $(k, \mathcal{D})$-refutation

$$R = (\mathbf{b}_0, W_0)(\mathbf{b}_1, W_1) \ldots (\mathbf{b}_m, W_m).$$

Throughout the construction we use Lemma 9 to apply inversion, composition, and extension invariance to finite witness sets. For each i, let $(\mathbf{b}'_i, W'_i) = \mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}_i, W_i)$, and fix a relabeling function $f_i \in \mathcal{R}_k$ with $\mathrm{dom}(f_i) = \mathbf{b}_i$ witnessing the minimum, i.e.,

$$(\mathbf{b}'_i, W'_i) = (f_i(\mathbf{b}_i), \rho^k_{\mathcal{D}}(f_i, W_i)).$$

Since each $(\mathbf{b}_i, W_i)$ is well-formed and $\mathrm{dom}(f_i) = \mathbf{b}_i$, Lemma 13 implies that every $(\mathbf{b}'_i, W'_i)$ is well-formed. We claim that $R' = (\mathbf{b}'_0, W'_0)\dots(\mathbf{b}'_m, W'_m)$ is an F-relabeled refutation for a suitable trace $F = (g_1, \dots, g_m)$. Initialization holds because $(\mathbf{b}_0, W_0) = (\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$, and the empty bag has only the identity relabeling, so $(\mathbf{b}'_0, W'_0) = (\mathbf{b}_0, W_0)$.

For progression, fix $i \in [m]$ and consider the clause of Definition 46 used by the original refutation at step i.

Introduce-vertex case. Suppose

$$(\mathbf{b}_i, W_i) = \big(\mathbf{b}_j \cup \{v\}, \mathcal{D}[k].\widehat{\mathsf{intro}}_v(W_j)\big)$$

for some $j < i$ and $v \notin \mathbf{b}_j$. Extend f_j to an injective $\bar{f}_j : \mathbf{b}_j \cup \{v\} \to [k+1]$ by Lemma 16, and set

$$x_i := \bar{f}_j(v), \qquad g_i := f_i \circ \bar{f}_j^{-1} : \bar{f}_j(\mathbf{b}_j \cup \{v\}) \to \mathbf{b}'_i.$$

Because $\bar{f}_j$ extends f_j, Extension Invariance gives $\rho^k_{\mathcal{D}}(\bar{f}_j, W_j) = W'_j$. Moreover,

$$g_i(\mathbf{b}'_j \cup \{x_i\}) = g_i(\bar{f}_j(\mathbf{b}_j \cup \{v\})) = f_i(\mathbf{b}_j \cup \{v\}) = \mathbf{b}'_i.$$

Using compatibility for introduce-vertex, Extension Invariance, and Composition,

$$\begin{aligned}
\rho^k_{\mathcal{D}}\big(g_i, \mathcal{D}[k].\widehat{\mathsf{intro}}_{x_i}(W'_j)\big) &= \rho^k_{\mathcal{D}}\big(f_i \circ \bar{f}_j^{-1}, \mathcal{D}[k].\widehat{\mathsf{intro}}_{\bar{f}_j(v)}(\rho^k_{\mathcal{D}}(\bar{f}_j, W_j))\big) \\
&= \rho^k_{\mathcal{D}}\big(f_i \circ \bar{f}_j^{-1}, \rho^k_{\mathcal{D}}(\bar{f}_j, \mathcal{D}[k].\widehat{\mathsf{intro}}_v(W_j))\big) \\
&= \rho^k_{\mathcal{D}}\big(f_i, \mathcal{D}[k].\widehat{\mathsf{intro}}_v(W_j)\big) \\
&= \rho^k_{\mathcal{D}}(f_i, W_i) = W'_i.
\end{aligned}$$

Thus the relabeled progression clause for introduce-vertex holds.

Forget-vertex case. Suppose

$$(\mathbf{b}_i, W_i) = \big(\mathbf{b}_j \setminus \{v\}, \mathcal{D}[k].\widehat{\mathsf{forget}}_v(W_j)\big)$$

for some $j < i$ and $v \in \mathbf{b}_j$. Set $x_i := f_j(v)$ and

$$g_i := f_i \circ (f_j|_{\mathbf{b}_j \setminus \{v\}})^{-1} : \mathbf{b}'_j \setminus \{x_i\} \to \mathbf{b}'_i.$$

Then

$$g_i(\mathbf{b}'_j \setminus \{x_i\}) = f_i(\mathbf{b}_j \setminus \{v\}) = \mathbf{b}'_i.$$

Since the forget transition lands on the smaller bag $\mathbf{b}_j \setminus \{v\}$, Extension Invariance allows

us to replace f_j by its restriction $f_j|_{\mathbf{b}_j\setminus\{v\}}$ after forgetting v. Because

$$g_i \circ (f_j|_{\mathbf{b}_j\setminus\{v\}}) = f_i,$$

we may then apply Composition on that restricted domain. Thus, using compatibility for forget-vertex, Extension Invariance, and Composition,

$$\begin{aligned}
\rho^k_{\mathcal{D}}\big(g_i, \mathcal{D}[k].\widehat{\mathsf{forget}}_{x_i}(W'_j)\big) &= \rho^k_{\mathcal{D}}\big(g_i, \mathcal{D}[k].\widehat{\mathsf{forget}}_{f_j(v)}(\rho^k_{\mathcal{D}}(f_j, W_j))\big)\\
&= \rho^k_{\mathcal{D}}\big(g_i, \rho^k_{\mathcal{D}}(f_j|_{\mathbf{b}_j\setminus\{v\}}, \mathcal{D}[k].\widehat{\mathsf{forget}}_{v}(W_j))\big)\\
&= \rho^k_{\mathcal{D}}\big(f_i, \mathcal{D}[k].\widehat{\mathsf{forget}}_{v}(W_j)\big)\\
&= \rho^k_{\mathcal{D}}(f_i, W_i) = W'_i.
\end{aligned}$$

Introduce-edge case. Suppose

$$(\mathbf{b}_i, W_i) = \big(\mathbf{b}_j, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W_j)\big)$$

for some $j < i$ and distinct $u, v \in \mathbf{b}_j$. Set

$$g_i := f_i \circ f_j^{-1} : \mathbf{b}'_j \to \mathbf{b}'_i.$$

Then $g_i(\mathbf{b}'_j) = f_i(\mathbf{b}_j) = \mathbf{b}'_i$. Since f_j is injective, $f_j(u) \neq f_j(v)$, and compatibility for introduce-edge gives

$$\begin{aligned}
\rho^k_{\mathcal{D}}\big(g_i, \mathcal{D}[k].\widehat{\mathsf{intro}}_{f_j(u),f_j(v)}(W'_j)\big) &= \rho^k_{\mathcal{D}}\big(g_i, \mathcal{D}[k].\widehat{\mathsf{intro}}_{f_j(u),f_j(v)}(\rho^k_{\mathcal{D}}(f_j, W_j))\big)\\
&= \rho^k_{\mathcal{D}}\big(g_i, \rho^k_{\mathcal{D}}(f_j, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W_j))\big)\\
&= \rho^k_{\mathcal{D}}\big(f_i, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(W_j)\big)\\
&= \rho^k_{\mathcal{D}}(f_i, W_i) = W'_i.
\end{aligned}$$

Join case. Suppose

$$(\mathbf{b}_i, W_i) = \big(\mathbf{b}_j, \mathcal{D}[k].\widehat{\mathsf{join}}(W_j, W_l)\big)$$

for some $j, l < i$ with $\mathbf{b}_j = \mathbf{b}_l$. By Corollary 11, the canonical bags satisfy $\mathbf{b}'_j = \mathbf{b}'_l$. Define

$$\pi := f_j \circ f_l^{-1} : \mathbf{b}'_l \to \mathbf{b}'_j, \qquad g_i := f_i \circ f_j^{-1} : \mathbf{b}'_j \to \mathbf{b}'_i.$$

Then $g_i(\mathbf{b}'_j) = f_i(\mathbf{b}_j) = \mathbf{b}'_i$, and π is a bijection between the canonical predecessor bags. Using Composition,

$$\rho^k_{\mathcal{D}}(\pi, W'_l) \;=\; \rho^k_{\mathcal{D}}(f_j, W_l),$$

because $W'_l = \rho^k_{\mathcal{D}}(f_l, W_l)$, $\pi = f_j \circ f_l^{-1}$, and Composition together with Inversion reduces

$\rho^k_{\mathcal{D}}(f_j \circ f_l^{-1}, W'_l)$ to $\rho^k_{\mathcal{D}}(f_j, W_l)$. Hence, by compatibility for join and another application of Composition,

$$\begin{aligned}\rho^k_{\mathcal{D}}\big(g_i, \mathcal{D}[k].\widehat{\mathsf{join}}(W'_j, \rho^k_{\mathcal{D}}(\pi, W'_l))\big) &= \rho^k_{\mathcal{D}}\big(g_i, \mathcal{D}[k].\widehat{\mathsf{join}}(\rho^k_{\mathcal{D}}(f_j, W_j), \rho^k_{\mathcal{D}}(f_j, W_l))\big)\\ &= \rho^k_{\mathcal{D}}\big(g_i, \rho^k_{\mathcal{D}}(f_j, \mathcal{D}[k].\widehat{\mathsf{join}}(W_j, W_l))\big)\\ &= \rho^k_{\mathcal{D}}\big(f_i, \mathcal{D}[k].\widehat{\mathsf{join}}(W_j, W_l)\big)\\ &= \rho^k_{\mathcal{D}}(f_i, W_i) = W'_i.\end{aligned}$$

Thus the relabeled progression clause for join holds exactly in the form required by Definition 47.

Finally, $(\mathbf{b}_m, W_m)$ is $(k, \mathcal{D})$-inconsistent, so

$$\forall w \in W_m : \ \mathcal{D}[k].\mathtt{Final}(\mathbf{b}_m, w) = 0.$$

Let $w' \in W'_m$. By the definition of W'_m, there exists $w \in W_m$ such that $w' = \rho^k_{\mathcal{D}}(f_m, w)$. Since $\mathbf{b}'_m = f_m(\mathbf{b}_m)$, Finality Invariance gives

$$\mathcal{D}[k].\mathtt{Final}(\mathbf{b}'_m, w') = \mathcal{D}[k].\mathtt{Final}(f_m(\mathbf{b}_m), \rho^k_{\mathcal{D}}(f_m, w)) = \mathcal{D}[k].\mathtt{Final}(\mathbf{b}_m, w) = 0.$$

Thus $(\mathbf{b}'_m, W'_m)$ is $(k, \mathcal{D})$-inconsistent. By Lemma 17, every pair $(\mathbf{b}'_i, W'_i)$ is fixed by canonization. Hence R' is a canonized refutation.

Conversely, suppose a canonized $(k, \mathcal{D})$-refutation R exists. Then R is by definition an F-relabeled refutation for some relabeling sequence F. By Theorem 11, this implies the existence of a k-instructive tree decomposition τ such that

$$\mathcal{G}(\tau) \notin \mathcal{P}[\mathcal{D}, \mathsf{ITD}].$$

By Lemma 2, the realized graph $\mathcal{G}(\tau)$ has treewidth at most k. □

This theorem confirms that symmetry breaking via canonization does not compromise the deductive power of the refutation system. Moreover, it enables practical reductions in complexity while preserving logical soundness and completeness.

To algorithmically determine whether a graph property defined by a coherent instructive DP-core includes all graphs of treewidth at most k, we employ a dynamic programming procedure that simulates the construction of a canonized refutation. The procedure explores states of the form $(\mathbf{b}, W)$, representing partial decompositions and their associated sets of witnesses, and applies all possible DP-core operations at each step. These include vertex introductions, vertex forgets, edge introductions, and joins. Crucially, to avoid

redundant computation and ensure termination, each generated state is immediately transformed into its canonical form via the canonization function $\mathrm{CAN}_{\mathcal{D}}$, and only previously unseen canonical states are further explored.

The algorithm maintains a working set of unexplored states and a history of visited canonical pairs. It terminates either when an inconsistent state is found, yielding a refutation, or when all reachable states have been explored without contradiction. This approach guarantees completeness via Theorem 12. Since Definition 47 treats join as an ordered operation, the join loop below explicitly enumerates both operand orientations rather than assuming any commutativity of $\widehat{\mathsf{join}}$. The algorithm is formally presented in Algorithm 2.

As in Chapter 4, we work with explicit encodings of bags, witnesses, and finite witness sets, so equality of states means equality of their finite encodings. The algorithmic theorem below is therefore stated with explicit executability hypotheses. Internal polynomiality, together with Lemma 8, makes the lifted DP-core operations and finality test computable on explicit witness-set encodings. The symmetry-specific extra operations are the witness action and a canonization procedure which returns both the canonical pair and one relabeling attaining the minimum in Definition 48; this relabeling is stored in insertion certificates. The theorem's bound counts only the number of canonical states enumerated; a concrete running-time estimate must additionally account for the costs of the lifted DP-core operations, witness actions, state comparison, and canonization.

In Algorithm 2, enqueue(X, Γ) abbreviates the following operation: canonize the raw successor X, obtaining both $\bar{X}$ and a canonical relabeling; if $\bar{X} \notin \hat{R}$, append $\bar{X}$ to Q, insert it into $\hat{R}$, and store cert$(\bar{X})$ recording the raw predecessor data Γ and that canonical relabeling.

In the algorithm below, the local symbols R, $\hat{R}$, and Q play the same roles as R, $\hat{R}$, and the worklist Y of the bare inclusion test in Chapter 5 (Algorithm 1); we keep the lighter notation here because canonization, certificate bookkeeping, and the join branch already make the algorithm visually heavy.

Input : A width bound $k \in \mathbb{N}$, and a finite, internally polynomial, ITD-coherent instructive DP-core $\mathcal{D}$ equipped with a computable witness action, preserving well-formedness, and equipped with a computable canonization procedure returning $\mathrm{CAN}^k_{\mathcal{D}}$ together with a canonical relabeling.

Output : A list R of canonical states together with insertion certificates cert witnessing a canonized $(k,\mathcal{D})$-refutation if some graph of treewidth at most k does not belong to $\mathcal{P}[\mathcal{D},\mathsf{ITD}]$, and "Inclusion Holds" otherwise.

$R \leftarrow [\,]$; `/* [] is the empty list. */`
$\hat{R} \leftarrow \{\,\}$; `/* { } is the empty set. */`
$X_0 \leftarrow \mathrm{CAN}^k_{\mathcal{D}}(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$;
$Q \leftarrow [X_0]$; `/* queue of unexplored canonical states */`
$\hat{R} \leftarrow \hat{R} \cup \{X_0\}$;
Store the initialization certificate $\mathrm{cert}(X_0)$;
while $Q \neq \emptyset$ **do**
  Remove the first element $(\mathbf{b}, \mathbf{W})$ from Q and append it to R;
  if $(\mathbf{b}, \mathbf{W})$ *is an inconsistent* $(k,\mathcal{D})$*-pair,* **then**
    return (R, cert) ; `/* In this case, R together with the stored certificates witnesses a canonized (k,D)-refutation. */`
  else
    foreach $u \in [k+1] \setminus \mathbf{b}$ **do**
      $X \leftarrow (\mathbf{b} \cup \{u\}, \mathcal{D}[k].\widehat{\mathsf{intro}}_u(\mathbf{W}))$;
      enqueue$(X, \mathrm{intro}(u, \mathbf{b}, \mathbf{W}))$;
    end
    foreach $u \in \mathbf{b}$ **do**
      $X \leftarrow (\mathbf{b} \setminus \{u\}, \mathcal{D}[k].\widehat{\mathsf{forget}}_u(\mathbf{W}))$;
      enqueue$(X, \mathrm{forget}(u, \mathbf{b}, \mathbf{W}))$;
    end
    foreach $u, v \in \mathbf{b}$ *with* $u \neq v$ **do**
      $X \leftarrow (\mathbf{b}, \mathcal{D}[k].\widehat{\mathsf{intro}}_{u,v}(\mathbf{W}))$;
      enqueue$(X, \mathrm{edge}(u, v, \mathbf{b}, \mathbf{W}))$;
    end
    foreach $(\mathbf{b}', \mathbf{W}') \in R$ *with* $\mathbf{b} = \mathbf{b}'$ **do**
      foreach $f \in \mathcal{R}_k$ *with* $\mathrm{dom}(f) = \mathbf{b}$ *and* $f(\mathbf{b}) = \mathbf{b}$ **do**
        $X \leftarrow (\mathbf{b}, \mathcal{D}[k].\widehat{\mathsf{join}}(\mathbf{W}, \rho^k_{\mathcal{D}}(f, \mathbf{W}')))$;
        enqueue$(X, \mathrm{join}_1(\mathbf{b}, \mathbf{W}, \mathbf{W}', f))$;
        $X \leftarrow (\mathbf{b}, \mathcal{D}[k].\widehat{\mathsf{join}}(\mathbf{W}', \rho^k_{\mathcal{D}}(f, \mathbf{W})))$;
        enqueue$(X, \mathrm{join}_2(\mathbf{b}, \mathbf{W}, \mathbf{W}', f))$;
      end
    end
  end
end
return "Inclusion Holds."

Algorithm 2: Canonization-Aware Inclusion Test

Example 4 (Join and canonization: alignment by permutation)**.** *Fix $k \geq 6$ and a bag $\mathbf{b} = \{2,5,7\}$. Consider a toy DP-core D^{edge} whose witness at bag $\mathbf{b}$ is the set of currently present boundary edges,*

$$W \subseteq \binom{\mathbf{b}}{2}.$$

Its witness action relabels endpoints:

$$\rho^k_{D^{\mathrm{edge}}}(f, W) = \{\{f(u), f(v)\} : \{u,v\} \in W\},$$

and lifted join is set union,

$$D^{\mathrm{edge}}[k].\widehat{\mathsf{join}}(W, W') = W \cup W'.$$

Let

$$X_1 = (\mathbf{b}, \{\{2,5\}\}), \qquad X_2 = (\mathbf{b}, \{\{5,7\}\}).$$

Before canonization, joining gives

$$(\mathbf{b}, \{\{2,5\},\{5,7\}\}),$$

that is, a boundary path.

Now canonize operands separately. Both map to the bag $\{1,2,3\}$ *and can share the same canonical representative:*

$$\mathrm{CAN}^k_{D^{\mathrm{edge}}}(X_1) = \mathrm{CAN}^k_{D^{\mathrm{edge}}}(X_2) = (\{1,2,3\}, \{\{1,2\}\}).$$

However, the canonical relabelings may differ. For instance,

$$c_1 : 2 \mapsto 1,\ 5 \mapsto 2,\ 7 \mapsto 3, \qquad c_2 : 5 \mapsto 1,\ 7 \mapsto 2,\ 2 \mapsto 3.$$

Their relative alignment is the permutation $\pi = c_1 \circ c_2^{-1}$ *of* $\{1,2,3\}$*, i.e.* $\pi(1) = 2,\ \pi(2) = 3,\ \pi(3) = 1$*. Algorithm 2 enumerates such permutations at join time and computes*

$$D^{\mathrm{edge}}[k].\widehat{\mathsf{join}}\left(\{\{1,2\}\}, \rho^k_{D^{\mathrm{edge}}}(\pi, \{\{1,2\}\})\right) = \{\{1,2\}\} \cup \{\{2,3\}\},$$

which canonizes to the expected boundary path state. This illustrates why the join loop applies relabelings before canonization: canonization merges canonical representatives, but join still depends on the relative identification between operand boundaries.

Theorem 13 (Correctness of the Relabeled Inclusion Test)**.** *Let* $\mathcal{D}$ *be a finite, internally polynomial,* ITD*-coherent instructive DP-core equipped with a computable witness action, preserving well-formedness, and equipped, for the fixed width* $k \in \mathbb{N}$*, with a computable canonization procedure which, on every finite well-formed pair, returns both* $\mathrm{CAN}^k_{\mathcal{D}}(\mathbf{b}, W)$ *and one relabeling attaining the minimum in Definition 48. Write*

$$\beta_k := \beta_{\mathcal{D}}(k) \qquad \textit{and} \qquad \mu_k := \mu_{\mathcal{D}}(k).$$

Then Algorithm 2 terminates after enumerating at most

$$(k+2)\sum_{i=0}^{\mu_k} \binom{2^{\beta_k+1}-1}{i} \ \le\ (k+2)\cdot(2^{\beta_k})^{O(\mu_k)}$$

distinct canonical states.

Moreover, Algorithm 2 returns before exhausting its queue if and only if there exists a

graph of treewidth at most k that does not belong to $\mathcal{P}[\mathcal{D}, \mathsf{ITD}]$. In that case, the returned data (R, cert) witness a canonized $(k, \mathcal{D})$-refutation.

Proof. Executability is immediate from the hypotheses: internal polynomiality and Lemma 8 make the lifted DP-core operations and the finality test computable on explicit encodings, while the witness action and canonization procedure are computable by assumption. Thus Algorithm 2 is an effective finite-state search procedure. If one starts only from a computable canonical-pair operator, the attaining relabeling can be recovered by enumerating the finite set of injections $\mathbf{b} \hookrightarrow [k+1]$, applying the witness action to each candidate, and choosing one injection whose relabeled pair equals the canonical pair.

Let Q be the queue of unexplored states and $\hat{R}$ the set of already discovered canonical states maintained by Algorithm 2.

The algorithm stores, for each state X when it is first inserted into $\hat{R}$, one *certificate of insertion* $\mathrm{cert}(X)$:

- $\mathrm{cert}(X_0)$ is the trivial initialization certificate for $X_0 = \mathrm{CAN}^k_{\mathcal{D}}(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$.
- If $X \neq X_0$ is inserted while processing the current state $(\mathbf{b}, W)$, then $\mathrm{cert}(X)$ records one raw successor template used by the algorithm, the predecessor state or states from the current list R, and one relabeling witnessing the canonization $X = \mathrm{CAN}^k_{\mathcal{D}}(X_{\mathrm{raw}})$.

Because the algorithm appends the current state to R before generating its successors, every predecessor mentioned in $\mathrm{cert}(X)$ already belongs to R at the moment X is inserted.

We establish the following invariant.

Invariant. Every state in $Q \cup \hat{R}$ is canonical and well-formed. Every state in $Q \cup \hat{R}$ has a fixed certificate of insertion. If such a state has already been processed, then its certificate refers only to earlier entries of the current list R; if it is still in Q, then its certificate refers only to states that were already processed when it was inserted, and those states will precede it when the queued state is later appended to R. The only exception is X_0, whose certificate is the initialization certificate. Moreover, either R is empty, or the list R of states already removed from Q, read together with the certificates of its entries, is a canonized semi-relabeled $(k, \mathcal{D})$-refutation.

Proof of the invariant. Initially,

$$X_0 = \mathrm{CAN}^k_{\mathcal{D}}(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}())$$

is canonical by Lemma 17, and it is well-formed by Definition 44(1) together with Lemma 13. Moreover, $\mathrm{cert}(X_0)$ is fixed as above. Since R is empty at initialization, the final clause of the invariant holds by its explicit "R is empty" alternative.

Assume now that the invariant holds just before a loop iteration, and let $(\mathbf{b}, W)$ be the state removed from Q and appended to R. If $(\mathbf{b}, W) = X_0$, then $\mathrm{cert}(X_0)$ makes the one-state list (X_0) a canonized semi-relabeled $(k, \mathcal{D})$-refutation. Otherwise $\mathrm{cert}((\mathbf{b}, W))$ witnesses that this state is obtained from earlier states of R by one progression step of Definition 47. Hence appending the current state preserves the claim that R is a canonized semi-relabeled refutation.

Consider any new state $\bar{X}$ generated while processing $(\mathbf{b}, W)$. By construction, $\bar{X} = \mathrm{CAN}^k_{\mathcal{D}}(X)$ for some raw successor X obtained from $(\mathbf{b}, W)$ by one of the four successor templates used in the algorithm. Use the relabeling returned by the canonization procedure for $\mathrm{CAN}^k_{\mathcal{D}}(X)$, and let $\mathrm{cert}(\bar{X})$ be the corresponding certificate of insertion.

If X arises from an introduce-vertex, forget-vertex, or introduce-edge step, then the chosen certificate records exactly one progression step from Definition 47 from the current state $(\mathbf{b}, W)$ to $\bar{X}$.

If X arises from the join loop, then X has one of the two forms

$$X = \big(\mathbf{b}, \mathcal{D}[k].\widehat{\mathsf{join}}(W, \rho^k_{\mathcal{D}}(f, W'))\big)$$

or

$$X = \big(\mathbf{b}, \mathcal{D}[k].\widehat{\mathsf{join}}(W', \rho^k_{\mathcal{D}}(f, W))\big)$$

for some state $(\mathbf{b}, W') \in R$ and some relabeling $f \in \mathcal{R}_k$ with $\mathrm{dom}(f) = \mathbf{b}$ and $f(\mathbf{b}) = \mathbf{b}$. Either way this is exactly a valid ordered join step of Definition 47, again followed by canonization, so $\mathrm{cert}(\bar{X})$ has predecessors already in R.

Because the current state $(\mathbf{b}, W)$ is well-formed and $\mathcal{D}$ preserves well-formedness, every raw successor X produced by one of the four templates is well-formed. Applying canonization to such an X is therefore legitimate, and Lemma 13 shows that the resulting state $\bar{X} = \mathrm{CAN}^k_{\mathcal{D}}(X)$ is again well-formed. By Lemma 17, $\bar{X}$ is canonical.

Therefore the invariant is preserved throughout the run. ◇

We now prove termination and the state bound. Let X be any state that ever appears in $Q \cup \hat{R}$. If $X \in R$, then the prefix of the current list R ending at X is a canonized semi-relabeled $(k, \mathcal{D})$-refutation with final state X. If $X \in (Q \cup \hat{R}) \setminus R$, then its certificate of insertion names only predecessor states that were already processed when X was inserted and hence already occur in the current R; appending X to the current list R therefore yields a canonized semi-relabeled $(k, \mathcal{D})$-refutation whose final state is X. Hence every state that ever appears in $Q \cup \hat{R}$ is the final state of such a refutation. Applying Lemma 15 with the identity relabeling on the final bag, and then Corollary 10, each such state occurs as the root state of some k-instructive decomposition τ. Hence its witness set is $(\mathcal{D}, k, |\tau|)$-useful in the sense of Chapter 4.

By definition of the width-only envelopes $\beta_k = \beta_{\mathcal{D}}(k)$ and $\mu_k = \mu_{\mathcal{D}}(k)$, every witness appearing in such a state has encoding length at most β_k, and every witness set has size at most μ_k. Therefore the number of possible witness encodings is at most $2^{\beta_k+1} - 1$, and the number of possible witness sets is at most

$$\sum_{i=0}^{\mu_k} \binom{2^{\beta_k+1} - 1}{i} \leq (2^{\beta_k})^{O(\mu_k)}.$$

By Corollary 11, the bag of a canonical state is completely determined by its size, so there are only $k + 2$ possible canonical bags, namely $\emptyset, \{1\}, \ldots, \{1, \ldots, k + 1\}$. Hence the total number of distinct canonical states is at most

$$(k + 2) \sum_{i=0}^{\mu_k} \binom{2^{\beta_k+1} - 1}{i} \leq (k + 2) \cdot (2^{\beta_k})^{O(\mu_k)}.$$

Because the algorithm inserts a state into $\hat{R}$ only if it has not been seen before, it can perform only finitely many insertions. Therefore it terminates.

It remains to prove correctness.

Soundness. Suppose the algorithm returns before exhausting Q. Then it returns on a state $(\mathbf{b}, W)$ that is $(k, \mathcal{D})$-inconsistent. At the moment of return, this state has just been appended to R. By the invariant, the returned data (R, cert) are such that the list R, together with the stored certificates of its entries, is a canonized semi-relabeled $(k, \mathcal{D})$-refutation. Reading off, for each non-initial entry of R, the relabeling stored in its certificate of insertion produces a relabeling trace $F = (f_1, \ldots, f_t)$; the predecessor data stored in those same certificates witnesses exactly the progression clauses of Definition 47. Hence R is a canonized F-relabeled semi-refutation. Since its final state is $(k, \mathcal{D})$-inconsistent, R is in fact a canonized F-relabeled $(k, \mathcal{D})$-refutation, i.e. a canonized

$(k, \mathcal{D})$-refutation. By Theorem 11, there is a k-instructive decomposition τ such that

$$\mathcal{G}(\tau) \notin \mathcal{P}[\mathcal{D}, \mathsf{ITD}].$$

By Lemma 2, the realized graph $\mathcal{G}(\tau)$ has treewidth at most k.

Completeness. Conversely, suppose there exists a graph of treewidth at most k that does not belong to $\mathcal{P}[\mathcal{D}, \mathsf{ITD}]$. By Theorem 12, there exists a canonized $(k, \mathcal{D})$-refutation

$$R^* = (\mathbf{b}_0, W_0)(\mathbf{b}_1, W_1) \dots (\mathbf{b}_m, W_m).$$

We show by induction on i that either the algorithm has already returned, or the state $(\mathbf{b}_i, W_i)$ is eventually inserted into $\hat{R}$ and, if no earlier return occurs, is eventually removed from Q and appended to R.

For $i = 0$, we have

$$(\mathbf{b}_0, W_0) = \mathrm{CAN}^k_{\mathcal{D}}(\emptyset, \mathcal{D}[k].\widehat{\mathsf{leaf}}()),$$

so this is exactly the initial state X_0 inserted by the algorithm unless the algorithm has already returned.

Assume now that $i > 0$. If the algorithm has already returned, there is nothing to prove. Otherwise, all earlier predecessor states of R^* used in the i-th step are eventually processed by the induction hypothesis. Consider the step producing $(\mathbf{b}_i, W_i)$.

If it is an introduce-vertex, forget-vertex, or introduce-edge step from some earlier state $(\mathbf{b}_j, W_j)$, then when the algorithm processes $(\mathbf{b}_j, W_j)$ it explicitly generates the corresponding raw DP-core successor X before canonization. Since $(\mathbf{b}_i, W_i)$ is canonical and is a relabeled version of this raw successor by Definition 47, Lemma 17 implies

$$\mathrm{CAN}^k_{\mathcal{D}}(X) = (\mathbf{b}_i, W_i).$$

Hence the algorithm inserts $(\mathbf{b}_i, W_i)$ unless it has already been seen.

If it is a join step using earlier states $(\mathbf{b}_j, W_j)$ and $(\mathbf{b}_\ell, W_\ell)$, then both predecessors are canonical. By Corollary 11, equal bag size implies $\mathbf{b}_j = \mathbf{b}_\ell$. Let $h : \mathbf{b}_\ell \to \mathbf{b}_j$ be the alignment relabeling used in Definition 47. Since $\mathbf{b}_j = \mathbf{b}_\ell$ and $h(\mathbf{b}_\ell) = \mathbf{b}_j$, the map h is a permutation of the common canonical bag. When the later of the two predecessor states is processed by the algorithm, the other one is already in R, and the join loop iterates over every relabeling $f \in \mathcal{R}_k$ with $\mathrm{dom}(f) = \mathbf{b}_j$ and $f(\mathbf{b}_j) = \mathbf{b}_j$, in particular over $f = h$. Moreover, the algorithm now enumerates both ordered join orientations. If

the later processed state is $(\mathbf{b}_j, W_j)$, then the first join template generates

$$X = \big(\mathbf{b}_j, \mathcal{D}[k].\widehat{\mathsf{join}}(W_j, \rho^k_{\mathcal{D}}(h, W_\ell))\big).$$

If the later processed state is $(\mathbf{b}_\ell, W_\ell)$, then the second join template generates the same ordered raw successor

$$X = \big(\mathbf{b}_j, \mathcal{D}[k].\widehat{\mathsf{join}}(W_j, \rho^k_{\mathcal{D}}(h, W_\ell))\big),$$

now with $(\mathbf{b}_j, W_j)$ taken from R and $(\mathbf{b}_\ell, W_\ell)$ as the currently processed state. Again $(\mathbf{b}_i, W_i)$ is a canonical relabeling of this raw successor, so Lemma 17 gives

$$\mathrm{CAN}^k_{\mathcal{D}}(X) = (\mathbf{b}_i, W_i).$$

Hence $(\mathbf{b}_i, W_i)$ is inserted.

If the algorithm returns before the final state of R^* is processed, then completeness is already established. Otherwise, by induction the final inconsistent state $(\mathbf{b}_m, W_m)$ is eventually inserted and processed. The algorithm then detects inconsistency and returns.

This proves both correctness directions and completes the proof. □

6.3 Summary

Symmetry breaking trims the search space explored by treewidth-based ATP without compromising correctness. Canonical representatives, together with witness actions, allow refutations to be stored in a reduced form and significantly accelerate the inclusion tests described in Chapter 5.

Chapter 7

Premise Search

In this chapter, we develop premise-based pruning techniques and incorporate them into our framework [de Oliveira Oliveira and Vadiee, 2024]. Whereas Chapter 6 focused on refutations for single DP-cores, we now study implication properties of the form $\mathcal{P}_1 \to \mathcal{P}_2$ and show how subgraph-closed premises enable aggressive pruning during breadth-first exploration.

7.1 Premise Search and Subgraph-Closed Properties

Throughout this chapter, graph properties are understood up to isomorphism and are properties of the finite loopless multigraphs realized by instructive terms, unless explicitly stated as properties of the underlying simple graph. Let $\mathcal{P}$ be such a graph property. We say that $\mathcal{P}$ is *subgraph-closed* if it satisfies the following condition.

Definition 50 (Subgraph-Closed Graph Property)**.** *A graph property $\mathcal{P}$ is subgraph-closed if for every graph $G \in \mathcal{P}$ and every (not necessarily induced) subgraph $H \subseteq G$, it holds that $H \in \mathcal{P}$.*

Examples. The property of being 3-colorable is subgraph-closed: any subgraph of a 3-colorable graph can also be colored with at most three colors, since removing vertices or edges cannot introduce new coloring conflicts. Likewise, the properties of being bipartite, triangle-free, and simple (having no parallel edges) are all subgraph-closed. By contrast, properties such as "containing a triangle" are not subgraph-closed, since deleting edges can destroy a witness.

Subgraph-closed premises are important in logical implications of the form $\mathcal{P}_1 \to \mathcal{P}_2$.

When $\mathcal{P}_1$ is subgraph-closed, we can exploit this structure to avoid exploring supergraphs of any graph that already violates the premise — a fact that becomes crucial when refutations are explored breadth-first.

The pruning rule of this chapter is sound only when the premise is subgraph-closed. For example, triangle-freeness is safe, while a premise such as "contains a triangle" is not: a partial term may fail to satisfy that property even though some extension later creates a triangle. Premise checks are therefore worth using only when their evaluation cost is outweighed by the pruning they unlock.

In our framework, graph properties are often expressed via coherent DP-cores. Suppose we are interested in verifying whether all graphs of treewidth at most k satisfy a property of the form

$$\mathcal{P}[\mathcal{D}_1, \mathsf{ITD}] \to \mathcal{P}[\mathcal{D}_2, \mathsf{ITD}],$$

where $\mathcal{D}_1$ and $\mathcal{D}_2$ are ITD-coherent DP-cores and $\mathcal{P}[\mathcal{D}_1, \mathsf{ITD}]$ is subgraph-closed. If we find a graph G of treewidth $\leq k$ such that $G \in \mathcal{P}[\mathcal{D}_1, \mathsf{ITD}]$ but $G \notin \mathcal{P}[\mathcal{D}_2, \mathsf{ITD}]$, then this graph serves as a counterexample to the implication.

However, even when such a graph is not yet found, the subgraph-closedness of $\mathcal{P}[\mathcal{D}_1, \mathsf{ITD}]$ lets us prune the search space:

Lemma 18 (Subterms realize subgraphs)**.** *Let $\rho \in \mathsf{ITD}_k$, and let σ be a subterm occurrence of ρ. Under the root-open ITD convention, σ is itself an element of ITD_k. Then $\mathcal{G}(\sigma)$ is isomorphic to a subgraph of $\mathcal{G}(\rho)$.*

Proof. The root-open convention from Section 3.4 makes every locally valid subterm occurrence an accepted instructive term with its own root bag, so $\mathcal{G}(\sigma)$ is defined. It is enough to treat the case where the selected occurrence σ is an immediate child of a parent occurrence π, because the general statement then follows by iterating the immediate-child claim along the unique ancestor path in the term tree from the occurrence σ to the root occurrence ρ.

Suppose first that π is obtained from σ by an introduce-vertex or introduce-edge constructor. By the semantics from Section 3.4, the realized graph of π is obtained from $\mathcal{G}(\sigma)$ by adding at most one vertex or one edge. Hence $\mathcal{G}(\sigma)$ is a subgraph of $\mathcal{G}(\pi)$.

Suppose next that π is obtained from σ by a forget constructor. Forgetting changes only the active bag and does not delete vertices or edges from the realized graph, so $\mathcal{G}(\sigma) \simeq \mathcal{G}(\pi)$.

Finally, suppose $\pi = \texttt{join}(\sigma_1, \sigma_2)$ and the selected occurrence σ is one of the two child

occurrences. The join semantics glues the child partial graphs along their common bag and takes their union, so the realized graph of each child embeds as a subgraph of $\mathcal{G}(\pi)$.

Thus every immediate parent of a subterm realizes a supergraph of that subterm, up to isomorphism. Iterating this argument from the occurrence σ up to the root occurrence ρ yields the claimed subgraph relation. □

Observation 2. *Let $\mathcal{P}$ be a subgraph-closed property. Let $\rho \in \mathsf{ITD}_k$, and let σ be a subterm occurrence of ρ. If $\mathcal{G}(\sigma) \notin \mathcal{P}$, then $\mathcal{G}(\rho) \notin \mathcal{P}$.*

Proof. Lemma 18 yields a subgraph $H \subseteq \mathcal{G}(\rho)$ that is isomorphic to $\mathcal{G}(\sigma)$. If $\mathcal{G}(\rho) \in \mathcal{P}$, then by subgraph-closedness we have $H \in \mathcal{P}$. Graph properties in this thesis are understood up to graph isomorphism, so $\mathcal{G}(\sigma) \in \mathcal{P}$ as well. By contraposition, if $\mathcal{G}(\sigma) \notin \mathcal{P}$, then $\mathcal{G}(\rho) \notin \mathcal{P}$. □

This observation is the heart of Premise Search: if a partial structure already violates the premise, we can disregard all its extensions. Operationally, this means that any partial decomposition whose graph already fails the premise can be pruned together with all extensions of it. In the DP-core setting, the observation is applied to partial terms and their witness sets.

7.2 From Pairs to Triples: Premise-Based Refutation

Recall that a $(k, \mathcal{D})$-pair $(\mathbf{b}, W)$ consists of a bag $\mathbf{b}$ and a witness set $W \subseteq \mathcal{D}[k].\mathcal{W}$ for the DP-core $\mathcal{D}$. For such a pair $(\mathbf{b}, W)$, we write

$$\mathtt{Acc}_{\mathcal{D}[k]}(\mathbf{b}, W) = 1 \quad \Longleftrightarrow \quad \exists w \in W \text{ with } \mathcal{D}[k].\mathtt{Final}(\mathbf{b}, w) = 1.$$

When the target property is an implication, we refine the state representation from pairs to triples.

Definition 51 (Premise Triple)**.** *A $(k, \mathcal{D}_1, \mathcal{D}_2)$-premise triple is a triple $(\mathbf{b}, W_1, W_2)$ such that for each $j \in \{1, 2\}$, the pair $(\mathbf{b}, W_j)$ is well-formed, i.e. every witness in $W_j \subseteq \mathcal{D}_j[k].\mathcal{W}$ has support contained in $\mathbf{b}$. We call a premise triple a counterexample triple if*

$$\mathtt{Acc}_{\mathcal{D}_1[k]}(\mathbf{b}, W_1) = 1 \quad \textit{and} \quad \mathtt{Acc}_{\mathcal{D}_2[k]}(\mathbf{b}, W_2) = 0.$$

A reachable premise triple records the joint root evaluation of some partial instructive term. By induction on the instructive progression rules, every reachable triple arises as the joint root state of some instructive term τ with $B(\tau) = \mathbf{b}$ and root witness sets (W_1, W_2). If a root-closed (empty root bag) presentation is desired for rechecking, one may post-process τ by appending forget constructors at the root without changing the realized graph. During search, however, we maintain arbitrary premise triples. To read their acceptance bits as graph-level truth values, we use the following lemma.

Lemma 19 (Acceptance Reflects Induced Property)**.** *Let $\mathcal{C} = \mathsf{ITD}$, let $\mathcal{D}$ be a $\mathcal{C}$-coherent DP-core, and let $(\mathbf{b}, W)$ be a state reachable from the initial state by the instructive progression rules. Then there exists an instructive tree decomposition term $\tau \in \mathsf{ITD}_k$ realizing a partial graph $H = \mathcal{G}(\tau)$ such that $B(\tau) = \mathbf{b}$ and $W = \Gamma[\mathcal{D}, k](\tau)$. Moreover,*

$$\mathtt{Acc}_{\mathcal{D}[k]}(\mathbf{b}, W) = 1 \quad \Longleftrightarrow \quad H \in \mathcal{P}[\mathcal{D}, \mathcal{C}].$$

Proof. The existence of τ follows by induction on the derivation of the reachable state. The initial state is realized by the leaf term. Each introduce-vertex, forget-vertex, and introduce-edge step corresponds to applying the matching constructor at the root of a previously realized term. A join step combines two previously realized terms with the same root bag, exactly as required by the join constructor of instructive decompositions. Hence every reachable state is the root state of some instructive term $\tau \in \mathsf{ITD}_k$.

For the equivalence, fix such an τ. Since $W = \Gamma[\mathcal{D}, k](\tau)$ and $B(\tau) = \mathbf{b}$, the definition of $\mathtt{Acc}_{\mathcal{D}[k]}$ gives

$$\mathtt{Acc}_{\mathcal{D}[k]}(\mathbf{b}, W) = 1 \quad \Longleftrightarrow \quad \tau \in \mathsf{Acc}(\mathcal{D}[k]).$$

Because $\tau \in \mathsf{ITD}_k$ and $\mathcal{D}$ is $\mathcal{C}$-coherent, Proposition 4 yields

$$\tau \in \mathsf{Acc}(\mathcal{D}[k]) \quad \Longleftrightarrow \quad \mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D}, \mathcal{C}].$$

Since $H = \mathcal{G}(\tau)$, the claim follows. □

Consequently, a reachable premise triple $(\mathbf{b}, W_1, W_2)$ is a genuine counterexample precisely when the realized graph of its witnessing term lies in $\mathcal{P}[\mathcal{D}_1, \mathsf{ITD}]$ but not in $\mathcal{P}[\mathcal{D}_2, \mathsf{ITD}]$. Conversely, if a reachable state fails one premise bit and that induced premise property is subgraph-closed, then Observation 2 shows that no extension can restore the premise, so the state may be pruned.

7.3 Boolean Combinations of Graph Properties

Many properties of interest can be expressed as Boolean combinations of simpler properties. Formally, given graph properties $\mathcal{P}_1, \ldots, \mathcal{P}_r$ and a Boolean function $\varphi : \{0,1\}^r \to \{0,1\}$, we define:

$$\varphi(\mathcal{P}_1, \ldots, \mathcal{P}_r) = \{H \mid \varphi(\mathcal{P}_1(H), \ldots, \mathcal{P}_r(H)) = 1\}.$$

Common operators include:

$$\begin{aligned} \neg\mathcal{P} &: \text{complement of } \mathcal{P}, \\ \mathcal{P}_1 \wedge \mathcal{P}_2 &: \text{intersection}, \\ \mathcal{P}_1 \vee \mathcal{P}_2 &: \text{union}, \\ \mathcal{P}_1 \to \mathcal{P}_2 &: \text{implication, i.e., } \neg\mathcal{P}_1 \vee \mathcal{P}_2. \end{aligned}$$

These combinations can be represented via collections of DP-cores and evaluated through a refutation strategy that jointly considers the witness structures for each individual core. In particular, the combinator construction of Theorem 9 applies unchanged when the premise contains subgraph-closed properties.

7.4 Refutations with Combined Cores

Given a collection of coherent DP-cores $\mathcal{D}_1, \ldots, \mathcal{D}_r$ and a Boolean function $\varphi : \{0,1\}^r \to \{0,1\}$, we define the following:

Definition 52 ($(k, \varphi, \mathcal{D}_1, \ldots, \mathcal{D}_r)$-Refutation)**.** *Write $\mathbf{S}_i = (S_i^1, \ldots, S_i^r)$ for an r-tuple of witness sets with $S_i^j \subseteq \mathcal{D}_j[k].\mathcal{W}$. We call a state $(\mathbf{b}_i, \mathbf{S}_i)$ well-formed if for every $j \in [r]$, every witness in S_i^j has support contained in $\mathbf{b}_i$. A well-formed state $(\mathbf{b}_i, \mathbf{S}_i)$ is $(k, \varphi, \mathcal{D}_1, \ldots, \mathcal{D}_r)$-inconsistent if*

$$\varphi\big(\mathtt{Acc}_{\mathcal{D}_1[k]}(\mathbf{b}_i, S_i^1), \ldots, \mathtt{Acc}_{\mathcal{D}_r[k]}(\mathbf{b}_i, S_i^r)\big) = 0.$$

A sequence $R = (\mathbf{b}_0, \mathbf{S}_0), \ldots, (\mathbf{b}_m, \mathbf{S}_m)$ is a refutation if:

1. ***Well-formedness:*** *each state $(\mathbf{b}_i, \mathbf{S}_i)$ is well-formed.*
2. ***Initialization:*** $(\mathbf{b}_0, \mathbf{S}_0) = \big(\emptyset,\ (\mathcal{D}_1[k].\widehat{\mathsf{leaf}}(), \ldots, \mathcal{D}_r[k].\widehat{\mathsf{leaf}}())\big)$.

3. ***Refutation:*** *the final state is inconsistent, i.e.*

$$\varphi\big(\mathtt{Acc}_{\mathcal{D}_1[k]}(\mathbf{b}_m, S_m^1), \ldots, \mathtt{Acc}_{\mathcal{D}_r[k]}(\mathbf{b}_m, S_m^r)\big) = 0.$$

4. ***Progression:*** *for each $i \in \{1, \ldots, m\}$, one of the following holds:*

   (a) ***Introduce a vertex:*** *there exist $t < i$ and $u \in [k+1] \setminus \mathbf{b}_t$ such that*

$$\mathbf{b}_i = \mathbf{b}_t \cup \{u\} \qquad \text{and} \qquad S_i^j = \mathcal{D}_j[k].\widehat{\mathsf{intro}}_u(S_t^j) \quad \text{for all } j \in [r].$$

   (b) ***Forget a vertex:*** *there exist $t < i$ and $u \in \mathbf{b}_t$ such that*

$$\mathbf{b}_i = \mathbf{b}_t \setminus \{u\} \qquad \text{and} \qquad S_i^j = \mathcal{D}_j[k].\widehat{\mathsf{forget}}_u(S_t^j) \quad \text{for all } j \in [r].$$

   (c) ***Introduce an edge:*** *there exist $t < i$ and distinct $u, v \in \mathbf{b}_t$ such that*

$$\mathbf{b}_i = \mathbf{b}_t \qquad \text{and} \qquad S_i^j = \mathcal{D}_j[k].\widehat{\mathsf{intro}}_{u,v}(S_t^j) \quad \text{for all } j \in [r].$$

   (d) ***Join:*** *there exist $t_1, t_2 < i$ such that*

$$\mathbf{b}_{t_1} = \mathbf{b}_{t_2} = \mathbf{b}_i \qquad \text{and} \qquad S_i^j = \mathcal{D}_j[k].\widehat{\mathsf{join}}(S_{t_1}^j, S_{t_2}^j) \quad \text{for all } j \in [r].$$

The refutation procedure is a breadth-first exploration of all reachable states, terminating when an inconsistent state in the sense of Definition 52 is found.

Lemma 20 (Reachable combined states come from one instructive term)**.** *Let $\mathcal{C} = \mathsf{ITD}$, let $\mathcal{D}_1, \ldots, \mathcal{D}_r$ be instructive DP-cores, and let $(\mathbf{b}, \mathbf{S})$ with $\mathbf{S} = (S^1, \ldots, S^r)$ be any state reachable from the leaf state by the progression rules of the preceding definition. Then there exists an instructive term $\tau \in \mathsf{ITD}_k$ such that*

$$B(\tau) = \mathbf{b} \qquad \text{and} \qquad \Gamma[\mathcal{D}_j, k](\tau) = S^j \quad \text{for every } j \in [r].$$

Proof. By induction on the derivation of the reachable state. The initial state is realized by the leaf term, and each component equals the corresponding leaf witness set by definition. For an introduce-vertex, forget-vertex, or introduce-edge step, apply the matching constructor at the root of the term realizing the predecessor state; because the DP-core operations are applied componentwise, the resulting parent term realizes the entire tuple of successor witness sets. For a join step, the two predecessor states have the same bag by definition. By the induction hypothesis they are realized by instructive terms τ_1, τ_2 with $B(\tau_1) = B(\tau_2) = \mathbf{b}$, so $\mathtt{join}(\tau_1, \tau_2)$ is an instructive term.

Its root witness set in each component is exactly the join transition used in the reachable-state derivation, again because the progression rule acts componentwise. Thus the same instructive term realizes the whole tuple. □

Lemma 21 (Instructive terms are closed under subterms)**.** *If $\tau \in \mathsf{ITD}_k$ and $\tau = a(\tau_1, \ldots, \tau_r)$, then $\tau_i \in \mathsf{ITD}_k$ for every i. In particular, every subterm occurrence of an instructive term is instructive.*

Proof. Immediate from the inductive definition of ITD_k as a term language with local constructor side conditions. This uses the root-open convention from Section 3.4: the ITD validator has $F_k = Q_k$, so a locally valid subterm is itself accepted as an instructive term with its own root bag. Under a root-closed-only convention, only the root occurrence would necessarily be accepted without adding extra forget constructors. □

7.5 The Premise Search Algorithm

We now present the algorithm that searches for such a refutation, using premise-based pruning when possible. Fix a set of premise indices $I \subseteq [r]$ such that for every $a \in \{0,1\}^r$,

$$\varphi(a) = 0 \implies a_i = 1 \text{ for all } i \in I.$$

For instance, for an implication $\varphi(a_1, a_2) = \neg a_1 \vee a_2$ we may take $I = \{1\}$. We say that the *premise holds* for a state $(\mathbf{b}, \mathbf{S})$ (with $\mathbf{S} = (S^1, \ldots, S^r)$) if $\mathtt{Acc}_{\mathcal{D}_i[k]}(\mathbf{b}, S^i) = 1$ for every $i \in I$. When each $\mathcal{D}_i$ with $i \in I$ defines a subgraph-closed property, Observation 2 justifies pruning: once the premise fails at some subterm, it can never become true again along any extensions. The queue is initialized with the leaf state. After that, a generated successor X is enqueued only if it satisfies all premise cores, so every non-initial state in Q satisfies the premise. Since the refutation framework of Chapter 5 treats join as an ordered operation, Algorithm 3 explicitly enumerates both join orientations. This pruning is sound only under the subgraph-closedness hypothesis. If the premise is not subgraph-closed, pruning may discard valid counterexamples and should be disabled.

Example (Reed-type premises). In Chapter 8, the premise $\mathcal{A}_s = \neg\textsc{MultiEdge} \wedge \neg\textsc{Clique}_3 \wedge \neg\textsc{MaxDeg}_{s+1}$ is subgraph-closed. Hence any partial term whose realized graph already violates $\mathcal{A}_s$ (e.g., by containing a triangle in the underlying simple graph, or by exceeding the maximum-degree bound) can be pruned together with all of its extensions.

For the theorem below, we read Algorithm 3 as an explicit procedure on encoded bags and witness sets. Accordingly, we assume the constituent DP-cores are internally polynomial in the sense of Definition 33. Then the acceptance predicates are computable by scanning the explicit witness sets and evaluating `Final` on each witness, while the lifted introduce-vertex, forget-vertex, introduce-edge, and join maps used by the algorithm are computable by Lemma 8. Equality of search states is equality of these explicit encodings. For certificate reconstruction, the implementation stores with every newly discovered state an insertion certificate: the constructor used and the predecessor state or predecessor pair from which it was generated (with the leaf state carrying the initial certificate). Thus the list returned by the algorithm is read together with these predecessor pointers; unfolding them recovers the refutation term, and hence the corresponding counterexample graph, when the algorithm returns a rejecting state.

Theorem 14 (Correctness and Termination)**.** *Let $\mathcal{C} = \mathsf{ITD}$. Let $\mathcal{D}_1, \dots, \mathcal{D}_r$ be finite, internally polynomial, instructive, $\mathcal{C}$-coherent DP-cores that preserve well-formedness in the sense of Definition 44, let $\varphi : \{0,1\}^r \to \{0,1\}$ be a Boolean function, and let $I \subseteq [r]$ satisfy that for every $a \in \{0,1\}^r$,*

$$\varphi(a) = 0 \implies a_i = 1 \text{ for all } i \in I.$$

Suppose that for each $i \in I$, the induced component property $\mathcal{P}[\mathcal{D}_i, \mathcal{C}]$ is subgraph-closed in the graph-category convention fixed at the beginning of this chapter. Then Algorithm 3 terminates. Moreover, it examines at most

$$2^{k+1} \cdot \prod_{i=1}^{r} \delta_{\mathcal{D}_i}(k)$$

distinct search states. Finally, the algorithm returns a refutation if and only if

$$\mathcal{P}[\mathcal{C}_k] \not\subseteq \varphi(\mathcal{P}[\mathcal{D}_1, \mathcal{C}], \dots, \mathcal{P}[\mathcal{D}_r, \mathcal{C}]).$$

Proof. Executability. Internal polynomiality and Lemma 8 ensure that every local operation queried by Algorithm 3 is computable on explicit encodings of bags and witness sets, and equality of stored states is just equality of those encodings. Hence the pseudocode defines an actual finite-state reachability procedure.

Well-formedness. We claim that every state generated by Algorithm 3 is well-formed in the sense of Definition 52. For the initial state $(\emptyset, \mathbf{S}_0)$, each component $(\emptyset, \mathcal{D}_i[k].\widehat{\mathsf{leaf}}())$ is well-formed by Definition 44(1). For every later generated state, the algorithm applies one of the four instructive lifted transitions componentwise to an already well-formed predecessor state (or pair of states in the join case), so Definition 44(2)–(5) implies that

Input : Finite, internally polynomial DP-cores $\mathcal{D}_1, \dots, \mathcal{D}_r$ over instructive decompositions, Boolean function $\varphi : \{0,1\}^r \to \{0,1\}$, premise indices $I \subseteq [r]$ satisfying $\varphi(a) = 0 \Rightarrow a_i = 1$ for all $i \in I$, width bound $k \in \mathbb{N}$.
Side condition: for every $i \in I$, the indexed property $\mathcal{P}[\mathcal{D}_i, \mathsf{ITD}]$ is subgraph-closed; this is used in the soundness analysis of premise pruning, not as a separate runtime input.
Output : A $(k, \varphi, \mathcal{D}_1, \dots, \mathcal{D}_r)$-refutation R if one exists. Otherwise, returns "Inclusion Holds".

$R \leftarrow [\,]$;
$\hat{R} \leftarrow \emptyset$;
$\mathbf{S}_0 \leftarrow (\mathcal{D}_1[k].\widehat{\mathsf{leaf}}(), \dots, \mathcal{D}_r[k].\widehat{\mathsf{leaf}}())$;
$Q \leftarrow [(\emptyset, \mathbf{S}_0)]$; `// queue of unexplored states`
$\hat{R} \leftarrow \hat{R} \cup \{(\emptyset, \mathbf{S}_0)\}$;
while $Q \neq \emptyset$ **do**
  Remove the first element $(\mathbf{b}, \mathbf{S})$ from Q and append it to R;
  if $\varphi(\mathtt{Acc}_{\mathcal{D}_1[k]}(\mathbf{b}, S^1), \dots, \mathtt{Acc}_{\mathcal{D}_r[k]}(\mathbf{b}, S^r)) = 0$ **then**
    return R; `// In this case, R is a refutation.`
  end
  foreach $u \in [k+1] \setminus \mathbf{b}$ **do**
    $\mathbf{S}' \leftarrow (\mathcal{D}_1[k].\widehat{\mathsf{intro}}_u(S^1), \dots, \mathcal{D}_r[k].\widehat{\mathsf{intro}}_u(S^r))$;
    $X \leftarrow (\mathbf{b} \cup \{u\}, \mathbf{S}')$;
    if $X \notin \hat{R}$ **then**
      $\hat{R} \leftarrow \hat{R} \cup \{X\}$;
      if *premise holds for* X **then**
        Append X to the end of Q;
      end
    end
  end
  foreach $u \in \mathbf{b}$ **do**
    $\mathbf{S}' \leftarrow (\mathcal{D}_1[k].\widehat{\mathsf{forget}}_u(S^1), \dots, \mathcal{D}_r[k].\widehat{\mathsf{forget}}_u(S^r))$;
    $X \leftarrow (\mathbf{b} \setminus \{u\}, \mathbf{S}')$;
    if $X \notin \hat{R}$ **then**
      $\hat{R} \leftarrow \hat{R} \cup \{X\}$;
      if *premise holds for* X **then**
        Append X to the end of Q;
      end
    end
  end
  foreach $u, v \in \mathbf{b}$ *with* $u \neq v$ **do**
    $\mathbf{S}' \leftarrow (\mathcal{D}_1[k].\widehat{\mathsf{intro}}_{u,v}(S^1), \dots, \mathcal{D}_r[k].\widehat{\mathsf{intro}}_{u,v}(S^r))$;
    $X \leftarrow (\mathbf{b}, \mathbf{S}')$;
    if $X \notin \hat{R}$ **then**
      $\hat{R} \leftarrow \hat{R} \cup \{X\}$;
      if *premise holds for* X **then**
        Append X to the end of Q;
      end
    end
  end
  foreach $(\mathbf{b}', \boldsymbol{T}) \in R$ **do**
    if $\mathbf{b}' = \mathbf{b}$ **then**
      $\mathbf{S}' \leftarrow (\mathcal{D}_1[k].\widehat{\mathsf{join}}(S^1, T^1), \dots, \mathcal{D}_r[k].\widehat{\mathsf{join}}(S^r, T^r))$;
      $X \leftarrow (\mathbf{b}, \mathbf{S}')$;
      if $X \notin \hat{R}$ **then**
        $\hat{R} \leftarrow \hat{R} \cup \{X\}$;
        if *premise holds for* X **then**
          Append X to the end of Q;
        end
      end
      $\mathbf{S}' \leftarrow (\mathcal{D}_1[k].\widehat{\mathsf{join}}(T^1, S^1), \dots, \mathcal{D}_r[k].\widehat{\mathsf{join}}(T^r, S^r))$;
      $X \leftarrow (\mathbf{b}, \mathbf{S}')$;
      if $X \notin \hat{R}$ **then**
        $\hat{R} \leftarrow \hat{R} \cup \{X\}$;
        if *premise holds for* X **then**
          Append X to the end of Q;
        end
      end
    end
  end
end
return "Inclusion Holds";

Algorithm 3: Premise-Pruned Inclusion Test

every component pair remains well-formed. Hence every state recorded in $Q \cup \hat{R} \cup R$ is well-formed.

Termination and bound. In the instructive setting, each search state has the form $(\mathbf{b}, \mathbf{S})$ where $\mathbf{b} \subseteq [k+1]$ is the current bag and $\mathbf{S} = (S^1, \ldots, S^r)$ is a tuple of witness-set states. There are exactly 2^{k+1} choices for $\mathbf{b}$. By Lemma 20, every generated tuple of witness-set components is realized by an instructive term τ. Therefore each component $S^i = \Gamma[\mathcal{D}_i, k](\tau)$ is a useful witness set for $\mathcal{D}_i$ at width k. Since each $\mathcal{D}_i$ is finite, there are at most $\delta_{\mathcal{D}_i}(k)$ possible useful witness sets for the i-th component. Algorithm 3 records each generated state in $\hat{R}$ and never enqueues a state twice, so it terminates after examining at most the stated number of states.

Soundness. If the algorithm returns R, let $(\mathbf{b}, \mathbf{S})$ be its last state. Then

$$\varphi(\mathtt{Acc}_{\mathcal{D}_1[k]}(\mathbf{b}, S^1), \ldots, \mathtt{Acc}_{\mathcal{D}_r[k]}(\mathbf{b}, S^r)) = 0.$$

By Lemma 20, there exists an instructive term $\tau \in \mathsf{ITD}_k$ whose root bag is $\mathbf{b}$ and whose computed root tuple is $\mathbf{S}$. Applying Lemma 19 componentwise, for each $j \in [r]$ we have

$$\mathtt{Acc}_{\mathcal{D}_j[k]}(\mathbf{b}, S^j) = 1 \quad \Longleftrightarrow \quad \mathcal{G}(\tau) \in \mathcal{P}[\mathcal{D}_j, \mathcal{C}].$$

Since φ evaluates to 0 on the acceptance bits, we obtain

$$\mathcal{G}(\tau) \notin \varphi(\mathcal{P}[\mathcal{D}_1, \mathcal{C}], \ldots, \mathcal{P}[\mathcal{D}_r, \mathcal{C}]).$$

By Lemma 2, $\mathcal{G}(\tau)$ has treewidth at most k. Thus τ witnesses a counterexample graph of width at most k, and the returned sequence is a valid $(k, \varphi, \mathcal{D}_1, \ldots, \mathcal{D}_r)$-refutation: initialization and progression hold by construction, the final state satisfies the refutation clause by the displayed equation, and the well-formedness clause holds by the preceding paragraph. As before, the witnessing term can be made root-closed by appending forget constructors at the root.

Completeness. We first record the queue invariant used below:

$$X \in \hat{R} \text{ and premise holds for } X \quad \Longrightarrow \quad X \in Q \cup R.$$

Indeed, when a state is generated for the first time, the algorithm inserts it into $\hat{R}$. If the premise holds for that state, the same update step appends it to Q. Thereafter the state remains either queued or already processed, because states are removed from Q only when they are appended to R.

Assume the inclusion fails. Then there exists a graph

$$G \in \mathcal{P}[\mathcal{C}_k] \setminus \varphi(\mathcal{P}[\mathcal{D}_1, \mathcal{C}], \ldots, \mathcal{P}[\mathcal{D}_r, \mathcal{C}]).$$

Because $\mathcal{C} = \mathsf{ITD}$, choose $\tau^\star \in \mathsf{ITD}_k$ with $\mathcal{G}(\tau^\star) \simeq G$. For every subterm occurrence τ' of $\tau^\star$, define the associated search state

$$X_{\tau'} := \Big(B(\tau'), \big(\Gamma[\mathcal{D}_1, k](\tau'), \ldots, \Gamma[\mathcal{D}_r, k](\tau')\big)\Big).$$

We claim by structural induction on τ' that, unless the algorithm has already halted with a refutation, it eventually processes every state $X_{\tau'}$.

For the leaf term $\mathtt{leaf}$, the state $X_{\mathtt{leaf}}$ is exactly the initial state of Algorithm 3, hence it is processed unless the algorithm halts immediately with a refutation.

Suppose next that $\tau' = a(\sigma)$ is obtained from a unary child σ. By the induction hypothesis, X_σ is eventually processed unless the algorithm has already halted with a refutation, in which case we are done. When the algorithm processes X_σ, it applies the corresponding lifted DP-core operation componentwise, so it generates exactly the raw successor $X_{\tau'}$. To see that $X_{\tau'}$ is not pruned, fix $i \in I$. By the definition of $\mathtt{Acc}$,

$$\mathtt{Acc}_{\mathcal{D}_i[k]}\big(B(\tau'), \Gamma[\mathcal{D}_i, k](\tau')\big) = 1 \iff \tau' \in \mathsf{Acc}(\mathcal{D}_i[k]).$$

Since τ' is a subterm occurrence of the instructive term $\tau^\star$, Lemma 21 gives $\tau' \in \mathsf{ITD}_k$. Therefore Proposition 4 gives

$$\tau' \in \mathsf{Acc}(\mathcal{D}_i[k]) \iff \mathcal{G}(\tau') \in \mathcal{P}[\mathcal{D}_i, \mathcal{C}].$$

Let $a^\star \in \{0,1\}^r$ be the membership vector of G in the component properties. Because $G \notin \varphi(\cdots)$, we have $\varphi(a^\star) = 0$, so the hypothesis on I implies that $a^\star_i = 1$ for every $i \in I$. Hence $\mathcal{G}(\tau^\star) \in \mathcal{P}[\mathcal{D}_i, \mathcal{C}]$ for every premise index i. The contrapositive of Observation 2 then implies $\mathcal{G}(\tau') \in \mathcal{P}[\mathcal{D}_i, \mathcal{C}]$, because each premise property is subgraph-closed and τ' is a subterm occurrence of $\tau^\star$. Thus all premise bits of $X_{\tau'}$ are 1. If $X_{\tau'} \notin \hat{R}$, then the algorithm appends $X_{\tau'}$ to Q when it is first generated. If $X_{\tau'} \in \hat{R}$ already, then the premise bits are still 1 because they depend only on the state $X_{\tau'}$, so the queue invariant implies $X_{\tau'} \in Q \cup R$. In either case, unless the algorithm has already halted with a refutation, $X_{\tau'}$ is eventually processed.

Finally, let $\tau' = \mathtt{join}(\sigma_1, \sigma_2)$. By the induction hypothesis, both X_{σ_1} and X_{σ_2} are eventually processed unless the algorithm has already halted with a refutation. When the later of the two states is processed, the other one is already stored. If the two children

yield the same search state, this is still covered: the algorithm first appends the state currently being processed to R and then iterates over all stored states with the same bag, including that very state. Hence in every case the algorithm explicitly generates both ordered join successors. One of these is exactly $X_{\tau'}$, because dynamization applies the lifted join transition componentwise. The same premise argument as above shows that $X_{\tau'}$ is not pruned. Therefore every state $X_{\tau'}$ along the constructor trace of $\tau^\star$ is eventually processed, in particular the root state $X_{\tau^\star}$.

For $X_{\tau^\star}$, the acceptance bits are exactly the membership bits of G in the component properties, so $\varphi = 0$ at that state. Therefore Algorithm 3 eventually reaches a $(k, \varphi, \mathcal{D}_1, \ldots, \mathcal{D}_r)$-inconsistent state and returns a refutation. □

Remark 13 (Premise-bounded termination for non-finite cores). Theorem 14 assumes that all cores are finite. This guarantees a finite explored state space at fixed width. For non-finite invariant cores, Algorithm 3 may fail to terminate because witness sets can encode arbitrarily large integers even at width 0. A sufficient application-specific replacement is a computable finite-domain condition on the premise-restricted search: for the fixed width k, suppose that for each component $\mathcal{D}_i$ there is a computable finite family $\mathcal{U}_i(k)$ of witness sets such that every premise-satisfying reachable term τ satisfies

$$\Gamma[\mathcal{D}_i, k](\tau) \in \mathcal{U}_i(k).$$

Then the same state-space argument bounds the premise-restricted search by

$$2^{k+1} \cdot \prod_{i=1}^{r} |\mathcal{U}_i(k)|$$

stored states. Numeric side conditions, such as a vertex-count bound, are useful for non-finite cores only insofar as they imply such finite families of reachable witness sets.

7.6 Summary

Premise Search enhances the efficiency of model checking and ATP on graphs of bounded treewidth by exploiting the structural monotonicity of subgraph-closed properties. When the premise of a logical implication is subgraph-closed, we may avoid constructing and verifying any supergraph of a graph that already violates the premise. This pruning mechanism, implemented through structured triples and Boolean combinations of DP-cores, allows early termination of unfruitful search branches and leads to faster, more scalable verification.

Chapter 8

Triangle-Free Reed Case Study

In this chapter, we present a case study of width-based automated theorem proving applied to Reed's coloring conjecture [Reed, 1998], which states that every simple graph of maximum degree Δ and clique number ω admits a proper coloring with at most $\lceil \frac{\Delta+\omega+1}{2} \rceil$ colors. Our computational focus, however, is the *triangle-free* specialization: if G is triangle-free then $\omega(G) \leq 2$, and Reed's conjecture implies the simpler bound

$$\chi(G) \leq \left\lceil \frac{\Delta(G)+3}{2} \right\rceil.$$

For $\Delta \leq 4$ this inequality follows from standard results (Brooks' theorem together with the odd cycle case), so the first potentially non-trivial maximum-degree case is $\Delta = 5$. More generally, the workflow below encodes the triangle-free Reed bound as a parameterized family and tests it on bounded treewidth/pathwidth classes using TreeWidzard. The case-study setup follows the approach introduced in [de Oliveira Oliveira and Vadiee, 2024] and is integrated here with our framework and evaluation pipeline.

The goals of this chapter are fourfold: to give a concrete example of width-based automated theorem proving in action, to demonstrate the effectiveness of the search strategies introduced in Chapters 6 and 7, to report verification results for the triangle-free specialization on bounded graph classes, and to show tightness phenomena by producing counterexamples for natural strengthenings using TreeWidzard.

8.1 Reed's Conjecture and the Triangle-Free Case

We recall that a proper c-coloring of a graph $G = (V, E)$ is a map $\varphi : V \to [c]$ such that $\varphi(u) \neq \varphi(v)$ whenever $\{u, v\} \in E$. The chromatic number $\chi(G)$ is the least c for which a proper c-coloring exists.

In graph coloring theory, a classical fact is that every graph G satisfies

$$\omega(G) \leq \chi(G) \leq \Delta(G) + 1,$$

where $\omega(G)$ is the size of the largest clique in G, $\chi(G)$ is the chromatic number, and $\Delta(G)$ is the maximum degree of G. The lower bound $\chi(G) \geq \omega(G)$ is immediate (a clique needs distinct colors), while the upper bound $\chi(G) \leq \Delta(G) + 1$ comes from the greedy coloring algorithm [Bondy and Murty, 2008].

In 1998, Reed proposed a far-reaching conjecture suggesting that $\chi(G)$ should lie closer to the average of these two extremes [Reed, 1998]. Formally:
Conjecture 1 ([Reed, 1998]). For every graph G,

$$\chi(G) \leq \left\lceil \frac{\Delta(G) + 1 + \omega(G)}{2} \right\rceil.$$

This bound is tight: complete graphs K_n (where $\omega = n$ and $\Delta = n - 1$) and odd cycles (where $\omega = 2$ and $\Delta = 2$) both achieve equality. The conjecture captures a deep intuition in graph theory: large cliques force large chromatic number, but the converse need not hold, and the conjecture posits that the pair (Δ, ω) nevertheless tightly constrains χ.

The proposed bound, if true, would unify and significantly sharpen earlier results. For example, Brooks' Theorem shows that unless G is a complete graph or an odd cycle, $\chi(G) \leq \Delta(G)$ [Brooks, 1941]. Reed's conjecture can be viewed as interpolating between the trivial bounds $\omega(G)$ and $\Delta(G) + 1$, and it recovers Brooks' conclusion whenever $\omega(G) \leq \Delta(G) - 1$.

Bridging Lower and Upper Bounds. The conjecture states that $\chi(G)$ is at most the average of $\omega(G)$ and $\Delta(G) + 1$:

$$\chi(G) \leq \left\lceil \frac{1}{2}(\Delta(G) + 1 + \omega(G)) \right\rceil.$$

In cases where Δ greatly exceeds ω, this represents a dramatic improvement over the naïve upper bound. Large cliques are one reason for large χ, but high chromatic number

can also occur with small clique number. Reed's conjecture predicts that nevertheless the pair (Δ, ω) controls χ. In particular, it leads to families of χ-bounded results: for fixed ω, the chromatic number is $O(\Delta)$, and for fixed Δ, it is $O(\omega)$. Thus, graphs with small cliques cannot have chromatic number close to $\Delta + 1$ if the conjecture holds.

Strengthening of Brooks' Theorem. As mentioned, Brooks' Theorem ensures $\chi(G) \leq \Delta(G)$ except in the complete-graph and odd-cycle cases. The conjecture offers a quantitative refinement. For instance, an odd cycle C_{2k+1} has $\omega = 2$ and $\Delta = 2$, so the bound becomes

$$\chi(C_{2k+1}) \leq \left\lceil \frac{2+1+2}{2} \right\rceil = 3,$$

which is tight, as odd cycles are 3-colorable. More generally, for graphs saturating Brooks' bound ($\chi = \Delta$), the conjecture implies

$$\chi \leq \left\lceil \frac{\Delta + 1 + \omega}{2} \right\rceil.$$

When $\omega \leq \Delta - 1$, this yields

$$\chi \leq \left\lceil \frac{\Delta + 1 + (\Delta - 1)}{2} \right\rceil = \Delta,$$

which is strictly less than $\Delta+1$. When $\omega = \Delta+1$ (as in complete graphs), the conjecture yields $\chi \leq \Delta + 1$ and the bound is tight. Hence, if the conjecture holds, only graphs with clique number very close to $\Delta + 1$ can require nearly $\Delta + 1$ colors.

Chromatic Clique Tradeoff. Another perspective is to consider the gap $\Delta + 1 - \omega$. The conjecture implies that χ cannot lie too close to the upper end of this interval. Instead, it is pushed toward ω by about half the gap. Concretely, if $\omega(G) \leq r$ for some fixed constant r, then Reed's bound gives

$$\chi(G) \leq \left\lceil \frac{\Delta(G) + r + 1}{2} \right\rceil.$$

In particular, for each fixed clique bound r, this yields $\chi(G) \leq \frac{1}{2}\Delta(G) + O(1)$. This validates the intuition that sparse graphs, even with large maximum degree, are not necessarily difficult to color. Proving such a universal 1/2 factor would be a major breakthrough, highlighting the role of clique number as a controlling term. Conversely, it would imply that graphs with bounded clique number cannot require chromatic number close to the trivial upper bound $\Delta + 1$.

8.2 Results and Special Cases

Despite substantial effort over the past two decades, Reed's Conjecture remains open in full generality. However, a number of important partial results and confirmations have been achieved for special graph classes and in approximate forms. We summarize some of the most notable developments.

(a) Reed's ε-Theorem (1998) and Improvements. As initial evidence for the conjecture, Reed proved an "ε-version": there exist constants $\varepsilon > 0$ and Δ_0 such that every graph G with $\Delta(G) \geq \Delta_0$ satisfies

$$\chi(G) \leq \lceil (1-\varepsilon)(\Delta(G)+1) + \varepsilon\omega(G) \rceil$$

[Reed, 1998]. This result falls short of the desired $\frac{1}{2}$ weighting, but it already implies a strict improvement over the greedy bound $\Delta + 1$ whenever $\omega(G) < \Delta(G) + 1$.

Reed's original argument did not specify a large ε. In fact, it yields an extremely small explicit value, on the order of 10^{-8} (see, e.g., the threshold $1/70\,000\,000$ quoted in [Kang, 2022]) and requires sufficiently large maximum degree. Nonetheless, it confirmed the qualitative idea that χ can be bounded away from $\Delta + 1$.

In subsequent years, this ε-bound was improved by other researchers. For example, King and Reed showed that one may take $\varepsilon = 1/130\,000$ for sufficiently large Δ [King and Reed, 2016], and Bonamy, Perrett, and Postle improved this to $\varepsilon = 1/26$ [Bonamy et al., 2022]. Further improvements (including work attributed to Delcourt and Postle) are recorded in [Kang, 2022], bringing the bounds closer to the conjectured $\varepsilon = \frac{1}{2}$ threshold.

Each of these results can be seen as making Reed's Conjecture "more true" in an asymptotic sense, by showing that even tighter bounds are possible under broad conditions. However, the ultimate $\frac{1}{2}$ barrier remains elusive and appears out of reach for current techniques.

(b) Triangle-Free Graphs ($\omega \leq 2$). One important special case is when G has no triangles (i.e., $\omega(G) \leq 2$). In this case, Conjecture 1 demands

$$\chi(G) \leq \left\lceil \frac{\Delta(G)+3}{2} \right\rceil.$$

While this specific bound remains unproved in the worst case, a landmark result of Johansson (1996) establishes an even stronger statement for triangle-free graphs: he

showed that there is a constant C such that

$$\chi(G) = O\left(\frac{\Delta(G)}{\ln \Delta(G)}\right)$$

for every triangle-free graph G [Molloy, 2019]. In other words, for sufficiently large maximum degree, triangle-free graphs can be colored with dramatically fewer than $\frac{1}{2}\Delta$ colors.

Johansson's theorem (proved via an ingenious probabilistic method) was originally stated in terms of list coloring and established $\chi_\ell(G) = O(\Delta/\log \Delta)$, which of course implies the same bound for the ordinary chromatic number. This result not only validates Reed's Conjecture for triangle-free graphs in an asymptotic sense, but far surpasses it.

In particular, for large Δ, Johansson's bound guarantees $\chi(G) \ll \frac{1}{2}(\Delta + 1)$, providing a wide safety margin for the conjecture. For small or moderate values of Δ, many cases can be handled by case analysis or computer verification on restricted graph classes. Taken together, these results provide strong evidence and asymptotic progress, but the exact bound $\chi(G) \leq \left\lceil \frac{\Delta(G)+3}{2} \right\rceil$ for all triangle-free graphs remains the target statement. This motivates verifying it on structured classes such as bounded treewidth and pathwidth, which is the focus of the remainder of this chapter.

We note that ongoing research continues to improve coloring bounds for triangle-free and more generally K_t-free graphs (e.g., Molloy's refinement achieving $\chi(G) \leq (1 + o(1))\,\Delta/\ln \Delta$ [Molloy, 2019]). Since these bounds are asymptotically much smaller than the triangle-free Reed target $\lceil(\Delta+3)/2\rceil$, any counterexample to Reed's conjecture would have to occur in regimes where ω grows with Δ, or at small maximum degree.

(c) Claw-Free Graphs. A graph is claw-free if it contains no induced $K_{1,3}$ (no induced "claw"). This rich class includes, for example, all line graphs and many graph families with restricted structure. For claw-free graphs, Reed's Conjecture has been confirmed in full. King and Reed (2015) proved that every claw-free graph indeed satisfies

$$\chi(G) \leq \left\lceil \frac{\Delta(G) + 1 + \omega(G)}{2} \right\rceil,$$

matching the conjectured bound exactly [King and Reed, 2015]. Their proof uses the deep structure theory of claw-free graphs to either find a proper coloring meeting the bound or reduce the graph to a simpler configuration.

King and Reed also discuss a somewhat stronger "local" form of the conjecture, and prove it for claw-free graphs with a three-colorable complement [King and Reed, 2015].

Line graphs (graphs representing edge-adjacency in some host graph) form a prominent subclass of claw-free graphs, and they were one of the first special cases verified.

In a 2007 paper, King, Reed, and Vetta showed that if $L(H)$ is the line graph of an arbitrary graph H, then

$$\chi(L(H)) \leq \left\lceil \frac{\Delta(L(H)) + 1 + \omega(L(H))}{2} \right\rceil,$$

confirming Reed's Conjecture for all line graphs [King et al., 2007]. (Intuitively, coloring a line graph corresponds to edge-coloring the original graph H, and Reed's bound in this context provides an upper bound on the chromatic index of H in terms of the maximum degree and clique number of the line graph.)

The resolution of the conjecture for claw-free graphs is a major positive step, as it covers a broad family beyond trivial cases. It suggests that structural restrictions that forbid certain local configurations (like an induced claw) make the coloring problem more tractable.

(d) Other Graph Classes. A number of additional graph families have been verified to obey Reed's bound, often by tailored arguments. For instance, Schiermeyer (2016) proved that Reed's conjecture holds for P_5-free graphs (graphs with no induced 5-vertex path) in an asymptotic sense, and verified the conjectured bound for several subclasses of P_5-free graphs [Schiermeyer, 2016]. Similarly, graphs that are near-bipartite or have certain forbidden substructures have been checked in the literature. Each such result tends to exploit the structure of the class to find a coloring that meets the $\lceil(\Delta+1+\omega)/2\rceil$ threshold.

While we will not survey every result, it is worth noting that no counterexample to Reed's Conjecture is currently known. Many special graph classes have been verified to satisfy the conjectured bound. The remainder of this chapter contributes additional automated evidence for the triangle-free specialization on bounded-width graph classes.

8.3 Parameterized Version of Reed's Conjecture for Triangle-Free Graphs

The conjecture becomes more approachable once structural parameters are fixed. Throughout the thesis we focus on width-bounded graph classes, with the formal develop-

ment stated below for treewidth. This is the natural setting for width-based automated theorem proving: the search space is tame enough to admit exhaustive exploration, yet still rich enough to contain highly non-trivial configurations. The experiments in Section 8.6 also report the published pathwidth runs from [de Oliveira Oliveira and Vadiee, 2024]; those runs use the corresponding pathwidth-restricted search language in the implementation and are presented as empirical case-study evidence, not as a consequence of Proposition 5. In particular, parameterizing by treewidth isolates exactly the instances represented by the instructive tree decomposition class used in our formal inclusion test, and it provides a direct bridge between the combinatorial bounds of Reed's conjecture and the DP-cores developed in Chapters 4 and 5. We write

$$\textsc{GraphsTw}[k] \;:=\; \mathcal{P}[\mathsf{ITD}_k] \;=\; \{\, G \in \mathrm{GRAPHS} : tw(G) \le k \,\},$$

where the equality follows from Lemma 2. Thus $\textsc{GraphsTw}[k]$ is the ambient treewidth-k slice induced by the ITD semantics, namely the class of incidence structures of loopless undirected multigraphs of treewidth at most k. When we later restrict to simple graphs, this is enforced explicitly by the premise $\neg\textsc{MultiEdge}$ rather than by changing the ambient universe.

It is worth noting that graphs of treewidth at most k are $(k+1)$-colorable [Bodlaender and Fomin, 2005]. The following theorem, due to Dvorák and Kawarabayashi, establishes a better upper bound, in terms of treewidth, for the chromatic number of simple triangle-free graphs. This is the only regime in which we invoke it below; in the ambient multigraph universe, simplicity is enforced explicitly by the guard $\neg\textsc{MultiEdge}$.

Theorem 15 ([Dvořák and Kawarabayashi, 2017])**.** *For any simple triangle-free graph G of treewidth $\le k$, $\chi(G) \le \lceil \frac{k+3}{2} \rceil$.*

Corollary 12 (Large chromatic number forces large treewidth)**.** *If G is a simple triangle-free graph, then $tw(G) \ge 2\chi(G) - 4$.*

Proof. Let G be a simple triangle-free graph and set $t := tw(G)$. By Theorem 15 we have $\chi(G) \le \lceil \frac{t+3}{2} \rceil$. If $t \le 2\chi(G) - 5$ then $\lceil \frac{t+3}{2} \rceil \le \left\lceil \frac{2\chi(G)-2}{2} \right\rceil = \chi(G) - 1$, a contradiction. Hence $t \ge 2\chi(G) - 4$. □

Observation 3 (Reduction to bounded degree)**.** *Fix $k \ge 1$. If G is a simple triangle-free graph and has treewidth at most k, then the inequality $\chi(G) \le \left\lceil \frac{\Delta(G)+3}{2} \right\rceil$ holds whenever $\Delta(G) \ge k$. Consequently, in order to verify the triangle-free specialization on simple graphs of treewidth at most k, it suffices to consider graphs of maximum degree at most $k-1$.*

Proof. Let G be a simple triangle-free graph of treewidth at most k and assume $\Delta(G) \geq k$. By Theorem 15 we have $\chi(G) \leq \lceil \frac{k+3}{2} \rceil$, while $\left\lceil \frac{\Delta(G)+3}{2} \right\rceil \geq \lceil \frac{k+3}{2} \rceil$ since $\Delta(G) \geq k$. Therefore $\chi(G) \leq \left\lceil \frac{\Delta(G)+3}{2} \right\rceil$. In particular, any counterexample of treewidth at most k must satisfy $\Delta(G) \leq k-1$, proving the reduction. □

8.4 DP-core Encoding of the Triangle-Free Case

This section encodes the triangle-free specialization as a Boolean combination of catalog DP-cores.

8.4.1 Boolean Encoding via DP-cores

Throughout this section, constituent graph properties are named in mathematical language. Code-style identifiers are reserved for the explicit property-file example and related implementation remarks.

To express the triangle-free specialization within the DP-core framework, we translate every ingredient of the implication into a DP-core constituent. Let $\textsc{Colorable}_c$ denote the property of being c-colorable, let $\textsc{MaxDeg}_d$ denote the property of having a vertex of degree at least d, let $\textsc{Clique}_\ell$ denote the property of containing a clique of size at least ℓ in the underlying simple graph, and let $\textsc{MultiEdge}$ denote the property of containing a multiple edge. For a fixed integer $s \geq 0$ we set

$$\textsc{Reed}(s) \equiv \Big(\neg\textsc{MultiEdge} \wedge \neg\textsc{Clique}_3 \wedge \neg\textsc{MaxDeg}_{s+1}\Big) \rightarrow \textsc{Colorable}_{\lceil (s+3)/2 \rceil}. \tag{8.1}$$

Note that $\textsc{MaxDeg}_d$ means $\Delta(G) \geq d$. Therefore $\neg\textsc{MaxDeg}_{s+1}$ expresses the bound $\Delta(G) \leq s$. Restricted to simple graphs, $\textsc{Reed}(s)$ says that every triangle-free graph of maximum degree at most s is $\lceil (s+3)/2 \rceil$-colorable.

In TreeWidzard, each instance $\textsc{Reed}(s)$ is provided as a property file that names the constituent cores and then states the Boolean implication (see Appendix B, Table B.1). For example, the case $s = 2$ is encoded as:

```
me   := HasMultipleEdges()
tri  := SimpleCliqueNumber_AtLeast(3)
```

```
deg3 := MaximumDegree_AtLeast(3)
col3 := ChromaticNumber_AtMost(3)
Formula
NOT me IMPLIES (NOT tri IMPLIES (NOT deg3 IMPLIES col3))
```

Property-file identifiers. For each fixed s, the property file instantiates the four mathematical constituents of (8.1) using the implementation identifiers from Appendix B:

- `HasMultipleEdges()` for the multiple-edge guard enforcing simplicity,
- `SimpleCliqueNumber_AtLeast(3)` for triangle exclusion in the underlying simple graph,
- `MaximumDegree_AtLeast(`$s{+}1$`)` for the maximum-degree premise, and
- `ChromaticNumber_AtMost(`$\lceil (s+3)/2 \rceil$`)` for the c-colorability conclusion.

The ATP objective is then the language inclusion induced by this file at width bound k. For the treewidth verification route justified below, it suffices to generate one such file for each pair (k, s) with $s \leq k-1$. The experimental pipeline may additionally generate files for larger values of s, so that the published benchmark tables can compare the same degree bounds across pathwidth and treewidth configurations; run metadata stores each file together with the corresponding ATP logs and evidence artifacts.

Thus, verifying the triangle-free specialization on graphs of treewidth at most k reduces to checking whether $\textsc{GraphsTw}[k] \subseteq \textsc{Reed}(s)$ for every $s \in \{0, \ldots, k-1\}$. Each component of the Boolean formula in (8.1) admits a finite, internally polynomial DP-core, and all constituents except the raw clique detector are coherent on the full instructive class. The clique core is only used behind the explicit simplicity guard $\neg\textsc{MultiEdge}$, and we make that guard formal below. The bitlength $\beta(k)$ of a core is the maximum number of bits needed to encode a witness during the evaluation of a k-instructive decomposition. For pointers: $\mathcal{C}_c^{\chi}$ is specified in Section 4.8.1, while $\mathcal{C}_d^{\Delta}$, $\mathcal{C}_\ell^{\omega}$, and $\mathcal{C}^{\mathrm{ME}}$ are indexed in Chapter 4 and fully specified in Appendix D.4, Appendix D.5, and Appendix D.6, respectively.

Theorem 16 (DP-cores for the constituents of $\textsc{Reed}(s)$)**.** *Let $d, c, \ell \in \mathbb{N}$ satisfy $d \geq 1$, $c \geq 1$, and $\ell \geq 2$. Then there exist finite, internally polynomial, instructive DP-cores $\mathcal{C}_d^{\Delta}$, $\mathcal{C}_c^{\chi}$, $\mathcal{C}_\ell^{\omega}$, and $\mathcal{C}^{\mathrm{ME}}$ satisfying the following properties. Each of these decision cores is regarded here with the constant trivial invariant, so the invariant clause of coherence is vacuous and only decomposition-independent acceptance is relevant.*

1. *$\mathcal{C}_d^{\Delta}$ has bitlength $O(k\log(d+1))$, is coherent with respect to* ITD*, and decides whether the realized graph has maximum degree at least d.*

2. *$\mathcal{C}_c^{\chi}$ has bitlength $O(k\log(c+1))$, is coherent with respect to* ITD*, and decides c-colorability.*

3. *$\mathcal{C}_\ell^{\omega}$ has bitlength $O(k\log\ell)$ and, on the subclass of terms whose realized graph is simple, is coherent with respect to* ITD *and decides whether the underlying simple graph contains a clique of size at least ℓ.*

4. *$\mathcal{C}^{\mathrm{ME}}$ has bitlength $O(k^2)$, is coherent with respect to* ITD*, and decides whether the realized graph contains a multiple edge.*

Proof. For $\mathcal{C}_c^{\chi}$, Section 4.8.1 defines the core, Theorem 21 proves correctness, Proposition 8 gives decomposition-invariant acceptance, and Theorem 22 yields the stated bitlength bound. The explicit witness encoding and constructor-wise transitions of that section are finite and internally polynomial in the sense of Definitions 32 and 33; this is recorded explicitly in Lemma 31.

For $\mathcal{C}_d^{\Delta}$, Appendix D.4 proves correctness and records the bound $\beta = O((k+1)\log(d+1))$ in Theorem 27; the same appendix concludes coherence in its equivariance/coherence subsection. The witness language and local updates are explicitly bounded and constructor-wise computable, so the core is finite and internally polynomial.

For $\mathcal{C}_\ell^{\omega}$, Appendix D.5 proves correctness under the simplicity promise in Theorem 28, records the bitlength bound in Theorem 29, and derives coherence in the subsequent equivariance/coherence subsection. The same appendix gives an explicit finite witness language and polynomial-time local rules, so the core is finite and internally polynomial; only its semantic correctness and coherence are restricted to the simple-graph regime.

For $\mathcal{C}^{\mathrm{ME}}$, Appendix D.6 proves correctness and records $\beta = O(k^2)$ in Theorem 30; coherence is derived in the same appendix. Its witness language is finite and its local computations are polynomial-time by direct inspection of the explicit constructor rules. □

Remark (Multiplicity Bounds). In addition to the bitlength bounds in Theorem 16, we use that $\mathcal{C}_d^{\Delta}$ and $\mathcal{C}^{\mathrm{ME}}$ have multiplicity 1, whereas $\mathcal{C}_\ell^{\omega}$ has multiplicity at most $(\ell+1)^{k+2}$ by Theorem 29 (hence $\leq 4^{k+2}$ for $\ell = 3$), and $\mathcal{C}_c^{\chi}$ has multiplicity at most c^{k+1} by Theorem 22.

Lemma 22 (Guarded use of the clique core)**.** *Fix $s \geq 0$ and consider the Boolean combination* (8.1)*. Although Theorem 16(3) only gives semantic correctness of the*

clique core $\mathcal{C}_3^\omega$ on simple realized graphs, the same syntactic product construction used in Theorem 9 can still be formed. The guarded combination nevertheless yields a finite, internally polynomial, instructive-coherent DP-core for $\textsc{Reed}(s)$ *with the same constituent complexity parameters as in Theorem 10.*

Proof. Finiteness and internal polynomiality do not rely on constituent coherence: they follow directly from Theorem 16 and from the syntactic product construction in the proof of Theorem 9. In the present decision-only application, Condition 2b is vacuous: we use the constituent cores with their constant trivial invariants, and the Boolean combinator (8.1) depends only on the acceptance bits. It therefore remains to justify directly that the specific guarded Boolean combination (8.1) is coherent and computes the intended graph property on all realized graphs.

Let $k, k' \in \mathbb{N}$, and let $\tau_1 \in \mathsf{ITD}_k$ and $\tau_2 \in \mathsf{ITD}_{k'}$ realize isomorphic graphs. If one — equivalently, both — of $\mathcal{G}(\tau_1)$ and $\mathcal{G}(\tau_2)$ has a multiple edge, then $\mathcal{C}^{\mathrm{ME}}$ accepts both terms by Theorem 16(4), so the antecedent of (8.1) is false for both terms. Hence the guarded Boolean combination itself evaluates to the same value on τ_1 and τ_2, regardless of the raw truth value produced by the clique constituent on these non-simple inputs. If both realized graphs are simple, then Theorem 16(3) applies to the clique constituent, while items (1), (2), and (4) handle the remaining cores. In that case every constituent truth value depends only on the realized graph, hence so does the Boolean combination (8.1). Therefore the resulting combined DP-core is coherent and decides $\textsc{Reed}(s)$ on the full instructive class.

The runtime argument in Theorem 10 depends only on the constituent witness parameters and on the product-state exploration performed by the combination construction. Those ingredients are unchanged here, so the same asymptotic bound applies to this guarded combination. □

8.4.2 Applying Theorem 10

Theorem 10 (Theorem 16 in [de Oliveira Oliveira and Vadiee, 2024]) provides the inclusion-test construction for Boolean combinations of coherent DP-cores. In the present setting, Theorem 16 and Lemma 22 provide the guarded product core and the complexity hypotheses needed to use the inclusion-test argument for (8.1). The resulting consequence forms the algorithmic backbone of the Reed chapter.

Proposition 5 (Application of Theorem 10)**.** *Let $k \geq 1$. For each $s \in \{0, \ldots, k-1\}$ one can decide whether* $\textsc{GraphsTw}[k] \subseteq \textsc{Reed}(s)$ *in time $2^{2^{O(k \log(k+2))}}$. Consequently, the*

triangle-free specialization on graphs of treewidth at most k can be verified by performing at most k such inclusion tests.

Proof. Fix s and consider the Boolean combination (8.1). By Theorem 16 and Lemma 22, this guarded combination yields a finite, internally polynomial, instructive-coherent combined DP-core for $\textsc{Reed}(s)$. The underlying instructive decomposition class has arity 2 and is recognized by the bag-aware ITD automaton of Section 3.4, whose states are active bags, whose transitions are explicit, and whose complexity is $2^{O(k)}$ (Chapter 3). Hence the inclusion-test argument of Theorem 10 applies to that guarded combined core.

For $s \le k-1$ set $c := \lceil (s+3)/2 \rceil$. Then $c \le \lceil (k+2)/2 \rceil \le k+1$. The c-colorability core $\mathcal{C}_c^{\chi}$ therefore has witness bitlength $O(k \log(c+1)) = O(k \log(k+2))$, and by Theorem 22 its per-node multiplicity is at most the number of c-colorings of a bag of size at most $k+1$, namely $c^{k+1} = 2^{O(k \log(k+2))}$.

For the remaining cores, $\mathcal{C}_d^{\Delta}$ and $\mathcal{C}^{\mathrm{ME}}$ have multiplicity 1, whereas the clique core $\mathcal{C}_\ell^{\omega}$ is not generally deterministic. By Theorem 29 its multiplicity is bounded by $(\ell+1)^{k+2}$, and in particular for $\ell = 3$ we have $4^{k+2} = 2^{O(k)}$. Consequently, the constituent multiplicity envelope entering Theorem 10 is still $2^{O(k \log(k+2))}$, dominated by the $\mathcal{C}_c^{\chi}$ component. Moreover, the maximum witness bitlength among the constituents is polynomial in k (in particular, $\mathcal{C}^{\mathrm{ME}}$ has bitlength $O(k^2)$ by Theorem 16), so $\beta(k) \cdot \mu(k) = 2^{O(k \log(k+2))}$, where $\mu(k)$ denotes that constituent envelope. Plugging these bounds into the inclusion-test argument of Theorem 10 yields the claimed $2^{2^{O(k \log(k+2))}}$ running time for each fixed s. Observation 3 shows that every simple triangle-free graph of treewidth at most k with $\Delta(G) \ge k$ already satisfies the desired bound, so it is enough to test the values $s \in \{0, \ldots, k-1\}$. Hence at most k inclusion tests suffice. □

Corollary 13 (Double-exponential Reed test)**.** *For each $k \ge 1$, the triangle-free specialization can be tested on graphs of treewidth at most k in time $2^{2^{O(k \log(k+2))}}$.*

Proof. By Proposition 5, at most k inclusion tests are needed, each with running time $2^{2^{O(k \log(k+2))}}$. The additional factor k is absorbed by the same double-exponential bound. □

8.5 Premise Search and Symmetry Breaking

The inclusion tests from Proposition 5 are only effective in practice when combined with the search strategies of Chapters 6 and 7. For a fixed parameter s we isolate the premise

$$\mathcal{A}_s \;=\; \neg\textsc{MultiEdge} \wedge \neg\textsc{Clique}_3 \wedge \neg\textsc{MaxDeg}_{s+1},$$

so that $\textsc{Reed}(s)$ becomes $\mathcal{A}_s \to \textsc{Colorable}_{\lceil (s+3)/2 \rceil}$. The search procedure implemented in TreeWidzard treats $\mathcal{A}_s$ as the pruning condition in a premise triple, exactly as prescribed by Observation 2. The following lemma justifies this choice.

Lemma 23 (Subgraph-closed Reed premise)**.** *For every $s \geq 0$, the property $\mathcal{A}_s$ is closed under taking subgraphs.*

Proof. The three conjuncts defining $\mathcal{A}_s$ are all subgraph-closed. Removing vertices or edges cannot create multiple edges, cannot create a new triangle, and cannot increase the maximum degree, so each negated property is preserved by subgraphs. Their conjunction is therefore subgraph-closed, establishing the claim. □

By Lemma 23 and Observation 2, any partial term that violates $\mathcal{A}_s$ can be pruned immediately. The benchmark data reported in [de Oliveira Oliveira and Vadiee, 2024] show that this premise module is essential in practice: without it, the breadth-first exploration quickly stalls on Reed-type instances, whereas enabling it keeps the frontier small enough for the search to make progress. This effect is reflected again in the pathwidth and treewidth tables later in this section.

Symmetry breaking delivers the second essential ingredient. The DP-cores employed above carry substantial redundancy in their witness sets: two local objects that differ only by a permutation of bag vertices encode the same partial solution. The canonization techniques from Chapter 6 collapse these symmetric states, lowering both the number of stored witnesses and the branching factor seen by the search. In the Reed experiments, the combination of state canonization and premise search routinely shrinks witness sets by an order of magnitude, making the double-exponential theoretical bound of Corollary 13 tangible.

8.6 Experiments

This section summarizes the Reed case-study experiments from [de Oliveira Oliveira and Vadiee, 2024]. They provide the end-to-end case-study evidence used in this chapter and were not rerun as a separate thesis-local campaign. In those experiments, each inclusion test was realized as a property file encoding (8.1) for a fixed value of s, and the treewidth searches were executed with the `ParallelBreadthFirstSearch` strategy on k-instructive decompositions. For theoretical treewidth verification it is enough, by Observation 3, to run the cases $s \leq k-1$. The published benchmark suite also includes selected larger degree bounds, both for comparison with the pathwidth runs and to show how the search strategies behave beyond the verification-minimal range.

These published experiments demonstrate the necessity of the techniques developed in the previous chapters. Without the premise module or without canonization, the queue of partial terms explodes, and even modest admissible instances can fail to make practical progress under the stated resource budgets. With both modules enabled, the same families become tractable at the width bounds reported below. Whenever the implication fails — for example, when the conclusion is deliberately weakened to $\textsc{Colorable}_{\lceil (s+2)/2 \rceil}$ — TreeWidzard outputs a concrete counterexample decomposition, providing a certificate that feeds back into the mathematical analysis.

At the qualitative level, the first strengthening that fails under the premise $\Delta \leq 2$ is the demand for 2-colorability, with the 5-cycle C_5 as the expected obstruction. The quantitative evidence cited below is taken from the corresponding pathwidth and treewidth benchmark tables in [de Oliveira Oliveira and Vadiee, 2024].

Beyond verification, the same search infrastructure can be used to refute plausible strengthenings. Figure 8.1 reproduces two triangle-free counterexamples reported in [de Oliveira Oliveira and Vadiee, 2024] and found there by TreeWidzard at pathwidth 4. The left graph has chromatic number 4, showing that the bound of Theorem 15 cannot be strengthened to 3 colors at width 4. The right graph has maximum degree 4 and chromatic number 4, refuting the natural strengthening of (8.1) that would conclude 3-colorability under the premise $\Delta \leq 4$. For such refuted runs, TreeWidzard can export the concrete witness graph and decomposition artifacts (`.gr` graph files and `.td` decomposition files); these are rechecked in model-checking mode to reproduce `PROPERTY NOT SATISFIED` independently of the ATP controller.

The larger benchmark tables below summarize the experimental outcomes for this case study. Four configurations are compared. The baseline breadth-first search (BFS) is the reference. We also test the baseline augmented only with premise pruning

pw	Δ	BFS	BFS+Premise	ISO-BFS	ISO-BFS+Premise
2	2	2 503	149	520	38
2	3	4 814	341	945	76
2	4	9 877	617	1 843	128
3	2	24 482 279	1 916	1 031 545	141
3	3	24 908 274	9 850	1 050 960	579
3	4	5 193 467	27 720	223 920	1 451
4	2	$> 40\,982\,845^*$	29 184	$> 9\,449\,990$	486
4	3	$> 24\,741\,126^*$	1 156 954	$> 9\,903\,864$	12 375
4	4	$> 33\,517\,713^*$	2 438 694	$> 7\,945\,250$	24 494

Table 8.1: Number of explored states when testing Reed's conjecture on triangle-free graphs of bounded pathwidth. Entries prefixed by $>$ indicate that the run did not terminate and the reported number is a lower bound. The superscript * marks runs that stopped because they exceeded the cluster memory limit. When the superscript is absent, the run hit the 24-hour timeout.

(BFS+Premise), the symmetry-breaking version that canonizes states modulo bag-label relabelings (ISO-BFS), and the combination of symmetry breaking with premise pruning (ISO-BFS+Premise). Table 8.1 presents the state counts obtained on graphs of bounded pathwidth. Numbers prefixed by $>$ indicate that the run did not terminate and the reported value is a lower bound. The superscript * marks runs that were cut short because they exceeded the memory limit. When the superscript is absent, the run hit the 24-hour wall-clock limit. The corresponding wall-clock times and peak memory tables are reported in the full version of the state-canonization paper [de Oliveira Oliveira and Urmian, 2026a]; we reproduce the state counts here because they capture the qualitative effect most directly. The improvement delivered by combining state canonization with premise pruning is especially salient for pathwidth 3: for $\Delta = 2$, the raw search visits 24,482,279 states, whereas the combined configuration visits only 141 states.

Premise pruning is the key enabler for the harder instances: it prevents the search from expanding partial terms that already violate triangle-freeness, simplicity, or the degree bound, and it keeps the frontier small enough to proceed. Canonization is complementary: by collapsing states that only differ by a permutation of bag labels, it reduces both the number of stored states and the effective branching factor. Canonization alone is not uniformly superior on the smaller instances: for example, at pathwidth 2 the BFS+Premise variant can still explore fewer states than ISO-BFS. The robust message is therefore about the combination, not either ingredient in isolation. Across all reported parameters ISO-BFS+Premise yields the smallest explored state space.

For triangle-free graphs of bounded treewidth, the symmetry-breaking engine alone already exceeds the 24-hour limit for the harder configurations, so the practically relevant comparison is with the premise-enhanced variant. Table 8.2 reports the explored

tw	Δ	ISO-BFS	ISO-BFS+Premise
2	2	$> 25\,351$	57
2	3	$> 25\,960$	153
2	4	$> 27\,632$	330
3	2	$> 69\,621$	194
3	3	$> 71\,183$	1 268
3	4	$> 72\,891$	4 080
4	2	$> 71\,858$	616
4	3	$> 73\,532$	$> 4\,942$
4	4	$> 75\,426$	$> 13\,438$

Table 8.2: Number of explored states for the treewidth experiments. The symmetry-breaking engine alone (ISO-BFS) times out or runs out of memory for every configuration. Combining it with premise pruning yields dramatic reductions and terminates for $\text{tw} \leq 3$ and for $(\text{tw}, \Delta) = (4, 2)$. For $\text{tw} = 4$ with $\Delta \in \{3, 4\}$, the combined configuration still times out within the 24-hour limit.

state counts. Although Observation 3 shows that only degrees $\Delta \leq \text{tw} - 1$ can host counterexamples, larger Δ values are also reported for direct comparability with the pathwidth benchmarks. For $\text{tw} \leq 3$ (and for $(\text{tw}, \Delta) = (4, 2)$), the combination of canonization and premise pruning keeps the explored state space within a few thousand states, whereas the symmetry-only variant hits the time or memory limits on every instance. For $\text{tw} = 4$ with $\Delta \in \{3, 4\}$, even the combined configuration still reaches the 24-hour limit, but the explored state space remains vastly smaller than for ISO-BFS alone. The corresponding wall-clock times and peak memory consumption are reported in the full version of the state-canonization paper [de Oliveira Oliveira and Urmian, 2026a].

The multiplicity table from the state-canonization project is reproduced here from [de Oliveira Oliveira and Urmian, 2026a; de Oliveira Oliveira and Vadiee, 2024]: the combined canonization-and-pruning engine (ISO-BFS+Premise) limits the witness multiplicities to $(14, 1, 18, 1)$ for pathwidth 3 and $(41, 1, 30, 1)$ for pathwidth 4 with $\Delta = 2$, whereas the raw breadth-first search regularly pushes the chromatic-number component above 40 before running out of memory. The qualitative effect is the same on the treewidth side: once both optimizations are enabled, the observed multiplicities stay in the same regime (around $(41, 1, \approx 30, 1)$) while the explored state space drops dramatically. These figures help explain why the premise-aware, canonized configuration exhibits such a dramatic reduction in explored states.

The largest published positive verification run in [de Oliveira Oliveira and Urmian, 2026a; de Oliveira Oliveira and Vadiee, 2024] concerns triangle-free graphs of pathwidth at most 5 and maximum degree 3. With both canonization and premise pruning enabled, the search enumerates 746,187 states, terminates in roughly 41 hours, and keeps the

witness multiplicities at $(122, 1, 52, 1)$, producing a fully automated confirmation of the triangle-free Reed bound on that class. Without canonization or premise pruning, the same configuration is prohibitively large.

Taken together as a published case study integrated into the thesis, these results show that the width-bounded ATP pipeline can verify triangle-free Reed-style statements on bounded-width classes, produce explicit counterexamples to natural strengthenings, and benefit decisively from combining premise pruning with canonization. They do not claim anything about the unbounded Reed conjecture beyond the width classes tested here, nor do they show that canonization alone is a uniformly dominant optimization.

8.7 Summary

Overall, the experiments reinforce the theoretical message of this chapter. The DP-core formulation supplies the algebraic backbone, Theorem 10 guarantees decidability with explicit complexity bounds, and the combination of premise search with state canonization makes the exploration feasible in practice. The counterexample workflow further demonstrates how width-based ATP produces machine-checkable certificates whenever the implication fails within the chosen width bound. The next part addresses systematic correctness and empirical validation before the thesis concludes with its overall synthesis and future directions.

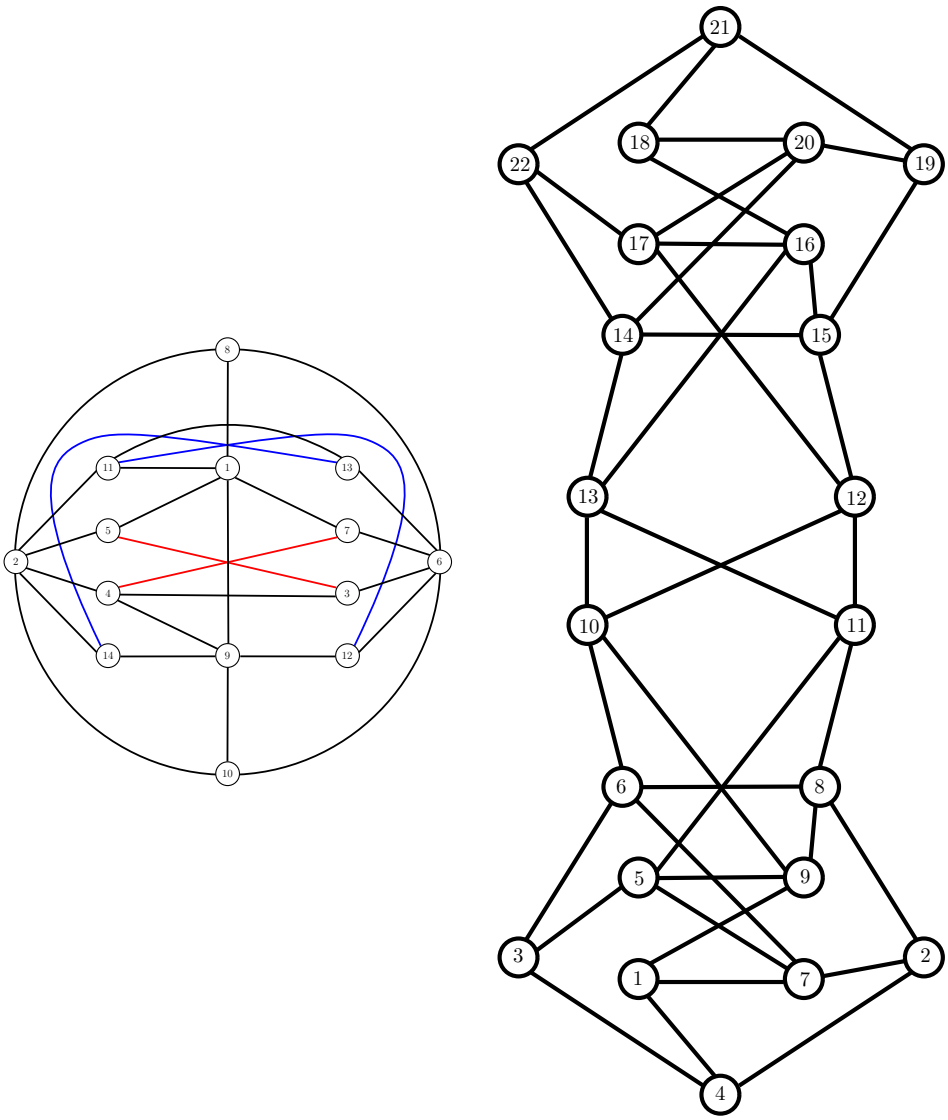


Figure 8.1: Two counterexamples found at pathwidth 4; the same counterexamples also appear in [de Oliveira Oliveira and Vadiee, 2024]. Left: a triangle-free graph of chromatic number 4. Right: a triangle-free graph of maximum degree 4 and chromatic number 4.

Part V

Correctness and Validation

Chapter 9

Systematic Correctness

Width-based automated theorem proving relies on DP-cores that express graph properties and invariants. To use DP-cores as deductive building blocks, we must be confident that their local transition rules implement the intended global meaning. In practice, correctness arguments for such dynamic programs are often ad hoc: bespoke proofs tailored to each property, with limited reuse and limited guidance for extending the catalog. In this chapter, we develop a systematic proof framework for treelike DP-cores that unifies and extends the per-core arguments from the published work. Rather than proving correctness from scratch for each core, we isolate a reusable local proof schema. Global correctness then follows once the core-specific denotation is fixed, the local soundness-and-completeness law is proved, and any required cleaning or canonization lemmas are discharged.

Our proof framework aligns two complementary views of a DP-core. On the operational side, a core induces a dynamic semantics $\Gamma[\mathcal{D}, k](\tau)$ obtained by evaluating the transition system bottom-up on an instructive term $\tau \in \mathsf{ITD}_k$, with cleaning (canonization) applied at each constructor. On the semantic side, we define a denotational meaning $\mathrm{Sem}_{\mathcal{D}}(\tau)$ in terms of the intended global combinatorial objects (colorings, covers, paths, …) and how they project to the current bag interface. The central claim is that, once the local transition rules are validated against this intended meaning via the local soundness-and-completeness law (LSC), the two semantics coincide. This identification is what turns the catalog of Chapter 4 into a collection of reliable, reusable building blocks: it yields the appropriate graph-level decomposition-independence statement for each core type (root acceptance invariance for decision cores, and root acceptance together with invariant agreement for invariant-valued cores), it exposes the witness parameters that control the generic evaluation bounds used later, and it makes precise why cleaning preserves both the denotation and the root verdict.

Formally, Theorem 17 proves $\Gamma[\mathcal{D}, k](\tau) = \mathrm{Sem}_{\mathcal{D}}(\tau)$ for all $\tau \in \mathsf{ITD}_k$ under (LSC). The framework then develops the standard consequences of this identification: coherence, symmetry on bags via equivariance, a generic runtime bound in terms of the per-core witness parameters $\beta_{\mathcal{D}}$ and $\mu_{\mathcal{D}}$ from Chapter 4 (recalled in §9.1.5), and the fact that cleaning is semantically harmless because it is built into the semantic operators via F_a.

Section 9.1 develops the abstract proof framework and can be read independently of the later instantiations. Section 9.2 illustrates the method with the chromatic decision core as a worked example. Detailed instantiations for optimization cores (vertex cover, maximum independent set, vertex counting), supplementary framework notes, and boundary-equivalence lower-bound material are collected in Appendix D. Section 9.3 closes with navigation pointers and brief guidance for extending the method beyond the catalog. We now formalize the framework abstractly and isolate the single local proof rule that all later instantiations will verify.

Notation crosswalk with Chapter 4. Chapter 4 introduces the same local semantic idea in a lighter catalog style: some concrete entries use per-core predicate names such as $\mathsf{P}^{\chi}_{c,k}$, suppress explicit bag parameters when harmless, and present the chromatic witnesses as partitions rather than labeled total maps. In this chapter and Appendix D we standardize the generic name $\mathsf{Interp}_{\mathcal{D}}$ for the local interpretation predicate, make bag-awareness explicit when needed, and use the total-map chromatic presentation because it interacts more cleanly with the algebraic proof framework. Lemma 28 later shows that the two chromatic witness presentations coincide extensionally.

In this chapter, we strengthen the abstract Chapter 4 interface by assuming that for each width k, the cleaning map $\mathtt{Clean}_k : \mathcal{P}_{\mathrm{fin}}(W_k) \to \mathcal{P}_{\mathrm{fin}}(W_k)$ is *shrinking* and *idempotent*. Chapter 4 treated cleaning as an arbitrary total operation on finite witness sets because that sufficed for fixed-term evaluation and complexity accounting; the algebraic semantics developed here additionally uses these two laws to build canonical witness-set algebras and homomorphic dynamic semantics. No further structural properties of cleaning are required in this chapter beyond shrinking and idempotence.

9.1 A Systematic Proof Framework

The goal of this section is to isolate the common structure behind all DP-core correctness proofs. The idea is to view each core as an algebra on witness sets, relate it to a family of *global* combinatorial objects (colorings, covers, paths, …), and capture the intended meaning of a witness by a local judgment. A single local proof rule (INT) per constructor

then yields global correctness via a master theorem.

Proof recipe for DP-core correctness. Fix a width bound k and a DP-core instance $\mathcal{D}[k]$. A correctness proof starts by making explicit what the witnesses are intended to summarize. Concretely, one chooses, for every realized graph G, a *finite* set $\mathcal{X}(G)$ of global combinatorial objects appropriate for the target problem (e.g. proper colorings, vertex covers, or path subgraphs). For an instructive term σ with top bag $B(\sigma)$, one then defines a projection map $\pi_{B(\sigma)} : \mathcal{X}(\mathcal{G}(\sigma)) \to W_k$ that extracts the local summary of a global object at the interface exposed by the bag. Although W_k is the global witness universe of the DP-core instance, each bag B determines a well-typed slice $W_k(B) \subseteq W_k$ of witnesses whose support/domain is compatible with B; all denotational sets and transition outputs below are understood to lie in the appropriate slice $W_k(B(\sigma))$. This projection induces the local interpretation predicate $\mathsf{Interp}_{\mathcal{D}}(\sigma, w)$, asserting that w correctly summarizes (via $\pi_{B(\sigma)}$) at least one object in $\mathcal{X}(\mathcal{G}(\sigma))$.

With these choices fixed, the core proof obligation is local and constructor-wise. For each constructor a and tuple of child terms $(\sigma_i)_i$, one proves the equivalence that the parent witness w is semantically valid exactly when it can be obtained by applying the transition $\widehat{a}$ to compatible child witnesses. This is the (INT) rule:

$$\mathsf{Interp}_{\mathcal{D}}\big(a(\vec{\sigma}), w\big) \iff \exists \vec{w} \, \Big(\bigwedge_i \mathsf{Interp}_{\mathcal{D}}(\sigma_i, w_i) \ \wedge \ w \in \widehat{a}(\vec{w}) \Big).$$

Whenever a parent term $\tau = a(\sigma_1, \ldots, \sigma_r)$ is fixed, we use the shorthand $\widehat{a}(\vec{w}) := \Delta_{k,a}(B(\tau), \vec{w})$ and $\mathsf{F}_a := \mathsf{F}_{a,B(\tau)}$. We refer to this equivalence as (INT) (Equation (INT)). Intuitively, (INT) states that the transition system is neither too weak (it fails to produce some semantically meaningful summary) nor too strong (it produces a summary lacking a genuine semantic explanation).

When cleaning is trivial on denotational sets (as in many decision cores), (INT) is often the most direct route to the local soundness-and-completeness law (LSC). When cleaning performs nontrivial canonization (for instance, retaining only extremal values per signature), one typically establishes (LSC) at the set level directly, together with an explicit argument that canonization preserves denotation and does not remove any witness needed for root acceptance or invariant computation. Once (LSC) holds for all constructors, Theorem 17 yields $\Gamma[\mathcal{D}, k](\sigma) = \mathrm{Sem}_{\mathcal{D}}(\sigma)$ for all $\sigma \in \mathsf{ITD}_k$. Moreover, when $\mathsf{Interp}_{\mathcal{D}}$ characterizes denotation, Theorem 19 identifies $\mathrm{Sem}_{\mathcal{D}}(\sigma)$ with the set of witnesses satisfying $\mathsf{Interp}_{\mathcal{D}}(\sigma, \cdot)$, enabling a direct semantic reading of the computed witness sets.

PROOF ROADMAP

1. Choose the global-object family $\mathcal{X}(G)$ and the bag projection π_B.
2. Define the local interpretation predicate $\mathsf{Interp}_{\mathcal{D}}$ induced by these global objects.
3. Prove (INT) constructor by constructor, or establish (LSC) directly when cleaning performs nontrivial canonization.
4. Invoke Theorems 17 and 19 to conclude dynamic semantics equals denotational semantics and to read off the intended graph property or invariant at the root.

Finally, if one wishes to extract explicit proof evidence, the same inductive structure supports certificate extraction by recording, at each constructor application, which child witnesses justify the chosen parent witness (Appendix D.11.2).

Terms, graphs, and bags. Fix a width-k instructive alphabet Σ_k with the standard operators

$$\texttt{leaf}, \quad \texttt{intro\_}v, \quad \texttt{forget\_}v, \quad \texttt{intro\_}u\texttt{\_}v, \quad \texttt{join}.$$

Write $\mathcal{T}_{\Sigma_k}$ for the term algebra over Σ_k, and write $\mathrm{Terms}(\Sigma_k)$ for its carrier of ground terms. Let $\mathsf{ITD}_k \subseteq \mathrm{Terms}(\Sigma_k)$ be the language of k-instructive decompositions from Section 3.4. Whenever $\tau \in \mathsf{ITD}_k$, let $G := \mathcal{G}(\tau)$ be its realized graph, $B := B(\tau) \subseteq [k+1]$ its top bag, and $\theta[\tau] : B \to V(G)$ the canonical labeling as defined in Section 3.4. We use the notation $\mathcal{G}(\tau)$, $B(\tau)$, and $\theta[[]\tau]$ only for instructive terms in ITD_k.

Lemma 24 (Instructive terms are closed under subterms)**.** *If $\tau \in \mathsf{ITD}_k$ and $\tau = a(\tau_1, \ldots, \tau_r)$, then $\tau_i \in \mathsf{ITD}_k$ for every i. In particular, every subterm of an instructive term is instructive.*

Proof. Immediate from the inductive definition of ITD_k as a term language over Σ_k with local side conditions. □

Define term height by

$$\textsc{Height}(\texttt{leaf}) = 0, \qquad \textsc{Height}\big(a(\tau_1, \ldots, \tau_r)\big) = 1 + \max_i \textsc{Height}(\tau_i).$$

Root bags. We do not require the root bag to be empty. Unless stated otherwise, all *semantic* statements below quantify over instructive terms $\tau \in \mathsf{ITD}_k$, since the realized-graph semantics $\mathcal{G}(\tau)$ and bag interface $B(\tau)$ are defined there. Root-closed terms with $B(\tau) = \emptyset$ are a convenient normal form when discussing plain (unboundaried) graphs:

any instructive term can be made root-closed by appending forget constructors at the root, which does not change the realized graph (only the interface exposed at the root).

Remark 14 (Generality beyond treewidth). Although we phrase the framework in terms of instructive tree decompositions and realized graphs, the proofs in §§9.1–9.1.6 depend only on the fact that terms form an initial Σ-algebra, that each constructor a has a transition $\widehat{a}$ on witnesses, and that we have chosen global objects $\mathcal{X}(-)$ with a projection π_B. The same proof calculus therefore applies verbatim to any dynamic program that can be expressed as "evaluate a term bottom-up by combining local states and then canonically cleaning them", regardless of whether the underlying objects are graphs or not.

9.1.1 Dynamic semantics as the unique homomorphism

A DP-core instance is

$$\mathcal{D}[k] = (\Sigma_k, W_k, \texttt{Final}_k, \Delta_k, \texttt{Clean}_k, \texttt{Inv}_k)$$

with witness-level transitions $\widehat{a} : W_k^r \to \mathcal{P}_{\text{fin}}(W_k)$ for $a \in \Sigma_k$ of arity r (this is the same object denoted $\Delta_{k,a}$ in Chapter 4). As in Chapter 4 (*Lifted transitions*), we lift each witness-level transition to an operator on witness sets and incorporate cleaning. In the present chapter we denote this lifted operation by

$$\mathsf{F}_a(S_1, \dots, S_r) := \texttt{Clean}_k\left(\bigcup_{(w_1,\dots,w_r)\in S_1\times\cdots\times S_r} \widehat{a}(w_1, \dots, w_r) \right).$$

Equivalently, F_a is the cleaned set-lifting $\mathcal{D}[k].\hat{a}$ from Chapter 4, written with explicit dependence on a to emphasize the underlying constructor family before bag annotation is made explicit.

Intuitively, F_a combines child witness sets and then removes redundant states via $\texttt{Clean}_k$. We now package this canonization algebraically in a bag-aware form.

Bag-aware convention. In Chapter 4, transition maps and finality are defined relative to the current bag $B \subseteq [k+1]$. In this section we keep that dependence explicit in local operators. Equivalently, one may pass to a bag-annotated instructive alphabet: for each instructive term $\tau \in \mathsf{ITD}_k$, let $\overline{\tau}$ be the term obtained by annotating each constructor occurrence with its validator-computed root bag. Whenever homomorphisms or fly automata are used below, this annotated viewpoint is intended. Accordingly,

Definitions 53 and 54 are formally stated over the bag-annotated signature $\overline{\Sigma}_k$, while Σ_k continues to denote the underlying instructive signature. Later formulas may still write an unannotated constructor symbol $a \in \Sigma_k$ when the parent instructive term $\tau = a(\vec{\sigma})$ is fixed; this is shorthand for the unique bag-annotated occurrence $(a, B(\tau))$ determined by the validator. For example, if $\tau = \texttt{intro\_}v(\sigma)$ and the validator computes $B(\tau) = B(\sigma) \cup \{v\}$, then the relevant local operator is indexed by $(\texttt{intro\_}v, B(\tau))$. Likewise, we write $\texttt{Final}_k$ for the finality family and $\texttt{Final}_k(B, w)$ when the current bag needs to be shown explicitly.

Definition 53 (Cleaning as canonization, canonical-set algebra)**.** *Assume* $\texttt{Clean}_k : \mathcal{P}_{\mathrm{fin}}(W_k) \to \mathcal{P}_{\mathrm{fin}}(W_k)$ *is idempotent and shrinking (for all* $S \in \mathcal{P}_{\mathrm{fin}}(W_k)$*,* $\texttt{Clean}_k(S) \subseteq S$*), i.e. a canonization operator on finite witness sets. For each admissible bag-annotated constructor* (a, B) *of arity* r*, define*

$$\mathsf{F}_{a,B}(S_1, \dots, S_r) := \texttt{Clean}_k\left(\bigcup_{(w_1,\dots,w_r) \in S_1 \times \cdots \times S_r} \Delta_{k,a}(B, w_1, \dots, w_r) \right).$$

Let

$$\mathrm{Cl}(W_k) := \{S \in \mathcal{P}_{\mathrm{fin}}(W_k) : \texttt{Clean}_k(S) = S\}.$$

Then each $\mathsf{F}_{a,B}$ *restricts to* $\mathrm{Cl}(W_k)$*. Define*

$$\mathcal{A}_{\mathcal{D}[k]} := \big(\mathrm{Cl}(W_k) \ , \ \mathsf{F}_{a,B} \big)_{(a,B) \in \overline{\Sigma}_k}.$$

Then $\mathcal{A}_{\mathcal{D}[k]}$ *is an* $\overline{\Sigma}_k$*-algebra.*

This turns the DP-core at width k into a concrete bag-aware algebra. The dynamic semantics will be the unique homomorphism from the bag-annotated term algebra into this canonical-set algebra.

Definition 54 (Dynamic semantics (homomorphic characterization))**.** *Let* $\mathcal{T}_{\overline{\Sigma}_k}$ *be the term algebra over the bag-annotated alphabet. The dynamic semantics is the unique homomorphism*

$$[\![-]\!]_{\mathcal{D}[k]} : \mathcal{T}_{\overline{\Sigma}_k} \to \mathcal{A}_{\mathcal{D}[k]}$$

with constructor clauses

$$[\![(a, B)(t_1, \dots, t_r)]\!]_{\mathcal{D}[k]} = \mathsf{F}_{a,B}\big([\![t_1]\!]_{\mathcal{D}[k]}, \dots, [\![t_r]\!]_{\mathcal{D}[k]}\big).$$

Equivalently, for every instructive term $\tau \in \mathsf{ITD}_k$*,*

$$[\![\overline{\tau}]\!]_{\mathcal{D}[k]} = \Gamma[\mathcal{D}, k](\tau),$$

and recursively

$$\Gamma[\mathcal{D},k]\big(a(\tau_1,\dots,\tau_r)\big) = \mathsf{F}_{a,B(a(\tau_1,\dots,\tau_r))}\big(\Gamma[\mathcal{D},k](\tau_1),\dots,\Gamma[\mathcal{D},k](\tau_r)\big).$$

Proposition 6 (Agreement with recursive dynamization)**.** *For every instructive term* $\tau \in \mathsf{ITD}_k$*,*

$$[\![\overline{\tau}]\!]_{\mathcal{D}[k]} = \Gamma[\mathcal{D},k](\tau).$$

Proof. By structural induction on τ, unfolding the bag-aware recursive clauses for $\Gamma[\mathcal{D},k]$ and the defining equation for $[\![-]\!]_{\mathcal{D}[k]}$ in parallel. □

Deterministic fly-automaton convention. In this proposition, a deterministic fly automaton is a symbolic bottom-up recognizer whose states are effectively represented and whose state space need not be finite. Formally, it consists of a ranked alphabet, a state domain Q, computable transition maps $\delta_a : Q^r \to Q$ for symbols a of arity r, and a decidable final predicate $Q_f \subseteq Q$. Its unique run on a ground term labels each node bottom-up by applying the corresponding δ_a, and $L(\mathcal{F})$ denotes the set of ground terms whose root state satisfies Q_f. We use only this deterministic run semantics here, not the closure or decidability facts for finite tree automata.

Proposition 7 (DP-cores induce deterministic fly automata (bag-aware form))**.** *Fix* k *and a DP-core instance* $\mathcal{D}[k]$*. Assume that* $\mathrm{Cl}(W_k)$ *is effectively given, that each bag-aware lifted transition* $\mathsf{F}_{a,B}$ *is computable, and that* $\mathtt{Final}_k$ *is decidable. Define a deterministic fly automaton* $\mathcal{F}_{\mathcal{D}[k]}$ *over the bag-annotated alphabet* $\overline{\Sigma}_k$ *by:*

- *state space*

$$Q := \{\bot\} \;\cup\; \big(\mathcal{P}([k+1]) \times \mathrm{Cl}(W_k)\big),$$

  where $\bot$ *is a rejecting sink for inadmissible bag evolutions;*

- *transitions on non-sink states:*

$$\delta_{\mathit{leaf}}() := (\emptyset, \mathsf{F}_{\mathit{leaf},\emptyset}()),$$

$$\delta_{\mathit{intro\_u}}(B,S) := \begin{cases} (B \cup \{u\},\ \mathsf{F}_{\mathit{intro\_u},\,B\cup\{u\}}(S)), & u \notin B, \\ \bot, & u \in B, \end{cases}$$

$$\delta_{\mathit{forget\_u}}(B,S) := \begin{cases} (B \setminus \{u\},\ \mathsf{F}_{\mathit{forget\_u},\,B\setminus\{u\}}(S)), & u \in B, \\ \bot, & u \notin B, \end{cases}$$

$$\delta_{\textit{intro\_u\_v}}(B,S) := \begin{cases} (B,\ \mathsf{F}_{\textit{intro\_u\_v},B}(S)), & u,v \in B,\ u \neq v, \\ \bot, & \textit{otherwise}, \end{cases}$$

$$\delta_{\textit{join}}\big((B,S_1),(B,S_2)\big) := (B, \mathsf{F}_{\textit{join},B}(S_1,S_2)),$$

and $\delta_{\textit{join}}(q_1,q_2) := \bot$ *when one child is* $\bot$ *or bag components disagree; for each unary constructor* $a \in \{\textit{intro\_u}, \textit{forget\_u}, \textit{intro\_u\_v}\}$*, set* $\delta_a(\bot) := \bot$*;*

- *final states*

$$Q_f := \{(B,S) \in \mathcal{P}([k+1]) \times \mathrm{Cl}(W_k) : \exists w \in S,\ \mathtt{Final}_k(B,w) = 1\}.$$

Then for every instructive term $\tau \in \mathsf{ITD}_k$*, the unique bottom-up run of* $\mathcal{F}_{\mathcal{D}[k]}$ *on the bag-annotated term* $\overline{\tau}$ *labels each position* p *by*

$$\big(B(\tau|_p), \Gamma[\mathcal{D},k](\tau|_p)\big).$$

In particular, the root state is

$$\big(B(\tau), \Gamma[\mathcal{D},k](\tau)\big),$$

and

$$\tau \in \mathsf{Acc}(\mathcal{D}[k]) \iff \overline{\tau} \in L(\mathcal{F}_{\mathcal{D}[k]}).$$

Proof. We prove by structural induction on instructive subterms σ of τ the following stronger claim:

1. the run of $\mathcal{F}_{\mathcal{D}[k]}$ on the bag-annotated term $\overline{\sigma}$ never reaches the sink state $\bot$, and
2. the root of $\overline{\sigma}$ is labeled by
$$\big(B(\sigma), \Gamma[\mathcal{D},k](\sigma)\big).$$

For the leaf term, instructive validity gives $B(\mathtt{leaf}) = \emptyset$, and the automaton transition is

$$\delta_{\mathtt{leaf}}() = (\emptyset, \mathsf{F}_{\mathtt{leaf},\emptyset}()).$$

By definition of the bag-aware dynamization, this is exactly

$$(\emptyset, \Gamma[\mathcal{D},k](\mathtt{leaf})),$$

so the sink is not reached.

Assume now that the claim holds for the immediate instructive subterms. Write the

already computed child labels as

$$\big(B(\sigma_i), \Gamma[\mathcal{D}, k](\sigma_i)\big),$$

which are non-sink by the induction hypothesis.

If $\sigma = \texttt{intro\_}u(\sigma_1)$, instructive validity gives $u \notin B(\sigma_1)$ and $B(\sigma) = B(\sigma_1) \cup \{u\}$. Hence the non-sink branch of $\delta_{\texttt{intro\_}u}$ applies and yields

$$\Big(B(\sigma), \mathsf{F}_{\texttt{intro\_}u,\, B(\sigma)}\big(\Gamma[\mathcal{D}, k](\sigma_1)\big)\Big) = \big(B(\sigma), \Gamma[\mathcal{D}, k](\sigma)\big).$$

If $\sigma = \texttt{forget\_}u(\sigma_1)$, instructive validity gives $u \in B(\sigma_1)$ and $B(\sigma) = B(\sigma_1) \setminus \{u\}$. Therefore the non-sink branch of $\delta_{\texttt{forget\_}u}$ applies and returns

$$\Big(B(\sigma), \mathsf{F}_{\texttt{forget\_}u,\, B(\sigma)}\big(\Gamma[\mathcal{D}, k](\sigma_1)\big)\Big) = \big(B(\sigma), \Gamma[\mathcal{D}, k](\sigma)\big).$$

If $\sigma = \texttt{intro\_}u\texttt{\_}v(\sigma_1)$, instructive validity gives $u, v \in B(\sigma_1)$ with $u \neq v$ and $B(\sigma) = B(\sigma_1)$. Thus the non-sink branch of $\delta_{\texttt{intro\_}u\texttt{\_}v}$ applies and yields

$$\Big(B(\sigma), \mathsf{F}_{\texttt{intro\_}u\texttt{\_}v,\, B(\sigma)}\big(\Gamma[\mathcal{D}, k](\sigma_1)\big)\Big) = \big(B(\sigma), \Gamma[\mathcal{D}, k](\sigma)\big).$$

Finally, if $\sigma = \texttt{join}(\sigma_1, \sigma_2)$, instructive validity gives $B(\sigma_1) = B(\sigma_2) = B(\sigma)$. Since both child states are already non-sink and have the same bag component, the sink clause of $\delta_{\texttt{join}}$ is not triggered. The automaton therefore returns

$$\Big(B(\sigma), \mathsf{F}_{\texttt{join},\, B(\sigma)}\big(\Gamma[\mathcal{D}, k](\sigma_1), \Gamma[\mathcal{D}, k](\sigma_2)\big)\Big) = \big(B(\sigma), \Gamma[\mathcal{D}, k](\sigma)\big),$$

again by the recursive definition of $\Gamma[\mathcal{D}, k]$.

This proves both admissibility (the sink is never reached on validated instructive inputs) and the displayed state formula. Hence each position p of $\overline{\tau}$ is labeled by

$$\big(B(\tau|_p), \Gamma[\mathcal{D}, k](\tau|_p)\big).$$

At the root, acceptance in the automaton is

$$\exists w \in \Gamma[\mathcal{D}, k](\tau) \text{ with } \texttt{Final}_k(B(\tau), w) = 1,$$

which is exactly the Chapter 4 acceptance condition for DP-cores. □

9.1.2 Denotational semantics and the local law

For the target property/invariant, let $\mathcal{X}(G)$ be a finite set of global objects (colorings, covers, paths, etc.). The *bag projection*

$$\pi_B : \mathcal{X}(G) \to W_k$$

returns the canonical witness on the top bag. For an instructive term $\tau \in \mathsf{ITD}_k$ with realized graph $G := \mathcal{G}(\tau)$ and top bag $B := B(\tau)$, set

$$\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\tau) := \{\ \pi_B(x)\ :\ x \in \mathcal{X}(G)\ \} \in \mathcal{P}_{\mathrm{fin}}(W_k), \qquad \mathrm{Sem}_{\mathcal{D}}(\tau) := \mathtt{Clean}_k\big(\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\tau)\big).$$

Since graphs in our setting are finite and we assume that the chosen object family $\mathcal{X}(G)$ is finite for each realized graph G, the pre-cleaning set $\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\tau)$ is finite, so the application of $\mathtt{Clean}_k$ is well-defined. We treat $\mathrm{Sem}_{\mathcal{D}}$ as the denotation throughout. $\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}$ is an auxiliary pre-cleaning set used only to make the role of canonization explicit.

Definition 55 (Local soundness and completeness (LSC))**.** *For each $a \in \Sigma_k$ of arity r and all $\tau_1, \dots, \tau_r \in \mathsf{ITD}_k$ such that $a(\tau_1, \dots, \tau_r) \in \mathsf{ITD}_k$,*

$$\mathrm{Sem}_{\mathcal{D}}\big(a(\tau_1, \dots, \tau_r)\big) = \mathsf{F}_{a, B(a(\tau_1,\dots,\tau_r))}(\mathrm{Sem}_{\mathcal{D}}(\tau_1), \dots, \mathrm{Sem}_{\mathcal{D}}(\tau_r))\,. \tag{LSC}$$

Thus (LSC) states that the denotational semantics is a homomorphism for the same algebra structure $(\mathsf{F}_a)_a$ that drives the dynamic semantics.

Lemma 25 (Denotational sets are canonical)**.** *For every instructive term $\tau \in \mathsf{ITD}_k$ we have*

$$\mathtt{Clean}_k\big(\mathrm{Sem}_{\mathcal{D}}(\tau)\big) = \mathrm{Sem}_{\mathcal{D}}(\tau),$$

i.e. denotational witness sets are canonical.

Proof. By definition, $\mathrm{Sem}_{\mathcal{D}}(\tau) = \mathtt{Clean}_k\big(\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\tau)\big)$. Since $\mathtt{Clean}_k$ is idempotent, applying it again does not change the set. □

9.1.3 Master equivalence theorems

This subsection records the core equivalence theorem connecting the dynamic and denotational semantics.

Theorem 17 (Dynamic = denotational)**.** *Assume* (LSC). *Then for every instructive*

term $\tau \in \mathsf{ITD}_k$,

$$\Gamma[\mathcal{D}, k](\tau) = \mathrm{Sem}_{\mathcal{D}}(\tau).$$

Proof. We proceed by induction on $h = \textsc{Height}(\tau)$. For $h = 0$, τ is a nullary constructor a, and both $\Gamma[\mathcal{D}, k](\tau)$ and $\mathrm{Sem}_{\mathcal{D}}(\tau)$ are equal to $\mathsf{F}_{a,B(\tau)}()$ by Definition 54 and (LSC). For the inductive step, write $\tau = a(\tau_1, \ldots, \tau_r)$ with each $\textsc{Height}(\tau_i) < h$. Since $\tau \in \mathsf{ITD}_k$, all subterms τ_i are instructive as well by Lemma 24. By the induction hypothesis, $\Gamma[\mathcal{D}, k](\tau_i) = \mathrm{Sem}_{\mathcal{D}}(\tau_i)$ for all i, so

$$\Gamma[\mathcal{D}, k](\tau) = \mathsf{F}_{a,B(\tau)}\big(\mathrm{Sem}_{\mathcal{D}}(\tau_1), \ldots, \mathrm{Sem}_{\mathcal{D}}(\tau_r)\big) \overset{\text{(LSC)}}{=} \mathrm{Sem}_{\mathcal{D}}(\tau).$$

□

Remark 15 (Main proof obligation). Theorem 17 is an instance of the standard initiality/structural-induction principle for term algebras: once the denotation is shown to satisfy (LSC), it is forced to coincide with the dynamic semantics. Thus, for a concrete DP-core the substantive correctness proof obligation is to establish (LSC) (or (INT) plus a lemma showing that cleaning preserves the intended meaning) constructor-wise.

Deferred supplementary framework notes. For readability, several optional or auxiliary framework components are deferred to Appendix D.11: an abstract-interpretation reformulation of the local law, a derivation/certificate proof-theoretic view, and a generic dominance-cleaning template for optimization cores.

9.1.4 Equivariance and coherence

Let S_{k+1} act on labels and witnesses by permutation. On instructive terms, $\rho \cdot \tau$ denotes the term obtained from τ by relabeling every constructor parameter u to $\rho(u)$.

Equivariance expresses that the DP-core is insensitive to how bag labels are permuted: relabeling and then running the core is the same as running the core and then relabeling its witnesses.

Definition 56 (Equivariance)**.** *A core $\mathcal{D}$ is equivariant if for every permutation $\rho \in S_{k+1}$, every admissible bag $B \subseteq [k+1]$, every constructor symbol a, and every witness tuple $\vec{w}$, we have*

$$\rho \cdot \Delta_{k,a}(B, \vec{w}) = \Delta_{k,\rho a}(\rho(B), \rho \cdot \vec{w}),$$

where ρa is obtained from a by relabeling its label parameters via ρ. Moreover,

$$\begin{aligned}\texttt{Clean}_k(\rho \cdot S) &= \rho \cdot \texttt{Clean}_k(S),\\ \texttt{Final}_k(\rho(B), \rho \cdot w) &= \texttt{Final}_k(B, w),\\ \texttt{Inv}_k(\rho \cdot S) &= \texttt{Inv}_k(S).\end{aligned}$$

Remark 16 (Relation to Chapter 6). The symmetry-breaking results of Chapter 6 are formulated in terms of witness actions for partial bag relabelings. For the concrete cores treated in this chapter and Appendix D, the relabeling maps described in the core-specific equivariance paragraphs are exactly the witness-level renamings used there. Accordingly, those paragraphs are the places where the abstract symmetry assumptions of Chapter 6 are instantiated for the catalog cores used later in the thesis.

Lemma 26 (Naturality)**.** *Assume $\mathcal{D}$ is equivariant. Assume moreover that the chosen global-object semantics is compatible with relabeling in the following sense: for every instructive term $\tau \in \mathsf{ITD}_k$ and every $\rho \in S_{k+1}$ there is a canonical transport bijection*

$$\rho_* : \mathcal{X}(\mathcal{G}(\tau)) \to \mathcal{X}(\mathcal{G}(\rho \cdot \tau))$$

such that for every $x \in \mathcal{X}(\mathcal{G}(\tau))$,

$$\pi_{B(\rho\cdot\tau)}(\rho_* x) = \rho \cdot \pi_{B(\tau)}(x).$$

Then for every $\tau \in \mathsf{ITD}_k$ we have

$$\big(B(\rho \cdot \tau),\, \Gamma[\mathcal{D}, k](\rho \cdot \tau)\big) = \big(\rho(B(\tau)),\, \rho \cdot \Gamma[\mathcal{D}, k](\tau)\big),$$

and likewise

$$\big(B(\rho \cdot \tau),\, \mathrm{Sem}_{\mathcal{D}}(\rho \cdot \tau)\big) = \big(\rho(B(\tau)),\, \rho \cdot \mathrm{Sem}_{\mathcal{D}}(\tau)\big).$$

Proof. Let ρ_T denote the action of ρ on instructive terms, and define the bag-aware state action

$$\rho_D(B, S) := (\rho(B), \rho \cdot S).$$

By Definition 56, relabeling commutes with each bag-aware transition $\mathsf{F}_{a,B}$, so ρ_D is an automorphism of the bag-aware transition system from Proposition 7. Hence, by structural induction on subterms, for every instructive term τ,

$$\big(B(\rho \cdot \tau), \Gamma[\mathcal{D}, k](\rho \cdot \tau)\big) = \rho_D\big(B(\tau), \Gamma[\mathcal{D}, k](\tau)\big) = \big(\rho(B(\tau)), \rho \cdot \Gamma[\mathcal{D}, k](\tau)\big).$$

For denotational semantics, fix $\tau \in \mathsf{ITD}_k$ and write $B := B(\tau)$. Then $B(\rho \cdot \tau) = \rho(B)$

and

$$\begin{aligned}\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\rho\cdot\tau) &= \{\pi_{B(\rho\cdot\tau)}(y) : y \in \mathcal{X}(\mathcal{G}(\rho\cdot\tau))\}\\ &= \{\pi_{B(\rho\cdot\tau)}(\rho_* x) : x \in \mathcal{X}(\mathcal{G}(\tau))\}\\ &= \{\rho\cdot\pi_{B(\tau)}(x) : x \in \mathcal{X}(\mathcal{G}(\tau))\}\\ &= \rho\cdot\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\tau).\end{aligned}$$

Applying $\texttt{Clean}_k$ and using its equivariance yields

$$\mathrm{Sem}_{\mathcal{D}}(\rho\cdot\tau) = \rho\cdot\mathrm{Sem}_{\mathcal{D}}(\tau),$$

which gives the second displayed equality in the lemma. □

Corollary 14 (Coherence, split form)**.** *Let $\mathcal{D}$ be a treelike DP-core over the instructive alphabets.*

(a) Assume there exists a graph property $\mathcal{P}$ preserved under graph isomorphism such that, for every width bound k and every term $\tau \in \mathsf{ITD}_k$,

$$\tau \in \mathsf{Acc}(\mathcal{D}[k]) \quad \textit{if and only if} \quad \mathcal{G}(\tau) \in \mathcal{P}.$$

Then $\mathcal{D}$ satisfies Definition 25(2a)*.*

(b) Assume additionally that $\mathcal{D}$ is intended to induce a graph-level invariant and that there exists a graph invariant I, preserved under graph isomorphism, such that for every width bound k and every term $\tau \in \mathsf{ITD}_k$,

$$\mathtt{Inv}_k\big(\Gamma[\mathcal{D},k](\tau)\big) \;=\; I(\mathcal{G}(\tau)).$$

Then $\mathcal{D}$ satisfies Definition 25(2b)*.*

Hence, if both (a) *and* (b) *hold, then $\mathcal{D}$ is* ITD*-coherent in the sense of Definition 25.*

In particular, every canonical DP-core in the catalog satisfies the appropriate part(s): decision cores are handled by (a)*, while invariant-valued cores invoke both* (a) *and* (b) *once their root invariant has been identified with the corresponding graph-level invariant.*

Proof. Fix $k, k' \in \mathbb{N}$ and terms $\tau \in \mathsf{ITD}_k$ and $\tau' \in \mathsf{ITD}_{k'}$ with $\mathcal{G}(\tau) \simeq \mathcal{G}(\tau')$. Under the hypothesis of (a), isomorphism-invariance of $\mathcal{P}$ gives

$$\mathcal{G}(\tau) \in \mathcal{P} \iff \mathcal{G}(\tau') \in \mathcal{P}.$$

By the assumed acceptance characterization, this is equivalent to

$$\tau \in \mathsf{Acc}(\mathcal{D}[k]) \iff \tau' \in \mathsf{Acc}(\mathcal{D}[k']),$$

which is exactly Definition 25(2a).

Under the additional hypothesis of (b), isomorphism-invariance of I gives

$$I(\mathcal{G}(\tau)) = I(\mathcal{G}(\tau')).$$

Combining this with the assumed invariant identification yields

$$\mathtt{Inv}_k\big(\Gamma[\mathcal{D},k](\tau)\big) = I(\mathcal{G}(\tau)) = I(\mathcal{G}(\tau')) = \mathtt{Inv}_{k'}\big(\Gamma[\mathcal{D},k'](\tau')\big),$$

which is exactly Definition 25(2b). If both (a) and (b) hold, then both clauses of Definition 25 are satisfied, so $\mathcal{D}$ is ITD-coherent. □

Remark 17 (Decision coherence versus invariant coherence). The semantic identification

$$\Gamma[\mathcal{D},k](\tau) = \mathrm{Sem}_{\mathcal{D}}(\tau)$$

does not by itself determine which decomposition-independence statement one should extract at the root. For a decision core, the graph-level conclusion is only acceptance invariance:

$$\mathcal{G}(\tau) \simeq \mathcal{G}(\tau') \Longrightarrow \Big(\tau \in \mathsf{Acc}(\mathcal{D}[k]) \Leftrightarrow \tau' \in \mathsf{Acc}(\mathcal{D}[k'])\Big).$$

No canonical equality of raw root-open witness sets is asserted in that setting. For an invariant-valued core, one additionally asks for

$$\mathtt{Inv}_k\big(\Gamma[\mathcal{D},k](\tau)\big) = \mathtt{Inv}_{k'}\big(\Gamma[\mathcal{D},k'](\tau')\big)$$

whenever $\mathcal{G}(\tau) \simeq \mathcal{G}(\tau')$. Corollary 14 packages these two layers under the extra hypotheses that acceptance and `Inv` have already been identified with graph-level notions.

Graph model. By default, the realized graphs $\mathcal{G}(\tau)$ of instructive terms are finite undirected multigraphs: joins take multiset unions of edges, so parallel edges may arise. Whenever a core targets a simple-graph notion (e.g. clique), we state the required promise explicitly in Appendix D (Convention 3). Such promises can be enforced operationally by combining the core with the multi-edge detector core $\mathcal{C}^{\mathrm{ME}}$ treated below.

Boundaried graphs and vertex-pushouts. We use boundaried graphs and boundary gluing $\oplus_B$ as defined in Section 3.4 (Definition 18). We will freely refer to $G_1 \oplus_B G_2$ as the *vertex pushout along* B (disjoint union with identification of boundary vertices, keeping parallel edges distinct).

For a join node $\tau = \texttt{join}(\tau_1, \tau_2)$ with common bag B, the semantics of instructive decompositions gives a canonical isomorphism

$$(\mathcal{G}(\tau), \theta[\tau]) \;\cong\; (\mathcal{G}(\tau_1), \theta[\tau_1]) \oplus_B (\mathcal{G}(\tau_2), \theta[\tau_2]).$$

We write $\iota_i : \mathcal{G}(\tau_i) \hookrightarrow \mathcal{G}(\tau)$ for the induced pushout embeddings, so $\iota_i(\theta[\tau_i](u)) = \theta[\tau](u)$ for every $u \in B$.

Pushout reminder (join nodes). Let $\tau = \texttt{join}(\tau_1, \tau_2)$ be a join node with common bag B. By the semantics of instructive decompositions, $\mathcal{G}(\tau)$ is the pushout of the boundaried graphs $(\mathcal{G}(\tau_1), \theta[\tau_1])$ and $(\mathcal{G}(\tau_2), \theta[\tau_2])$ along the common boundary B. Concretely,

$$\iota_1(V(\mathcal{G}(\tau_1))) \cup \iota_2(V(\mathcal{G}(\tau_2))) = V(\mathcal{G}(\tau)),$$

$$\iota_1(V(\mathcal{G}(\tau_1))) \cap \iota_2(V(\mathcal{G}(\tau_2))) = \theta[\tau](B),$$

and $E(\mathcal{G}(\tau))$ is the multiset union of the image edge sets $\iota_1(E(\mathcal{G}(\tau_1)))$ and $\iota_2(E(\mathcal{G}(\tau_2)))$. When convenient, we identify a child graph with its image under ι_i.

Definition 57 (Pushout-gluing (join nodes))**.** *Fix a global-object semantics $X(G)$ (in applications, $X(G) = \mathcal{X}(G)$) and bag projections $\pi_{B(\sigma)} : X(\mathcal{G}(\sigma)) \to W_k$ as in Section 9.1.6. For a join node $\tau = \texttt{join}(\tau_1, \tau_2)$ with common bag B, we say that (X, π) is pushout-gluing at τ if:*

(G1) ***Restriction.*** *Every $x \in X(\mathcal{G}(\tau))$ has canonical restrictions $x_i := x|_{\tau_i} \in X(\mathcal{G}(\tau_i))$ ($i \in \{1, 2\}$), obtained by restricting along the embeddings ι_i.*

(G2) ***Compatibility and witness composition.*** *There is a relation $\mathrm{Comp}_B \subseteq W_k \times W_k$ and a (set-valued) partial composition $\otimes_B : W_k \times W_k \to \mathcal{P}_{\mathrm{fin}}(W_k)$ such that:*

- *(a) (sound gluing) if $x_i \in X(\mathcal{G}(\tau_i))$ and $\mathrm{Comp}_B(\pi_B(x_1), \pi_B(x_2))$, then for every $w \in \pi_B(x_1) \otimes_B \pi_B(x_2)$ there exists a glued $x \in X(\mathcal{G}(\tau))$ with $x|_{\tau_i} = x_i$ and $\pi_B(x) = w$.*
- *(b) (complete restriction) for every $x \in X(\mathcal{G}(\tau))$ with restrictions x_i, we have $\mathrm{Comp}_B(\pi_B(x_1), \pi_B(x_2))$ and $\pi_B(x) \in \pi_B(x_1) \otimes_B \pi_B(x_2)$.*

In the instantiations below, the join case is discharged either by verifying (G1) and (G2) for the chosen global objects and projection, or by giving an explicit join argument when more convenient.

Theorem 18 (Join from gluing)**.** *Let $\tau = \texttt{join}(\tau_1, \tau_2)$ be a join node with common bag B. Assume (X, π) is pushout-gluing at τ with some $\mathrm{Comp}_B, \otimes_B$.*

Then for the existential interpretation predicate

$$\mathsf{Interp}_X(\sigma, w) \;:\Leftrightarrow\; \exists x \in X(\mathcal{G}(\sigma))\ \big(\pi_{B(\sigma)}(x) = w\big),$$

the join case of (INT) *holds:*

$$\mathsf{Interp}_X(\tau, w) \;\Leftrightarrow\; \exists w_1, w_2\ \Big(\mathsf{Interp}_X(\tau_1, w_1) \wedge \mathsf{Interp}_X(\tau_2, w_2) \\ \wedge\ \mathrm{Comp}_B(w_1, w_2) \wedge w \in w_1 \otimes_B w_2\Big).$$

Proof. ($\Rightarrow$) Assume $\mathsf{Interp}_X(\tau, w)$. Then there exists $x \in X(\mathcal{G}(\tau))$ with $\pi_B(x) = w$. By (G1), x has restrictions $x_i := x|_{\tau_i} \in X(\mathcal{G}(\tau_i))$. Let $w_i := \pi_B(x_i)$. Then $\mathsf{Interp}_X(\tau_i, w_i)$ holds for $i \in \{1, 2\}$. By (G2b) applied to x and its restrictions, we have $\mathrm{Comp}_B(w_1, w_2)$ and $\pi_B(x) \in w_1 \otimes_B w_2$, i.e. $w \in w_1 \otimes_B w_2$.

($\Leftarrow$) Conversely, assume there exist w_1, w_2 such that $\mathsf{Interp}_X(\tau_i, w_i)$ holds for $i \in \{1, 2\}$, $\mathrm{Comp}_B(w_1, w_2)$ holds, and $w \in w_1 \otimes_B w_2$. Pick witnesses $x_i \in X(\mathcal{G}(\tau_i))$ with $\pi_B(x_i) = w_i$. By (G2a) (sound gluing) we obtain $x \in X(\mathcal{G}(\tau))$ with $x|_{\tau_i} = x_i$ and $\pi_B(x) = w$. Hence $\mathsf{Interp}_X(\tau, w)$. □

Example 5 (Gluing for c-colorability)**.** *Fix $c \in \mathbb{N}$ and let $X(G)$ be the set of proper c-colorings of G. For a bag B, the witness of a coloring γ is its restriction $\pi_B(\gamma) = \gamma|_B$, so witnesses are total maps $\chi : B \to [c]$. At a join node with bag B, compatibility is simply pointwise agreement:*

$$\mathrm{Comp}_B(\chi_1, \chi_2) \quad :\Leftrightarrow \quad \chi_1 = \chi_2.$$

The associated witness composition is therefore

$$\chi_1 \otimes_B \chi_2 = \begin{cases} \{\chi_1\}, & \text{if } \chi_1 = \chi_2, \\ \emptyset, & \text{otherwise.} \end{cases}$$

Theorem 18 then says that a parent bag-coloring is realizable at a join exactly when both child decompositions realize the same coloring on the common bag.

Canonical instantiations. We briefly indicate, for each core treated below, (i) the restriction/gluing operation on global objects, and (ii) the induced $(\mathrm{Comp}_B, \otimes_B)$ on witnesses. These instantiations justify one-line join proofs via Theorem 18. In all these instances, the chosen global-object semantics $X(G)$ is stable under restriction to join children and admits gluing along the join pushout. Moreover, the bag projection π_B is compatible with gluing in the sense captured by $(\mathrm{Comp}_B, \otimes_B)$. Theorem 18 packages the resulting join argument: once $(\mathrm{Comp}_B, \otimes_B)$ is specified for a core, the join case in later (INT) proofs reduces to a single citation.

9.1.5 Bitlength and complexity

We use the complexity measures *bitlength* and *multiplicity* from Chapter 4 (Definitions 28 and 29). For consistency with the literature and the per-core analyses below, we write $\beta_{\mathcal{D}}(k, n)$ for the bitlength at width k on terms of size at most n, and $\mu_{\mathcal{D}}(k, n)$ for the corresponding multiplicity (maximum size of a cleaned witness set). For finite cores, we drop the size parameter n. The full model-checking runtime bound in terms of (β, μ) is given in Section 4.7, Theorem 4 and its corollaries.

9.1.6 Local interpretation predicates and the DP proof rule

For each DP-core $\mathcal{D}[k]$ we fix a family of global objects $\mathcal{X}(G)$ and a bag projection $\pi_{B(\sigma)} : \mathcal{X}(\mathcal{G}(\sigma)) \to W_k$. We now factor denotation through a *predicate* explaining what a local witness *means* at a node.

Definition 58 (Local interpretation predicate)**.** *For a term σ and a witness $w \in W_k$, a predicate $\mathsf{Interp}_{\mathcal{D}}(\sigma, w)$ is intended to explain what w means (typically: that w is the bag projection of some global object in $\mathcal{X}(\mathcal{G}(\sigma))$, possibly strengthened to match a chosen canonization policy). It induces a witness-set semantics*

$$\mathrm{Sem}_{\mathcal{D}}^{\mathsf{Interp}}(\sigma) := \{\, w \in W_k : \mathsf{Interp}_{\mathcal{D}}(\sigma, w) \,\}.$$

In concrete cores, one defines $\mathsf{Interp}_{\mathcal{D}}$ explicitly and then shows that it characterizes the denotation, i.e. $\mathrm{Sem}_{\mathcal{D}}(\sigma) = \mathrm{Sem}_{\mathcal{D}}^{\mathsf{Interp}}(\sigma)$. Chapter 4 sometimes wrote the same local semantic relation with core-specific predicate names $\mathsf{Pred}_{\mathcal{D}}$; here $\mathsf{Interp}_{\mathcal{D}}$ is the canonical generic name, and those earlier symbols should be read as concrete instances of it. In many decision cores (where cleaning is the identity on denotational sets) $\mathsf{Interp}_{\mathcal{D}}$ is the existential predicate "$\exists x \in \mathcal{X}(\mathcal{G}(\sigma))$ with $\pi_{B(\sigma)}(x) = w$", while in optimization cores with nontrivial canonization one strengthens the predicate to match the chosen cleaning

(e.g., by a minimality condition as in Appendix D.12). In particular, for join nodes and existential predicates, the join case of (INT) *is discharged by Theorem 18.*

This lets us phrase the local law as a purely relational statement.

Typing convention for (INT) and (LSC). Definitions 53 and 54 use the bag-annotated signature $\overline{\Sigma}_k$. When we state (INT) or (LSC) below for a constructor symbol $a \in \Sigma_k$ and instructive parent term $\tau = a(\sigma_1, \ldots, \sigma_r)$, the intended algebra operation is the bag-annotated constructor $(a, B(\tau))$. This shorthand is type-correct because instructiveness fixes the parent bag uniquely.

For a constructor $a \in \Sigma_k$ of arity r and instructive terms $\sigma_1, \ldots, \sigma_r \in \mathsf{ITD}_k$ such that $a(\sigma_1, \ldots, \sigma_r) \in \mathsf{ITD}_k$, we require:

$$\mathsf{Interp}_{\mathcal{D}}\big(a(\sigma_1, \ldots, \sigma_r),\, w\big) \iff \exists w_1, \ldots, w_r \, \Big(\bigwedge_{i=1}^{r} \mathsf{Interp}_{\mathcal{D}}(\sigma_i, w_i) \;\wedge\; w \in \widehat{a}(w_1, \ldots, w_r) \Big). \tag{INT}$$

Equivalently, we may view (INT) as the inference rule

$$\frac{\mathsf{Interp}_{\mathcal{D}}(\sigma_1, w_1) \;\; \ldots \;\; \mathsf{Interp}_{\mathcal{D}}(\sigma_r, w_r) \quad w \in \widehat{a}(w_1, \ldots, w_r)}{\mathsf{Interp}_{\mathcal{D}}\big(a(\sigma_1, \ldots, \sigma_r), w\big)},$$

and its converse.

Lemma 27 (From (INT) to (LSC))**.** *Assume that* $\mathsf{Interp}_{\mathcal{D}}$ *characterizes denotation, i.e.* $\mathrm{Sem}_{\mathcal{D}}(\sigma) = \mathrm{Sem}_{\mathcal{D}}^{\mathsf{Interp}}(\sigma)$ *for all instructive terms* $\sigma \in \mathsf{ITD}_k$*. If* (INT) *holds for every constructor* $a \in \Sigma_k$ *whenever* $a(\vec{\sigma})$ *is an instructive term, then* (LSC) *holds for* $\mathcal{D}$*.*

Proof. Fix a constructor a of arity r and instructive terms $\sigma_1, \ldots, \sigma_r$ such that $a(\sigma_1, \ldots, \sigma_r) \in \mathsf{ITD}_k$, and abbreviate $S_i := \mathrm{Sem}_{\mathcal{D}}(\sigma_i) = \mathrm{Sem}_{\mathcal{D}}^{\mathsf{Interp}}(\sigma_i)$. Write

$$U_a := \bigcup_{\vec{w} \in \prod_i S_i} \widehat{a}(\vec{w}) \subseteq W_k.$$

Using (INT) and $\mathrm{Sem}_{\mathcal{D}} = \mathrm{Sem}_{\mathcal{D}}^{\mathsf{Interp}}$, we have

$$\mathrm{Sem}_{\mathcal{D}}\big(a(\vec{\sigma})\big) = \{w : \exists \vec{w} \in \prod_i S_i \text{ with } w \in \widehat{a}(\vec{w})\} = U_a.$$

On the other hand, by Definition 53,

$$\mathsf{F}_a(S_1, \ldots, S_r) = \mathtt{Clean}_k(U_a).$$

Since $U_a = \mathrm{Sem}_{\mathcal{D}}(a(\vec{\sigma}))$ and denotation is canonized by definition, we have $U_a \in \mathrm{Cl}(W_k)$ and therefore $\mathtt{Clean}_k(U_a) = U_a$. (Note that we use only idempotence of cleaning, not extensivity, which is important for cores where cleaning discards invalid states.) Therefore

$$\mathsf{F}_a\big(\mathrm{Sem}_{\mathcal{D}}(\sigma_1), \ldots, \mathrm{Sem}_{\mathcal{D}}(\sigma_r)\big) = \mathtt{Clean}_k(U_a) = U_a = \mathrm{Sem}_{\mathcal{D}}\big(a(\vec{\sigma})\big),$$

which is exactly (LSC). □

Theorem 19 (Fundamental DP-core correctness theorem)**.** *Assume* (LSC) *holds and* $\mathsf{Interp}_{\mathcal{D}}$ *characterizes denotation (i.e.* $\mathrm{Sem}_{\mathcal{D}}(\sigma) = \mathrm{Sem}_{\mathcal{D}}^{\mathsf{Interp}}(\sigma)$ *for all instructive* $\sigma \in \mathsf{ITD}_k$*). Then for all instructive terms* $\sigma \in \mathsf{ITD}_k$*,*

$$\Gamma[\mathcal{D}, k](\sigma) = \mathrm{Sem}_{\mathcal{D}}(\sigma) = \{w : \mathsf{Interp}_{\mathcal{D}}(\sigma, w)\}.$$

Consequently, once a concrete core separately identifies $\mathtt{Final}_k$ *with the intended root predicate and* $\mathtt{Inv}_k$ *with the intended graph invariant, the standard root-level soundness/completeness or exactness corollaries follow.*

Proof. Assuming (LSC), Theorem 17 gives $\Gamma[\mathcal{D}, k](\sigma) = \mathrm{Sem}_{\mathcal{D}}(\sigma)$. Since $\mathrm{Sem}_{\mathcal{D}} = \mathrm{Sem}_{\mathcal{D}}^{\mathsf{Interp}}$ by assumption, we also have $\mathrm{Sem}_{\mathcal{D}}(\sigma) = \{w : \mathsf{Interp}_{\mathcal{D}}(\sigma, w)\}$. □

Theorem 19 packages the reusable part of the argument. It does not by itself choose the global object family $\mathcal{X}(G)$, define the projection π_B, prove that $\mathsf{Interp}_{\mathcal{D}}$ characterizes denotation, discharge join/gluing arguments for join-heavy cores, or justify that cleaning/canonization preserves the denotation and the intended root output. These remain core-specific obligations, and they are exactly where most of the substantive work in later sections and Appendix D occurs.

Deferred certificate and dominance details. The framework-level derivation system and dominance-cleaning abstraction are provided in Appendix D.11. We keep the main chapter focused on the core semantic equivalence and the worked correctness instantiations.

9.2 c-Colorability

Fix $c \in \mathbb{N}_{\geq 1}$. This section discharges the correctness proof for the core $\mathcal{C}_c^{\chi}$ defined in Section 4.8.1. We decide whether $\mathcal{G}(\tau)$ admits a proper c-coloring, instantiating the

generic proof framework of Section 9.1 with proper colorings as global objects.

9.2.1 Witness space, cleaning, and acceptance

We first fix the witness representation and the corresponding cleaning and acceptance conventions.

Witness space. At a node with top bag B a witness is a total map $\chi : B \to [c]$. We always take $\mathrm{dom}(\chi) = B$ and identify such maps with elements of $[c]^B$. Set

$$W_k^{\mathrm{col}} = \big\{\, \chi \mid \exists B \subseteq [k{+}1],\ \chi \in [c]^B \,\big\}.$$

This is a labeled refinement of the partition-based witness presentation from Section 4.8.1; Lemma 28 makes the correspondence explicit.

Lemma 28 (Agreement with the Chapter 4 chromatic presentation)**.** *Fix $k \in \mathbb{N}$, an instructive term $\sigma \in \mathsf{ITD}_k$, and let $B := B(\sigma)$. For any bag $C \subseteq [k+1]$, define*

$$\mathrm{Part}_{\le c}(C) := \{\mathcal{P} \mid \mathcal{P} \text{ is a partition of } C \text{ with at most } c \text{ blocks}\},$$

$$\mathrm{part}_C : [c]^C \to \mathrm{Part}_{\le c}(C), \qquad \mathrm{part}_C(\chi) := \ker(\chi),$$

and let $\mathrm{can}_C : \mathrm{Part}_{\le c}(C) \to [c]^C$ be the canonical representative obtained by ordering the blocks of $\mathcal{P}$ by increasing least label and assigning the colors $1, \dots, |\mathcal{P}|$ in that order. Write

$$\Gamma_k^{\mathrm{part}}(\eta) \subseteq \mathrm{Part}_{\le c}(B(\eta)), \qquad \Gamma_k^{\mathrm{tot}}(\eta) \subseteq [c]^{B(\eta)}$$

for the witness sets computed at an instructive term η by the Chapter 4 partition presentation and by the total-map presentation used in this section, respectively. Then:

(i) For every $\mathcal{P} \in \mathrm{Part}_{\le c}(B)$,

$$\mathsf{P}_{c,k}^{\chi}(\sigma, \mathcal{P}) \iff \mathsf{Interp}_{\mathcal{C}_c^{\chi}}\big(\sigma, \mathrm{can}_B(\mathcal{P})\big).$$

(ii) Let a be a constructor of arity r, and let $B_1, \dots, B_r$ and B be the child and parent bags determined by an instructive occurrence of a. For every tuple of child partitions $\mathcal{P}_i \in \mathrm{Part}_{\le c}(B_i)$,

$$\big\{\mathrm{part}_B(\chi) \;\big|\; \chi \in \widehat{a}\big(\mathrm{can}_{B_1}(\mathcal{P}_1), \dots, \mathrm{can}_{B_r}(\mathcal{P}_r)\big)\big\}$$

is exactly the set of parent partitions prescribed by the corresponding transition rule for $\mathcal{C}_c^\chi$ in Section 4.8.1.

(iii) For every instructive term $\eta \in \mathsf{ITD}_k$,

$$\Gamma_k^{\mathrm{part}}(\eta) = \{\mathrm{part}_{B(\eta)}(\chi) \mid \chi \in \Gamma_k^{\mathrm{tot}}(\eta)\}.$$

Consequently, the total-map presentation used in this section and the partition presentation from Chapter 4 accept exactly the same instructive terms.

Proof. For (i), let $\mathcal{P} \in \mathrm{Part}_{\le c}(B)$. If $\mathsf{P}_{c,k}^\chi(\sigma, \mathcal{P})$ holds, choose a proper coloring $\kappa : V(\mathcal{G}(\sigma)) \to [c]$ whose equality relation on the bag is exactly $\mathcal{P}$. The colors used by κ on the blocks of $\mathcal{P}$ form a subset of $[c]$ of size $|\mathcal{P}|$. The map sending the color used on each block to the canonical color assigned to that block by $\mathrm{can}_B(\mathcal{P})$ is injective, hence extends to a permutation π of $[c]$. Applying π to κ on all vertices preserves properness and makes the bag projection equal to $\mathrm{can}_B(\mathcal{P})$. Thus the recoloured map witnesses $\mathsf{Interp}_{\mathcal{C}_c^\chi}(\sigma, \mathrm{can}_B(\mathcal{P}))$. The converse implication is immediate from Definition 59: if a proper coloring projects to $\mathrm{can}_B(\mathcal{P})$, then its equality relation on B is exactly $\mathcal{P}$.

For (ii), inspect the constructor clauses. For `leaf` both presentations produce the empty witness. For `intro_`v, extending a canonical coloring by a color already used on $B \setminus \{v\}$ inserts v into the corresponding block, while using a fresh color creates a new singleton block; after quotienting by part_B, these are exactly the Chapter 4 partition updates. For `forget_`v, both presentations delete v from the current bag data. For `intro_`u`_`v, the total-map transition accepts exactly when the endpoint colors differ, which is equivalent to u and v lying in different blocks. For `join`, the canonical representatives of two child partitions agree if and only if the partitions themselves agree, so join keeps exactly equal bag partitions. On instructive terms, no extra undefined-label cases arise, because the current domain is exactly the current bag and introduce-edge side conditions place both endpoints in that bag. Hence the transported transition system is exactly the one specified in Section 4.8.1.

For (iii), first note that total-map reachability is closed under global color permutations: if $\chi \in \Gamma_k^{\mathrm{tot}}(\eta)$ and π is a permutation of $[c]$, then $\pi \circ \chi \in \Gamma_k^{\mathrm{tot}}(\eta)$. This follows by induction on the derivation of χ: the leaf map is fixed, introduce and forget commute with applying π, the edge test only checks inequality, and join only checks equality of the two child maps.

We prove the displayed equality by structural induction on η. The inclusion from right to left is obtained by following a successful total-map derivation and applying part to every witness in it. The constructor analysis in (ii) shows that each quotient step is a

valid partition transition; in the join case, literal equality of child maps implies equality of their induced partitions.

For the converse inclusion, let a partition witness $\mathcal{P} \in \Gamma_k^{\mathrm{part}}(\eta)$ be produced by a partition derivation. The leaf case is immediate. For an introduce-vertex step, the induction hypothesis gives a total-map child witness inducing the child partition; if the new label is inserted into an old block, extend the child map by the color of that block, while if the new label forms a singleton block, choose any color not used by the child map. Such a color exists exactly when the partition transition allows creation of a new block. Forget-vertex and introduce-edge steps are immediate from restriction and from the fact that different partition blocks carry different colors. For a join step, the partition rule uses the same partition $\mathcal{P}$ in both children. By induction, choose child total maps χ_1, χ_2 inducing $\mathcal{P}$. Since χ_1 and χ_2 have the same kernel on the common bag, a permutation of $[c]$ sends χ_2 to χ_1 on that bag; by permutation closure the renamed χ_2 is still reachable in the second child. The total-map join can therefore combine two literally equal child maps and produce a parent total map whose quotient is $\mathcal{P}$. This proves the displayed equality. Since every chromatic witness is final, root acceptance is non-emptiness in both presentations, so the accepted instructive terms coincide. □

Cleaning and acceptance. $\mathtt{Clean}_k$ is the identity. Every witness is final. Acceptance at the root is non-emptiness of the root witness set.

9.2.2 Transitions as set-valued functions

Let $a \in \{\mathtt{leaf}, \mathtt{intro\_}v, \mathtt{forget\_}v, \mathtt{intro\_}u\mathtt{\_}v, \mathtt{join}\}$. With B the parent bag and B_i the ith child bag we regard

$$\widehat{a} : \prod_i [c]^{B_i} \to \mathcal{P}_{\mathrm{fin}}([c]^B)$$

as the restriction of the global transition $\widehat{a} : W_k^{\mathrm{r}(a)} \to \mathcal{P}_{\mathrm{fin}}(W_k)$ to the appropriate bags. Concretely:

- For the nullary constructor $\mathtt{leaf}$ with bag $B = \varnothing$ we set

$$\widehat{\mathtt{leaf}}() = \{\emptyset\}.$$

- For $\mathtt{intro\_}v$ the child bag is $B \setminus \{v\}$ and the parent bag is B. Given a coloring

$\chi' \in [c]^{B\setminus\{v\}}$ we define

$$\widehat{\texttt{intro\_}v}(\chi') = \{\, \chi \in [c]^B : \chi|_{B\setminus\{v\}} = \chi' \,\} = \{\, \chi'[v \mapsto i] : i \in [c] \,\}.$$

- For $\texttt{forget\_}v$ the child bag is $B \cup \{v\}$ and the parent bag is B. Given $\chi' \in [c]^{B\cup\{v\}}$ we set
$$\widehat{\texttt{forget\_}v}(\chi') = \{\, \chi'|_B \,\}.$$
- For $\texttt{intro\_}u\texttt{\_}v$ we have $B_i = B$ for the unique child. For $\chi \in [c]^B$ and $u, v \in B$,
$$\widehat{\texttt{intro\_}u\texttt{\_}v}(\chi) = \begin{cases} \{\chi\}, & \chi(u) \neq \chi(v), \\ \varnothing, & \chi(u) = \chi(v). \end{cases}$$
- For $\texttt{join}$ we have two children with the same bag B. For $\chi_1, \chi_2 \in [c]^B$,
$$\widehat{\texttt{join}}(\chi_1, \chi_2) = \begin{cases} \{\chi_1\}, & \chi_1 = \chi_2 \text{ on } B, \\ \varnothing, & \text{otherwise.} \end{cases}$$

The set-lift F_a is union (cleaning is identity).

9.2.3 Denotation and the local interpretation predicate

We now define the global objects and the interpretation predicate used to characterize denotation.

Global objects and projection. $\mathcal{X}_{\mathrm{col}}(G)$ is the set of proper c-colorings $\kappa : V(G) \to [c]$. For a term σ with bag $B(\sigma)$ and labeling $\theta[\sigma]$ define

$$\pi_{B(\sigma)}(\kappa) = \kappa \circ \theta[\sigma] \in [c]^{B(\sigma)}.$$

Local interpretation predicate.

Definition 59 (Chromatic interpretation)**.** *For a term σ and a witness $\chi \in [c]^{B(\sigma)}$, set*

$$\mathsf{Interp}_{\mathcal{C}_c^{\chi}}(\sigma, \chi) \iff \exists \kappa \in \mathcal{X}_{\mathrm{col}}(\mathcal{G}(\sigma))\ \forall u \in B(\sigma) :\ \chi(u) = \kappa\big(\theta[\sigma](u)\big).$$

Equivalently (partition reading): if $\equiv_\chi$ is the kernel of χ on $B(\sigma)$, then Interp *says there is a proper coloring κ whose restriction makes $u \equiv_\chi v$ exactly when $\kappa(\theta[\sigma](u)) = \kappa(\theta[\sigma](v))$.*

By construction, $\mathrm{Sem}_{\mathcal{C}_c^\chi}(\sigma) = \{\, \chi : \mathsf{Interp}_{\mathcal{C}_c^\chi}(\sigma, \chi) \,\}$.

9.2.4 Local law (INT / LSC)

We next verify the constructor-wise local interpretation law for this core.

Lemma 29 (Chromatic INT)**.** *Fix $k \in \mathbb{N}$, let a be a constructor of arity r, and let $\sigma_1, \ldots, \sigma_r \in \mathsf{ITD}_k$ be instructive child terms such that $\tau := a(\sigma_1, \ldots, \sigma_r) \in \mathsf{ITD}_k$. Then, for every $\chi \in [c]^{B(\tau)}$,*

$$\mathsf{Interp}_{\mathcal{C}_c^\chi}(\tau, \chi) \iff \exists \vec{\chi} \left(\bigwedge_i \mathsf{Interp}_{\mathcal{C}_c^\chi}(\sigma_i, \chi_i) \ \wedge \ \chi \in \widehat{a}(\vec{\chi}) \right).$$

Proof. Leaf. $\mathcal{G}(\texttt{leaf})$ is empty. The unique proper coloring κ projects to the empty map $\pi_\emptyset(\kappa) = \emptyset$, and by definition $\widehat{\texttt{leaf}}() = \{\emptyset\}$, so INT holds.

Introduce vertex. Let $\tau = \texttt{intro\_}v(\sigma)$ with parent bag $B(\tau) = B(\sigma) \cup \{v\}$. ($\Rightarrow$) Suppose $\mathsf{Interp}(\tau, \chi)$ holds, witnessed by a proper coloring κ of $\mathcal{G}(\tau)$. Restricting to $\mathcal{G}(\sigma)$ gives a proper coloring κ_σ with $\chi_1 := \pi_{B(\sigma)}(\kappa_\sigma) \in [c]^{B(\sigma)}$ satisfying $\mathsf{Interp}(\sigma, \chi_1)$. Moreover $\chi = \pi_{B(\tau)}(\kappa)$ extends χ_1 by a choice of color at v, so by the definition of $\widehat{\texttt{intro\_}v}$ we have $\chi \in \widehat{\texttt{intro\_}v}(\chi_1)$. ($\Leftarrow$) Conversely, assume there exists χ_1 with $\mathsf{Interp}(\sigma, \chi_1)$, witnessed by a proper coloring κ_σ on $\mathcal{G}(\sigma)$, and $\chi \in \widehat{\texttt{intro\_}v}(\chi_1)$. By construction χ extends χ_1 by some color $i \in [c]$ at v. Define κ on $\mathcal{G}(\tau)$ by coloring the new vertex $\theta[\tau](v)$ with i and agreeing with κ_σ elsewhere. Since $\texttt{intro\_}v$ adds an isolated vertex, no new edges are incident to $\theta[\tau](v)$ and κ is proper, witnessing $\mathsf{Interp}(\tau, \chi)$.

Forget vertex. Let $\tau = \texttt{forget\_}v(\sigma)$ with $B(\tau) = B(\sigma) \setminus \{v\}$. ($\Rightarrow$) If $\mathsf{Interp}(\tau, \chi)$ holds via a proper coloring κ on $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$, then $\chi_1 := \pi_{B(\sigma)}(\kappa)$ satisfies $\mathsf{Interp}(\sigma, \chi_1)$ and $\chi = \chi_1|_{B(\tau)} \in \widehat{\texttt{forget\_}v}(\chi_1)$. ($\Leftarrow$) Conversely, if $\mathsf{Interp}(\sigma, \chi_1)$ holds via a proper coloring κ on $\mathcal{G}(\sigma)$, then the same κ witnesses $\mathsf{Interp}(\tau, \chi_1|_{B(\tau)})$, and by definition $\chi_1|_{B(\tau)} \in \widehat{\texttt{forget\_}v}(\chi_1)$.

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$. Here $B(\tau) = B(\sigma)$ and the labeling agrees. ($\Rightarrow$) Suppose $\mathsf{Interp}(\tau, \chi)$ holds via a proper coloring κ on $\mathcal{G}(\tau)$. Restricting to $\mathcal{G}(\sigma)$ yields a proper coloring κ_σ and

$$\chi_1 := \pi_{B(\sigma)}(\kappa_\sigma) = \pi_{B(\tau)}(\kappa) = \chi.$$

Thus $\mathsf{Interp}(\sigma, \chi_1)$ holds and, because κ is proper on the edge uv, we have $\chi(u) \neq \chi(v)$, so $\chi \in \widehat{\texttt{intro\_}u\texttt{\_}v}(\chi_1)$. ($\Leftarrow$) Conversely, if $\chi_1 \in \mathrm{Sem}_{\mathcal{C}_c^\chi}(\sigma)$ via a proper coloring κ_σ and $\chi \in \widehat{\texttt{intro\_}u\texttt{\_}v}(\chi_1)$, then $\chi = \chi_1$ and $\chi(u) \neq \chi(v)$. The same coloring κ_σ witnesses

$\mathsf{Interp}(\tau, \chi)$, since adding the edge between two differently colored endpoints preserves properness.

Join. Let $\tau = \texttt{join}(\sigma_1, \sigma_2)$ with common bag $B := B(\tau) = B(\sigma_1) = B(\sigma_2)$. ($\Rightarrow$) Suppose $\mathsf{Interp}(\tau, \chi)$ holds via a proper coloring $\kappa : V(\mathcal{G}(\tau)) \to [c]$. Restricting κ to each child gives proper colorings κ_i of $\mathcal{G}(\sigma_i)$ with

$$\chi_i := \pi_{B(\sigma_i)}(\kappa_i) = \pi_{B(\tau)}(\kappa) = \chi.$$

Hence $\mathsf{Interp}(\sigma_i, \chi_i)$ holds for $i = 1, 2$. Since the child bag colorings agree, the definition of $\widehat{\texttt{join}}$ gives

$$\chi \in \widehat{\texttt{join}}(\chi_1, \chi_2).$$

($\Leftarrow$) Conversely, suppose there exist child witnesses χ_1, χ_2 such that $\mathsf{Interp}(\sigma_i, \chi_i)$ holds for $i = 1, 2$ and $\chi \in \widehat{\texttt{join}}(\chi_1, \chi_2)$. By definition of the join transition, χ_1 and χ_2 agree on the common bag B, and $\chi = \chi_1 \cup \chi_2$; since both domains equal B, this means $\chi = \chi_1 = \chi_2$. Choose proper child colorings κ_i witnessing $\mathsf{Interp}(\sigma_i, \chi_i)$. Because the child colorings agree on the identified bag, they glue to a unique map $\kappa : V(\mathcal{G}(\tau)) \to [c]$. Every parent edge comes from one child, so κ is proper. Moreover $\pi_{B(\tau)}(\kappa) = \chi$, hence $\mathsf{Interp}(\tau, \chi)$. □

9.2.5 From local law to global correctness

We now derive global correctness as a direct consequence of the local law and the proof framework.

Theorem 20 (Chromatic correctness via the proof framework)**.** *The chromatic interpretation law (Lemma 29), together with the fact that* $\texttt{Clean}_k$ *is the identity for* $\mathcal{C}_c^\chi$*, implies the LSC equality for* $\mathcal{C}_c^\chi$*. Hence, by Theorems 17 and 19, for every* $\tau \in \mathsf{ITD}_k$*,*

$$\Gamma[\mathcal{C}_c^\chi, k](\tau) = \mathrm{Sem}_{\mathcal{C}_c^\chi}(\tau) = \{\, \chi : \mathsf{Interp}_{\mathcal{C}_c^\chi}(\tau, \chi) \,\}.$$

Proof. Fix a constructor a of arity ℓ and child terms $\sigma_1, \ldots, \sigma_\ell$, and abbreviate $S_i := \mathrm{Sem}_{\mathcal{C}_c^\chi}(\sigma_i)$. Lemma 29 and Definition 58 give

$$\mathrm{Sem}_{\mathcal{C}_c^\chi}\big(a(\sigma_1, \ldots, \sigma_\ell)\big) = \big\{\chi : \exists \vec{\chi} \in S_1 \times \cdots \times S_\ell \text{ with } \chi \in \widehat{a}(\vec{\chi})\big\} =: U_a.$$

For the chromatic core, cleaning is the identity, so by Definition 53

$$\mathsf{F}_a(S_1, \ldots, S_\ell) = \texttt{Clean}_k(U_a) = U_a.$$

Thus

$$\mathrm{Sem}_{\mathcal{C}_c^X}\big(a(\sigma_1,\dots,\sigma_\ell)\big) = \mathsf{F}_a\big(\mathrm{Sem}_{\mathcal{C}_c^X}(\sigma_1),\dots,\mathrm{Sem}_{\mathcal{C}_c^X}(\sigma_\ell)\big)$$

for every constructor a and tuple of children, i.e. (LSC) holds for $\mathcal{C}_c^X$. Theorem 17 then gives $\Gamma[\mathcal{C}_c^X,k](\tau) = \mathrm{Sem}_{\mathcal{C}_c^X}(\tau)$ for all instructive terms $\tau \in \mathsf{ITD}_k$, and Definition 59 yields the characterization via $\mathsf{Interp}_{\mathcal{C}_c^X}$. □

Theorem 21 (Soundness and completeness)**.** *For every $k \in \mathbb{N}$ and every $\tau \in \mathsf{ITD}_k$ (not necessarily closed),*

$$\tau \in \mathsf{Acc}(\mathcal{C}_c^X[k]) \quad\Longleftrightarrow\quad \mathcal{G}(\tau) \text{ is } c\text{-colorable.}$$

Proof. By Theorem 20,

$$\Gamma[\mathcal{C}_c^X,k](\tau) = \mathrm{Sem}_{\mathcal{C}_c^X}(\tau).$$

Since every witness is final for this decision core,

$$\tau \in \mathsf{Acc}(\mathcal{C}_c^X[k]) \iff \Gamma[\mathcal{C}_c^X,k](\tau) \neq \emptyset \iff \mathrm{Sem}_{\mathcal{C}_c^X}(\tau) \neq \emptyset.$$

By Definition 59,

$$\chi \in \mathrm{Sem}_{\mathcal{C}_c^X}(\tau) \iff \exists \kappa : V(\mathcal{G}(\tau)) \to [c] \text{ proper with } \pi_{B(\tau)}(\kappa) = \chi.$$

Hence $\mathrm{Sem}_{\mathcal{C}_c^X}(\tau) \neq \emptyset$ iff $\mathcal{G}(\tau)$ is c-colorable. □

9.2.6 Equivariance and acceptance invariance

We record the equivariance argument and the decomposition-independent acceptance statement for the chromatic decision core.

Lemma 30 (Relabeling equivariance)**.** *For every permutation ρ, every operator a, and every compatible witness tuple $\vec{\chi}$, $\rho \cdot \widehat{a}(\vec{\chi}) = \widehat{\rho a}(\rho \cdot \vec{\chi})$. Consequently $\rho \cdot \Gamma[\mathcal{C}_c^X,k](\tau) = \Gamma[\mathcal{C}_c^X,k](\rho \cdot \tau)$ and likewise for* Sem*.*

Proof. The transitions use only equality tests on bag labels and function restriction/extension, which commute with permutations, so $\mathcal{C}_c^X$ is equivariant in the sense of Definition 56. To apply Lemma 26, take as global objects the proper c-colorings of the realized graph. For every permutation ρ, relabeling an instructive term by ρ does not change the underlying graph; it only renames the active bag labels. Hence the identity map on proper colorings gives the required transport bijection

$$\rho_* : \mathcal{X}(\mathcal{G}(\tau)) \to \mathcal{X}(\mathcal{G}(\rho \cdot \tau)).$$

For every proper coloring κ, the induced bag coloring on $B(\rho \cdot \tau) = \rho(B(\tau))$ is exactly $\rho \cdot \pi_{B(\tau)}(\kappa)$, so the projection compatibility hypothesis of Lemma 26 holds. The displayed consequences for $\Gamma[\mathcal{C}_c^\chi, k]$ and $\mathrm{Sem}_{\mathcal{C}_c^\chi}$ therefore follow from that lemma. □

Proposition 8 (Acceptance invariance for $\mathcal{C}_c^\chi$)**.** *Let $\tau \in \mathrm{ITD}_k$ and $\tau' \in \mathrm{ITD}_{k'}$. If $\mathcal{G}(\tau) \cong \mathcal{G}(\tau')$, then*

$$\tau \in \mathsf{Acc}(\mathcal{C}_c^\chi[k]) \iff \tau' \in \mathsf{Acc}(\mathcal{C}_c^\chi[k']).$$

Proof. By Theorem 21, acceptance is equivalent to c-colorability of the realized graph, and this property is isomorphism-invariant. □

Remark 18. For this decision core, the decomposition-independent statement is acceptance invariance as in Remark 17. One should not use Definition 25 with $\mathtt{Inv} = \mathrm{id}$ on raw root-open witness sets.

9.2.7 Bitlength, multiplicity, and running time

Let B be a bag of size $m := |B| \leq k{+}1$.

Theorem 22 (Encoding and multiplicity)**.** *A witness $\chi \in [c]^B$ can be encoded in $\beta_{\mathrm{col}}(k) = O((k+1)\log(c+1))$ bits. The per-node multiplicity satisfies $\mu_{\mathrm{col}}(k) \leq c^m \leq c^{k+1}$.*

Proof. Encode χ as the length-$(k+1)$ vector

$$z_\chi \in \{0, 1, \ldots, c\}^{[k+1]}$$

defined by $z_\chi(u) = \chi(u)$ when $u \in B$ and $z_\chi(u) = 0$ when $u \notin B$. The zero symbol records that the label u is absent from the current bag, so this encoding determines both the domain B and the color value of every active label. Each coordinate uses $O(\log(c+1))$ bits, hence the total encoding length is $O((k+1)\log(c+1))$. For multiplicity, there are exactly c^m possible colorings of B, so the witness set at any node has size at most $c^m \leq c^{k+1}$ when $m \leq k+1$. □

Lemma 31 (Internal polynomiality of $\mathcal{C}_c^\chi$)**.** *For fixed c, under the zero-padded encoding above, the core $\mathcal{C}_c^\chi$ is internally polynomial.*

Proof. Membership in W_k^{col} is checked by verifying that the encoding has length $k{+}1$ and each coordinate lies in $\{0, 1, \ldots, c\}$. The nullary transition outputs the unique empty coloring. For fixed c, the `intro_`u transition outputs at most c extensions of one input coloring. The transitions for `intro_`u, `forget_`u, `intro_`u`_`v, and `join` examine or

update only the $k+1$ coordinates of their input witnesses, so each transition and the finality test are computable in time polynomial in k and the total encoding length of the inputs. Cleaning is the identity. The invariant output is the constant empty string, so its output length is $O(1)$ and its computation is constant time. Thus all clauses of Definition 33 hold. □

Theorem 23 (Running time)**.** *For fixed c, on a term of size n, $T_{\mathrm{col}}(k,n) = n \cdot (k+1)^{O(1)} \cdot (\log(c+1))^{O(1)} \cdot c^{O(k+1)}$.*

Proof. By Lemma 31, Corollary 2 applies to $\mathcal{C}_c^{\chi}$. Substituting the bounds from Theorem 22 gives

$$n \cdot (k+1)^{O(1)} \cdot \big((k+1)\log(c+1)\big)^{O(1)} \cdot \big(c^{k+1}\big)^{O(1)},$$

which simplifies to the stated $n \cdot (k+1)^{O(1)} \cdot (\log(c+1))^{O(1)} \cdot c^{O(k+1)}$. Joins can be implemented by hashing bag-color tuples. □

Remark 19 (Partition variant and isomorphism). Lemma 28 already identifies the partition and total-map presentations semantically. For complexity estimates one may work directly with partitions. Let $\mathrm{Part}_{\le c}(B)$ be partitions of B with at most c blocks. The map $\mathrm{part} : [c]^B \to \mathrm{Part}_{\le c}(B),\ \mathrm{part}(\chi) = \text{kernel of } \chi$ is a surjection whose fibers are color-renamings. One may restate all transitions on $\mathrm{Part}_{\le c}(B)$ (introduce inserts v into an existing block or creates a fresh one, edge forbids u, v in the same block, and join keeps equal partitions), and define $\mathsf{Interp}_{\mathcal{C}_c^{\chi}}(\sigma, \mathcal{P}) \iff \exists c \in \mathcal{X}_{\mathrm{col}}(\mathcal{G}(\sigma))\ \forall u, v \in B(\sigma) : u \sim_{\mathcal{P}} v \Leftrightarrow c(\theta[\sigma](u)) = c(\theta[\sigma](v))$. The proofs above carry over verbatim. Multiplicities sharpen to restricted Bell numbers $\sum_{i \le \min\{c,m\}} S(m,i)$.

Additional core proofs. For space, the detailed instantiations of the proof framework for additional catalog cores (maximum degree, clique, multi-edge detection, and perfect matching) are collected in Appendix D.

9.3 Further Instantiations and Extensions

Further instantiations of the same proof pattern, including Maximum Independent Set and Vertex Counting, are deferred to Appendix D.

For brevity, certificate-reconstruction details are omitted here.

A lower-bound perspective via boundary equivalence and Myhill–Nerode-style arguments is also deferred to Appendix D.15.

9.3.1 Guidance for New Cores

The method of Section 9.1 is not limited to the catalog of cores treated above. To certify a new core at width k, follow the recipe of Section 9.1: choose global objects $\mathcal{X}(G)$ and a bag projection, define an interpretation predicate compatible with your cleaning, and prove the local law (INT) (or directly (LSC)) for each constructor. Theorem 19 then yields

$$\Gamma[\mathcal{D}, k](\sigma) = \mathrm{Sem}_{\mathcal{D}}(\sigma) = \{w : \mathsf{Interp}_{\mathcal{D}}(\sigma, w)\}$$

for all instructive terms $\sigma \in \mathsf{ITD}_k$, and soundness/completeness (or exactness for invariants) follows at the root. In practice, the only cases that typically require additional, core-specific arguments are join (via pushout-style gluing, Theorem 18) and cores with nontrivial canonization (where $\mathsf{Interp}_{\mathcal{D}}$ must characterize precisely the witnesses that survive cleaning, as in vertex cover).

Chapter 10

Empirical Validation of the Implemented DP-cores

This chapter reports a fixed-instance comparison between selected DP-cores implemented in TreeWidzard and external reference implementations. For all benchmarked properties except Vertex counting, the reference is the corresponding SAT encoding recorded in Appendix E; Vertex counting is checked instead against direct node counting in the comparison harness. For each benchmark graph, validated decomposition input, and fixed property query, TreeWidzard and the corresponding external reference are run on the same instance and their outputs are compared. For witness-bearing positive outcomes, the comparison harness additionally attempts an independent decode-and-check step; in this chapter we report the aggregate verdict agreement and use witness validation as a per-instance guard, rather than as a separate headline metric.

The point of the chapter is to test these implementations only on the selected benchmark families reported in the main aggregate table. It does not add new semantic proofs beyond the earlier correctness chapter, and it is not a full verification of the software stack. The detailed agreement results are reported later in the chapter. The comparison concerns the implemented fixed-instance execution workflow on validated graph/decomposition inputs; it is not a claim about the width-bounded ATP search machinery or about TreeWidzard as a whole.

10.1 Experiment setup and graph generation

The experiments in this chapter compare selected TreeWidzard DP-core implementations against the corresponding external references. For all benchmarked properties except Vertex counting, this reference is the SAT encoding from Appendix E; for Vertex

counting it is the direct graph-side node-count computation used by the harness. The CNF instances generated from the encodings in Appendix E are solved through the PySAT toolkit (version recorded in the experimental log) using its default backend, matching the configuration described in Appendix E. For each selected property, we fix the parameter values in the property file, run TreeWidzard and the relevant reference implementation on the same graph instance, and compare the resulting outcomes. When a positive witness or certificate is available, the harness also attempts to decode and validate it independently, but those witness-level checks are not tabulated separately in the aggregate summary below.

The benchmark instances come from four sources. The main reported aggregate in Table 10.1 uses the first two; the last two are supplementary checks.

Random Erdős–Rényi graphs. For this family we use $n \in \{20, 40, 60, 80, 100\}$, $p \in \{0.05, 0.10, 0.20, 0.30\}$, and 25 random seeds. For each choice of (n, p, s), we sample one graph $G \sim G(n, p)$. Each sampled graph is serialized in PACE `.gr` format, one tree decomposition is computed, and the resulting `.td` file is parsed and validated. Only graph/decomposition pairs whose parsed width is at most 30 are kept in the reported benchmark set.

Graphs generated from random decompositions. These runs start from validated decomposition structures rather than from raw graphs. We sample decomposition objects first, validate and serialize them as `.td`, and then derive the corresponding primal graphs as `.gr` files from bag incidence. This produces graph/decomposition pairs that are already aligned with the bottom-up DP workflow used by TreeWidzard.

Edge-case graphs. We also include a hand-crafted suite of small-width and threshold-style examples, such as paths, cycles, stars, and other structured sparse graphs. These are used as targeted robustness checks and are not merged into the main aggregate table.

Online enumeration of non-isomorphic graphs. As an additional local check, we also enumerate unlabeled graphs online with `nauty` (`geng`) up to eight vertices and feed them directly to TreeWidzard and, when enabled, to the SAT harness [McKay, 1998; McKay and Piperno, 2014]. These runs are reported separately from the main aggregate because they are meant as an end-to-end sanity check rather than as part of the main large-scale run.

10.2 Benchmarked properties and DP-cores

The main reported benchmark set in this chapter uses implemented DP-cores for the following graph properties:

Property family	Tested queries
Colorability	3-, 4-, and 5-colorability
Vertex cover	thresholds 3, 4, and 5
Clique on simple graphs	thresholds 3, 4, and 5
Independent set	threshold queries $\alpha(G) \geq 3$, $\alpha(G) \geq 5$, and $\alpha(G) \geq 6$
Hamiltonian path	decision query
Perfect matching	decision query
Feedback vertex set	thresholds 2, 3, and 4
Maximum degree	thresholds 3, 4, and 5
Multiple-edge detection	decision query
Vertex counting	invariant query

These are all fixed-instance queries: each benchmark row combines one graph instance with one concrete threshold or one unparameterized decision property, and TreeWidzard and the corresponding external reference are asked exactly the same question on that instance. The only special case is Vertex counting, which is not checked against a CNF encoding but against direct node counting in the comparison harness. The threshold values are not intended as canonical parameter choices. They were chosen as a small representative set of low-to-moderate fixed queries for each parameterized family, so that the benchmark study exercises each DP-core and its external counterpart at more than one query level while remaining within a manageable computational budget.

10.3 Execution environment and run setup

The experiments over these benchmark families were executed as Slurm array jobs with one array element per benchmarked query and an internal loop over all instances in the selected benchmark suite. The random-graph benchmark suite used 22 benchmark-query array tasks running the queries listed in Section 10.2 (the Hamiltonian-cycle decision query was excluded from the main reported set), for a total of 5,730 rows. Its runtime setup used four workers per task, a 1200-second per-instance timeout, and Slurm limits of 24:00:00 wall time, 4 CPUs, and 140 GiB of memory. The random tree-decomposition benchmark suite used the same one-task-per-query pattern with 22 array tasks, for a total of 10,950 rows; per-task instance counts vary by query (queries that exhausted their cluster budget contribute fewer instances and show up as `timeout`/`unknown` rows

Campaign	Total	Decided	Inconclusive	Comparable	Matches	Mismatches
Random graphs (main reported set)	5,730	4,998	732	4,998	4,998	0
Random tree decompositions (main reported set)	10,950	10,432	518	10,432	10,432	0
Aggregate	**16,680**	**15,430**	**1,250**	**15,430**	**15,430**	**0**

Table 10.1: Summary of external-reference agreement results and run outcomes for the main reported benchmark set.

in Table 10.1). These decomposition-based runs used eight workers per task, the same 1200-second per-instance timeout, and Slurm limits of 24:00:00 wall time, 8 CPUs, and 64 GiB of memory. Logs and results were stored per query so that reruns, comparisons, and discrepancy checks could be traced back to a single manifest entry.

10.4 Agreement results

Table 10.1 summarizes all run sets. Comparable rows are those for which the external reference and TreeWidzard return directly comparable outcomes. Inconclusive rows are `unknown`/`timeout` outcomes and are reported as incomplete computations, not correctness failures. These statuses reflect fixed resource budgets for the cluster runs, not semantic disagreement between TreeWidzard and its external reference.

Overall, 15,430 of the 16,680 processed rows in the main reported benchmark set are directly comparable, i.e., about 92.5% of the reported rows. In the random-graph benchmark set, 4,998 of 5,730 rows are directly comparable, i.e., about 87.2%; in the random tree-decomposition benchmark set, 10,432 of 10,950 rows are comparable, i.e., about 95.3%. These percentages make clear that the aggregate agreement rate is a statement about the comparable subset, not about every processed row. The remaining `timeout` and `unknown` rows reflect the fixed per-instance timeout and cluster resource limits from Section 10.3; they are incomplete runs, not semantic disagreement between the DP implementation and its external reference. For the Vertex counting query, the verdict comparison uses the invariant value emitted by the TreeWidzard DP run rather than the boolean accept-as-integer final value, so that the SAT-side numeric vertex count and the DP-side invariant are compared on the same footing.

The key outcome is that both reported run sets are mismatch-free on all comparable rows. Additionally, on the supplementary nauty-generated suite of all non-isomorphic graphs up to eight vertices, no disagreement was observed between TreeWidzard and the corresponding external reference. Across all run sets, rerun logs report zero execution

errors.

10.5 What the results establish about the implemented DP-cores

These data support a limited implementation-level conclusion. The point of this chapter is to check the implementation correctness of a selected set of DP-cores and, at the same time, to test that the corresponding fixed-instance TreeWidzard procedure behaves consistently end to end. On the reported fixed-instance benchmark suites in Table 10.1, the selected TreeWidzard implementations of these DP-cores agree with their external reference implementations on all comparable rows. For all benchmarked properties except Vertex counting, this reference is the SAT-based reference implementation from Appendix E; Vertex counting is checked against direct node counting. This exercises the fixed-instance DP execution procedure and the comparison harness used in the experiments, with witness decoding serving as an additional per-instance check on positive outcomes when such witnesses are available.

Appendix E matters because it fixes the SAT-based reference side of the comparison separately from the DP implementation for the SAT-covered properties. So this is not merely an internal TreeWidzard rerun. It is a comparison between the implemented DP-cores and independently written external references for the same fixed-instance queries. When satisfiable outcomes are available, the harness attempts to check the corresponding witnesses as well, although the present chapter does not break those witness checks out into a separate aggregate table.

The limits are straightforward. This chapter does not constitute full formal verification of the implementation. UNSAT outcomes are still trusted as solver output rather than replay-checked from proof logs, and the claim concerns only the benchmarked family set reported in Table 10.1, not the entire catalog of implemented DP-cores.

Part VI

Conclusion and Future Directions

Chapter 11

Conclusion and Future Directions

This thesis develops an ITD-centered, modular, algorithm-first framework for width-based reasoning on graphs. The central idea is to make decomposition-based dynamic programming explicit and reusable by packaging it as a DP-core: a semantic object that fixes a witness representation, constructor-wise transitions, cleaning behavior, and acceptance or invariant extraction. Once bounded-width graphs are represented by automata-validated decomposition terms and properties are represented by DP-cores, the same machinery supports two complementary modes of reasoning. In model-checking mode, a single validated decomposition is evaluated bottom-up. In theorem-proving mode, and under the hypotheses that make the bounded-width search space finite, the term space can be explored systematically, yielding either a concrete counterexample within the chosen width bound or a fixed-width guarantee that no such counterexample exists in that regime. The thesis complements this algorithmic perspective with a systematic correctness framework, an operational implementation of these workflows, and empirical cross-validation of the fixed-instance DP execution path against independent external references, mostly SAT encodings.

The broader contribution is best understood as a practical middle layer between two established extremes. On one side lie fully generic logic-to-automaton approaches, which offer strong expressiveness on bounded width but often obscure the witness design and the operational structure that matter in implementations. On the other side lie solver-only encodings, which can be powerful in practice but do not by themselves provide the same decomposition-aware semantics, modular reuse, or internal search workflows. The DP-core viewpoint keeps the algorithmic content visible: witness representations are exposed, transitions are inspectable, and optimization decisions such as cleaning, canonization, and premise-aware pruning become first-class parts of the reasoning system rather than accidental implementation details. This is what makes it possible to move from isolated width-based algorithms to an operational implementation of bounded-

width automated reasoning workflows.

11.1 Limitations

The main limitation of the framework is inseparable from its main strength: it is fundamentally parameterized by structural width. All exact bounded-width methods ultimately face a steep dependence on the width bound, and the present framework is no exception. Even when the graph property itself is conceptually simple, the reachable state space of cleaned witness sets can grow rapidly with width. The difficulty is amplified by density, because dense bags induce larger local interactions and hence larger intermediate witness sets. The completeness results of the ATP engine are therefore completeness results for fixed width, not practical tractability guarantees once the width grows. Search control improves the feasible frontier substantially, but it does not remove the underlying worst-case explosion.

A second limitation is the dependence on decomposition availability and decomposition quality. In fixed-instance model checking, the framework assumes that an appropriate decomposition term is already available or can be supplied by an external preprocessing step. In practice, constructing good decompositions is itself algorithmically nontrivial, and poor decomposition quality can dominate the running time of otherwise well-designed cores. This means that the framework should not be viewed as a self-contained solution to all bounded-width computation. Rather, it is a reasoning layer that sits on top of a decomposition pipeline and inherits some of that pipeline's strengths and weaknesses. The current operational development is especially strong in the instructive tree-decomposition setting, but its practical effectiveness still depends on how good the input decomposition is for the property being tested.

A third limitation concerns expressiveness at the level that matters operationally, namely witness design. DP-cores are only as practical as the witness summaries they maintain. Many natural graph properties admit compact local summaries on bounded width, and these are the cases where the framework is at its best. For other properties, no compact witness systems are currently known whose size and multiplicity remain well controlled under constructor composition. In such cases, the DP-core formalism remains semantically meaningful, but the associated exploration may become too large to serve as a practical ATP method at modest widths. Thus, although the framework is general as a semantic interface, its effective scope is determined by the existence of problem representations whose witness complexity remains manageable. This is a design issue, not merely an implementation issue.

There are also limits to the current search reductions. State canonization is highly effective because it targets a structured and recurrent source of redundancy, namely

symmetries induced by bag-label permutations. But it does not automatically quotient by all graph symmetries, nor does it solve the more general problem of isomorphism reduction for arbitrary partial graphs. Likewise, premise-based pruning is powerful when the hypothesis side of an implication has the right monotonicity properties, especially in subgraph-closed settings, but it is not a universal pruning principle. The present reductions therefore show that semantics-aware search control can transform feasibility in practice, yet they should be understood as important first reductions rather than as a complete theory of state-space collapse.

Finally, the trust story, while substantially stronger than that of an opaque experimental prototype, still has clear boundaries. The framework produces explicit counterexample graphs when an implication fails, and the thesis cross-validates the fixed-instance DP execution path against independent external references, mostly SAT encodings, on retained benchmark suites, which strengthens confidence in the implementation. Even so, this is not the same as full formal verification. Trust still rests on the correctness of the implemented DP-cores and the surrounding preprocessing and execution harness. In particular, UNSAT outcomes on the SAT side remain trusted as solver results unless proof logs are checked, and empirical agreement is evidence of implementation conformance, not a replacement for formal certification.

11.2 Outlook

The most immediate research direction is to extend the current ITD-centered operational framework to richer decomposition classes and broader width measures. The representation interface of the thesis was deliberately formulated for general treelike decomposition classes; Chapter 3 formulates that interface, and Appendix C shows that clique-width fits it at the representation level, even though the later semantic and implementation development remains centered on instructive tree decompositions. This makes clique-width a particularly natural next target. An extension in that direction would not be a cosmetic generalization: it would require a full operational account of DP-core semantics over clique-width terms, corresponding recognizer automata, and a proof-engine layer adapted to the label-based operators of clique-width expressions.

Such an extension would also be conceptually valuable. Treewidth-based dynamic programming is most natural when locality is expressed through bags and separators; clique-width changes the underlying programming idiom by replacing bag operations with label-based constructors such as disjoint union, relabeling, and edge creation between labels. A genuine clique-width version of the framework would therefore test how robust the DP-core perspective really is: can witness design, cleaning, canonization, and theorem proving be reformulated in a way that is native to label-based graph

generation rather than merely translated from bag-based semantics? If successful, this would broaden the range of graph classes accessible to the framework and would show that the thesis ideas are not tied to one decomposition syntax but to a more general principle of automata-validated structural reasoning.

A second direction is to strengthen the specification layer that sits above individual cores. At present, the framework already supports useful Boolean and invariant-valued combinators, and this compositionality is one of its main practical advantages. A natural next step is to design a higher-level front end in which users specify properties declaratively and the system compiles them into efficient DP-cores or combinations of DP-cores. The challenge is to gain usability without losing the explicit witness engineering that makes the framework controllable and optimizable. In other words, the goal should not be to replace algorithmic DP with a black-box compiler, but to build a specification layer that preserves the algorithm-first character of the thesis while reducing the amount of bespoke low-level core design required from the user.

A third family of problems concerns stronger search-space reduction. The current thesis demonstrates that canonization and premise pruning are decisive, but there is substantial room for more general reductions. One promising direction is deeper canonicalization that approaches full isomorphism reduction for partial graphs, not merely symmetry under bag relabeling. Another is the development of pruning principles beyond subgraph closure, perhaps using dominance relations tailored to individual witness domains, automata minimization ideas, or sound abstract-interpretation-style over-approximations that can rule out unproductive frontiers without compromising correctness. More generally, one would like a theory of reduction principles that is as systematic as the current theory of DP-core correctness, so that search control can be designed and justified at the same semantic level as the cores themselves.

A fourth direction is scalability at the systems level. The ATP engine explores large frontier sets, often in a breadth-first style, and this naturally suggests parallel and distributed execution. Restartable checkpoints, frontier caching, and incremental reuse of previously explored states could make the framework much more suitable for long-running computational investigations. This is especially relevant in conjecture-driven workflows, where one often refines a premise, changes a numeric threshold, or tests a nearby statement after an earlier run. In such settings, it is wasteful to restart exploration from scratch when much of the previously computed state space remains semantically relevant. A mature implementation should therefore support not only parallel exploration but also incremental reasoning across related ATP tasks.

The long-term direction that would most strengthen the foundational status of the work is formal certification in a proof assistant. The thesis already develops a systematic local-to-global proof methodology for DP-core correctness, and it already distinguishes

clearly between semantic guarantees and empirical implementation evidence. This makes the work well suited for a formalization effort. A fully certified version of the framework would formalize the semantics of the decomposition language, the notion of coherence, the constructor-wise proof rule underlying DP-core correctness, and the correctness of the individual catalog cores. It would then go further and formalize the proof engine itself: the exploration semantics, the soundness of canonization and premise pruning, the extraction of counterexamples, and the checking of replayable bounded-width support records. In that setting, one could certify not only that a given DP-core expresses the intended graph property, but also that the ATP engine manipulates such cores correctly.

This proof-assistant direction also suggests a more modular trust architecture. Rather than trusting a large implementation monolith, one could export proof artifacts that are independently checkable by a small kernel. Positive outcomes could be accompanied by certificates whose validity is checked formally, while negative outcomes could be tied to explicit counterexample graphs and verified reconstruction paths. When external oracles such as SAT solvers are used, one could require proof-producing solver output and formally checked proof replay. The resulting system would have a much smaller trusted computing base and would better align with the thesis's general emphasis on explicit evidence. In the strongest version of this agenda, the DP-core catalog and the proof engine would live in the same formal environment, so that extensions to the library come with machine-checked semantic guarantees by construction.

A final practical direction is simply to grow the library of well-justified cores and to test the framework on more case studies. The Reed-style bounded-width case study already demonstrates that the framework can support end-to-end investigation of nontrivial conjecture templates, complete with search reductions and explicit search workflows. But one case study is not a full map of the method's scope. Additional examples are needed to determine which kinds of graph statements are especially well matched to width-bounded ATP, which require new witness ideas, and which call for alternative decomposition representations. Expanding the catalog is therefore not only a matter of coverage; it is also how the framework will discover its own natural frontiers.

The overall lesson of the thesis is therefore both modest and ambitious. It is modest because the results are carefully width-bounded, because the operational development is still centered on the ITD setting, and because the software validation is empirical rather than fully formal. But it is ambitious because it shows that bounded-width automated reasoning can be organized around a reusable semantic layer that is expressive enough to encode interesting properties, algorithmic enough to admit real optimization, and explicit enough to produce inspectable evidence. Read at the right level, the thesis makes a contribution on three fronts: a formal semantics-and-correctness layer for DP-cores, an algorithmic ATP layer with completeness guarantees in the fixed-width

regimes covered by the ATP theorems, and an empirical implementation layer supported by independent cross-checking. The next stage is to broaden that middle layer to richer width measures such as clique-width, to scale it with stronger reductions and better systems-level support, and ultimately to place the whole framework on certified foundations in a proof assistant.

Bibliography

K. A. Abrahamson, J. A. Ellis, M. R. Fellows, and M. E. Mata. On the complexity of fixed parameter problems. In *30th Annual Symposium on Foundations of Computer Science*, pages 210–215. IEEE, 1989.

M. Aouchiche and P. Hansen. A survey of automated conjectures in spectral graph theory. *Linear Algebra and its Applications*, 432(9):2293–2322, 2010. doi: 10.1016/j.laa.2009.06.015.

M. Aouchiche, G. Caporossi, P. Hansen, and M. Laffay. Autographix: A survey. *Electronic Notes in Discrete Mathematics*, 22:515–520, 2005. doi: 10.1016/j.endm.2005.06.090.

S. Arnborg and A. Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

S. Arnborg, A. Proskurowski, and D. G. Corneil. Forbidden minors for the class of partial 3-trees. *Journal of Graph Theory*, 14(3):291–301, 1990.

S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

M. Bannach and S. Berndt. Practical access to dynamic programming on tree decompositions. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112, pages 6:1–6:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

J. Barat and P. Hajnal. On the pathwidth of chordal graphs. *Discrete Mathematics*, 231 (1-3):37–47, 2001.

D. W. Barnette. Conjecture 5. In W. T. Tutte, editor, *Recent Progress in Combinatorics: Proceedings of the Third Waterloo Conference on Combinatorics, May 1968*, page 343. Academic Press, New York, 1969.

J. Barát, G. Joret, and D. R. Wood. On the pathwidth of almost complete graphs. *Journal of Graph Theory*, 71(3):283–292, 2012.

J. Baste. Dynamic programming on tree decompositions as a framework for constrained path problems. *Discrete Mathematics & Theoretical Computer Science*, 26(1), 2024.

J. Baste, P. Fouilhoux, M. Matamala, M. d. Oliveira, C. Paul, and C. Rapine. Dynamic programming on tree decompositions with flow computation: Algorithms for some path problems. *Artificial Intelligence*, 313:103788, 2022.

R. Belmonte and M. Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013.

D. Bienstock, N. Robertson, P. Seymour, and R. Thomas. On the complexity of testing for treewidth. *SIAM Journal on Computing*, 20(5):865–873, 1991.

H. L. Bodlaender. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 1(1):43–57, 1988.

H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11(1–2):1–21, 1993.

H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, 1998. doi: 10.1016/S0304-3975(97)00228-4.

H. L. Bodlaender. Discovering treewidth. *Electronic Notes in Discrete Mathematics*, 21: 77–86, 2005.

H. L. Bodlaender and F. V. Fomin. Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, 349(1):22–30, 2005.

H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.

H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I: Upper bounds. *Information and Computation*, 208(3):259–275, 2010. doi: 10.1016/j.ic.2009.03.008. URL `https://doi.org/10.1016/j.ic.2009.03.008`.

H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. Approximating treewidth and pathwidth. *Algorithmica*, 75(3):411–429, 2016.

M. Bonamy, T. Perrett, and L. Postle. Colouring graphs with sparse neighbourhoods: Bounds and applications. *Journal of Combinatorial Theory, Series B*, 155:278–317, 2022. doi: 10.1016/j.jctb.2022.01.009.

J. A. Bondy and U. S. R. Murty. *Graph theory*. Springer Publishing Company, Incorporated, 2008.

R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1–6):555–581, 1992.

N. Brettell, J. Horsfield, A. Munaro, G. Paesani, and D. Paulusma. Bounding the mim-width of hereditary graph classes. *Journal of Graph Theory*, 99(1):117–151, 2022. doi: 10.1002/jgt.22730.

T. L. Brewster, M. J. Dinneen, and V. Faber. A computational attack on the conjectures of graffiti: New counterexamples and proofs. *Discrete Mathematics*, 147(1–3):35–55, 1995. doi: 10.1016/0012-365X(94)00227-A.

G. Brinkmann and B. D. McKay. Fast generation of planar graphs. *MATCH Communications in Mathematical and in Computer Chemistry*, 58(2):323–357, 2007.

R. L. Brooks. On coloring nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941. doi: 10.1017/S030500410002168X.

B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412(39):5187–5204, 2011. doi: 10.1016/j.tcs.2011.05.022.

A. Bundy. The use of explicit plans to guide inductive proofs. In *9th International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120. Springer, Berlin, Heidelberg, 1988. doi: 10.1007/BFb0012826.

B. Burgstaller, J. Blieberger, and B. Scholz. On the tree width of Ada programs. *International Conference on Reliable Software Technologies*, pages 78–90, 2004.

A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2008. URL `https://inria.hal.science/hal-03367725`. Online resource (accessed May 2026).

G. K. Conrado, A. K. Goharshady, and C. K. Lam. The bounded pathwidth of control-flow graphs. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2): 292:930–292:959, 2023. doi: 10.1145/3622807. URL `https://doi.org/10.1145/3622807`.

D. G. Corneil and U. Rotics. On the relationship between cliquewidth and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

D. G. Corneil, M. Habib, J.-M. Lanligne, B. Reed, and U. Rotics. Polynomial time recognition of clique-width$\leq$ 3 graphs. In *LATIN 2000: Theoretical Informatics: 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000 Proceedings 4*, pages 126–134. Springer, 2000.

B. Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

B. Courcelle and I. Durand. Fly-automata, their properties and applications. In *Implementation and Application of Automata*, volume 6807 of *Lecture Notes in Computer Science*, pages 264–272. Springer, 2011. doi: 10.1007/978-3-642-22256-6_24.

B. Courcelle and I. Durand. Automata for the verification of monadic second-order graph properties. *Journal of Applied Logic*, 10(4):368–409, 2012. doi: 10.1016/j.jal.2011.07.001.

B. Courcelle and I. Durand. Computations by fly-automata beyond monadic second-order logic. *Theoretical Computer Science*, 619:32–67, 2016. doi: 10.1016/j.tcs.2015.12.026. URL `https://doi.org/10.1016/j.tcs.2015.12.026`.

B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, 2012.

B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993.

B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

D. M. Cvetković and I. Pevac. Man-machine theorem proving in graph theory. *Artificial Intelligence*, 35(1):1–23, 1988. doi: 10.1016/0004-3702(88)90030-6.

M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015. doi: 10.1007/978-3-319-21275-3. URL https://doi.org/10.1007/978-3-319-21275-3.

M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960. doi: 10.1145/321033.321034.

M. de Oliveira Oliveira and S. Urmian. State canonization and early pruning in width-based automated theorem proving. https://arxiv.org/abs/2605.11025, 2026a. Full version of de Oliveira Oliveira and Vadiee [2024]. 66 pages, 2 figures, 10 tables.

M. de Oliveira Oliveira and S. Urmian. TreeWidzard: An engine for width-based dynamic programming and automated theorem proving. https://arxiv.org/abs/2605.09732, 2026b. Full version, 36 pages, 6 figures.

M. de Oliveira Oliveira and F. Vadiee. From width-based model checking to width-based automated theorem proving. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(5):6297–6304, 2023. doi: 10.1609/AAAI.V37I5.25775. URL https://doi.org/10.1609/aaai.v37i5.25775.

M. de Oliveira Oliveira and F. Vadiee. State canonization and early pruning in width-based automated theorem proving. In J. Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, volume 299 of *LIPIcs*, pages 33:1–33:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024. doi: 10.4230/LIPICS.FSCD.2024.33. URL https://doi.org/10.4230/LIPIcs.FSCD.2024.33.

E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):292–302, 2008.

R. Diestel. A short proof of the path-width theorem. *Journal of Combinatorial Theory, Series B*, 64(2):310–313, 1995.

J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970. doi: 10.1016/S0022-0000(70)80041-1. URL https://doi.org/10.1016/S0022-0000(70)80041-1.

R. G. Downey. A parameterized complexity tutorial. In A.-H. Dediu and C. Martín-Vide, editors, *Language and Automata Theory and Applications*, volume 7183 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2012.

R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995a.

R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1–2):109–131, 1995b.

R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, New York, NY, 1999. ISBN 978-0-387-94883-6. doi: 10.1007/978-1-4612-0515-9. URL `https://doi.org/10.1007/978-1-4612-0515-9`.

R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

Z. Dvořák and K.-i. Kawarabayashi. Triangle-free graphs of tree-width t are ⌈ (t+ 3)/ 2 ⌉-colorable. *European Journal of Combinatorics*, 66:95–100, 2017.

H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, second edition, 1995.

S. L. Epstein. On the discovery of mathematical theorems. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI 1987)*, pages 194–197. Morgan Kaufmann, 1987.

S. L. Epstein. Learning and discovery: One system's search for mathematical knowledge. *Computational Intelligence*, 4(1):42–53, 1988. doi: 10.1111/j.1467-8640.1988.tb00089.x.

R. Fagin. Finite-model theory—a personal perspective. *Theoretical Computer Science*, 116:3–31, 1993.

S. Fajtlowicz. On conjectures of graffiti. *Discrete Mathematics*, 72(1–3):113–118, 1988. doi: 10.1016/0012-365X(88)90199-9.

M. R. Fellows. Parameterized complexity: the main ideas and connections to practical computing. In R. Fleischer, B. M. E. Moret, and E. M. Schmidt, editors, *Experimental Algorithmics: From Algorithm Design to Robust and Efficient Software*, pages 51–77. Springer, 2002.

M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width is NP-complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, 2009. doi: 10.1137/070687256.

M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, 2 edition, 1996. doi: 10.1007/978-1-4612-2360-3.

J. Flum and M. Grohe. Parameterized complexity theory. *Springer Science & Business Media*, 2006.

J. Flum and M. Grohe. Parameterized complexity and logic. In L. Ong, editor, *Computer Science Logic*, volume 3634 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.

F. V. Fomin, P. A. Golovach, D. Lokshtanov, and S. Saurabh. Clique-width: on the price of generality. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 825–834. SIAM, 2009.

M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM*, 48(6):1184–1206, 2001. doi: 10.1145/504794.504798.

M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1–3):3–31, 2004. doi: 10.1016/j.apal .2004.01.007.

S. Garfunkel and H. Shank. On the undecidability of finite planar graphs. *The Journal of Symbolic Logic*, 36(1):121–126, 1971.

S. Garfunkel and H. Shank. On the undecidability of finite planar cubic graphs. *The Journal of Symbolic Logic*, 37(3):595–597, 1972.

P. A. Golovach, D. Paulusma, and E. J. van Leeuwen. Parameterized complexity of finding a 2-spider in a graph. *Theoretical Computer Science*, 748:98–109, 2018.

M. Grohe. Algorithmic meta theorems for sparse graph classes. In E. W. Mayr, editor, *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, pages 1–14. Springer, 2014.

M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM*, 64(3):17:1–17:32, 2017. doi: 10.1145/3051095.

J. Gustedt, O. A. Mæhle, and J. A. Telle. The treewidth of Java programs. In *International Workshop on Algorithm Engineering and Experiments*, pages 86–97. Springer, 2002.

H. Hadwiger. Über eine klassifikation der streckenkomplexe. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 88:133–142, 1943.

J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

R. Impagliazzo and R. Paturi. On the complexity of k-SAT. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 337–346, 2001.

R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

L. Jaffke, O.-j. Kwon, T. J. F. Strømme, and J. A. Telle. A unified polynomial-time algorithm for feedback vertex set on graphs of bounded mim-width. *SIAM Journal on Discrete Mathematics*, 33(3):1783–1802, 2019.

J. Jooken. Computer-assisted graph theory: a survey, 2025. URL https://arxiv.org/abs/2508.20825. Preprint.

M.-J. Kang, O.-j. Kwon, T. J. F. Strømme, and J. A. Telle. Parameterized complexity of finding a spanning tree with minimum reload cost. *Theoretical Computer Science*, 695:1–14, 2017.

R. J. Kang. On a conjecture of Reed. https://staff.fnwi.uva.nl/j.r.kang/2022DMVKang.pdf, 2022. DMV Annual Meeting slides (accessed January 2026).

K.-i. Kawarabayashi and B. Reed. Hadwiger's conjecture is decidable. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 445–454, 2009.

A. D. King and B. A. Reed. Claw-free graphs, skeletal graphs, and a stronger conjecture on
omega,
Delta, and
chi. *Journal of Graph Theory*, 78(3):157–194, 2015. doi: 10.1002/jgt.21797.

A. D. King and B. A. Reed. A short proof that χ can be bounded ε away from $\delta+1$ toward ω. *Journal of Graph Theory*, 81(1):30–34, 2016. doi: 10.1002/jgt.21858.

A. D. King, B. A. Reed, and A. Vetta. An upper bound for the chromatic number of line graphs. *European Journal of Combinatorics*, 28(8):2182–2187, 2007.

N. G. Kinnersley and M. A. Langston. Obstruction set isolation for the pathwidth of graphs. *Journal of Algorithms*, 16(2):239–254, 1994.

T. Kloks. *Treewidth: Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

J. Kneis, A. Langer, and P. Rossmanith. A practical algorithm for MSO model checking. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 108–119. Springer, 2009.

P. G. Kolaitis. Reflections on finite model theory. In *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 257–269. IEEE, 2007. doi: 10.1109/LICS.2007.39.

L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In *Computer Aided Verification*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, Berlin, Heidelberg, 2013. doi: 10.1007/978-3-642-39799-8_1.

F. Krause, L. Larisch, and F. Salfelder. Estimating the treewidth of sparse graphs and its applications to the minimum fill-in problem. *arXiv preprint arXiv:2004.03877*, 2020.

S. Kreutzer. Algorithmic meta-theorems. Technical Report TR08-147, Electronic Colloquium on Computational Complexity, 2008.

A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. In *Computer Science–Theory and Applications*, pages 52–66. Springer, 2014.

L. Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-662-07003-1.

W. McCune. Mace4 reference manual and guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, 2003.

B. D. McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26(2):306–324, 1998. doi: 10.1006/JAGM.1997.0898. URL `https://doi.org/10.1006/jagm.1997.0898`.

B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014. doi: 10.1016/j.jsc.2013.09.003. URL `https://doi.org/10.1016/j.jsc.2013.09.003`.

M. Molloy. The list chromatic number of graphs with small clique number. *Journal of Combinatorial Theory, Series B*, 134:264–284, 2019.

R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2006.

S.-i. Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1):10:1–10:20, 2008. doi: 10.1145/1435375.1435385.

S.-i. Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017.

S.-i. Oum and P. D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

B. Reed. ω, δ, and χ. *Journal of Graph Theory*, 27(4):177–212, 1998.

N. Robertson and P. D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

N. Robertson and P. D. Seymour. Graph minors. V. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986a. doi: 10.1016/0095-8956(86)90030-4.

N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986b.

N. Robertson and P. D. Seymour. Graph minors. X. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991.

N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 62(2):323–348, 1994.

J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965. doi: 10.1145/321250.321253.

J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*, volume 1–2. Elsevier and MIT Press, 2001.

I. Schiermeyer. Chromatic number of P_5-free graphs: Reed's conjecture. *Discrete Mathematics*, 339(7):1940–1943, 2016. doi: 10.1016/j.disc.2015.11.020.

S. Schulz. E — a brainiac theorem prover. *AI Communications*, 15(2–3):111–126, 2002.

D. Seese. The structure of the models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53:169–195, 1991.

P. D. Seymour. A shorter proof of the path-width theorem. *arXiv preprint arXiv:2301.03992*, 2023.

C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP*, pages 827–831, 2005.

A. Takahashi, S. Ueno, and Y. Kajitani. Minimal vertex separators of chordal graphs. *Discrete Applied Mathematics*, 50(1):81–87, 1994.

J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

M. Thorup. All structured programs have small treewidth and good register allocation. *Information and Computation*, 142(2):159–181, 1998.

G. S. Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1937. doi: 10.1112/plms/s2-42.1.230.

W. T. Tutte. A contribution to the theory of chromatic polynomials. *Canadian Journal of Mathematics*, 6:80–91, 1954.

W. T. Tutte. On the algebraic theory of graph colorings. *Journal of Combinatorial Theory*, 1(1):15–50, 1966.

S. Urmian. PhD thesis experimental artifact. `https://github.com/mlgorithm/phd-thesis-experiments`, 2025. Artifact repository.

M. Vatshelle. New width parameters of graphs. *Doctoral dissertation, University of Bergen*, 2012.

C. Weidenbach, B. Afshordel, U. Brahm, C. Cohrs, T. Engel, E. Keen, C. Theobalt, and D. Topić. System description: SPASS version 1.0.0. In *Automated Deduction — CADE-16*, volume 1632 of *Lecture Notes in Computer Science*, pages 378–382. Springer, Berlin, Heidelberg, 1999. doi: 10.1007/3-540-48660-7_34.

C.-Q. Zhang. *Integer Flows and Cycle Covers of Graphs*. Marcel Dekker, New York, 1997.

Appendices: Reader's Guide

The appendices below (Appendix A–Appendix E) provide reference material for the main chapters:

- **Appendix A (Notation and Conventions)** fixes recurring notation and standing assumptions (including graph-model conventions).
- **Appendix B (Problem Definitions)** collects the problem statements used for DP-cores and SAT encodings, and additional properties referenced in automated theorem proving.
- **Appendix C (Clique-Width)** records the clique-width background and its treelike-decomposition presentation.
- **Appendix D (Supplementary DP-core Correctness Proofs)** contains deferred DP-core instantiations, supplementary framework notes, and the boundary-equivalence lower-bound arguments.
- **Appendix E (SAT Encodings)** presents the SAT encodings used as external cross-check oracles.

Appendix A

Notation and Conventions

This appendix collects notation and conventions that recur throughout the thesis. When a symbol is overloaded in the literature, we state the convention used here explicitly.

A.1 Standing Assumptions

We collect default standing assumptions that apply unless stated otherwise.

Graphs. Unless stated otherwise, graphs in problem statements and benchmarks are finite, undirected, simple, and loopless. In the ITD semantics and proof-engine chapters, realized graphs $\mathcal{G}(\tau)$ are treated as undirected multigraphs (still loopless), and join operations use multiset union of edges. Parallel edges may therefore arise.

A.2 Sets and Asymptotics

Notation	**Meaning / convention**
$\mathbb{N}$	Natural numbers, with $0 \in \mathbb{N}$.
$\mathbb{Z}_c$	Integers modulo c. We write $\mathbb{Z}_c \setminus \{0\}$ for the nonzero residues.
$[n]$	The set $\{1, \ldots, n\}$.
$\mathcal{P}(A)$	Power set of A.
$\mathcal{P}_{\text{fin}}(A)$	Finite subsets of A.
$\mathcal{O}^*(\cdot)$	Suppresses polynomial factors in the input size.
log	Base 2, unless stated otherwise.

A.3 Graphs

Notation	Meaning / convention
$G = (V, E)$	A finite undirected graph (simple and loopless unless stated otherwise).
$G = (V, E, \rho)$	Graph with explicit incidence relation $\rho \subseteq E \times V$ (edge-to-endpoint incidence). Some sources swap the order and write $\rho \subseteq V \times E$; we treat these as the same incidence relation with arguments swapped.
GRAPHS	Ambient class of finite graph objects. By default this means simple graphs; in the ITD semantics and width-based ATP chapters it denotes loopless undirected multigraph incidence structures unless a simplicity promise is stated explicitly.
$G[X]$, $G - X$	Induced subgraph on X. Vertex deletion $G[V \setminus X]$.
$N_G(v)$, $N_G(X)$, $N_G[X]$	Neighborhood of v. Union of neighborhoods over X. Closed neighborhood of X.
$\deg_G(v)$, $\Delta(G)$	Degree of v. Maximum degree.
$\kappa(G)$, $\lambda(G)$	Vertex connectivity. Edge connectivity.
$\omega(G)$, $\chi(G)$, $\alpha(G)$	Clique number. Chromatic number. Independence number.
$G \simeq H$	Graph isomorphism.
$\uplus$	Disjoint union (components kept distinct).
$tw(G)$, $pw(G)$	Treewidth. Pathwidth.

A.4 Ranked Alphabets and Terms

Notation	Meaning / convention
(Σ, r)	Ranked alphabet with arity function $r : \Sigma \to \mathbb{N}$ (written r in the main text).
Σ_p	Symbols of arity p.
$\mathrm{Terms}(\Sigma)$, $\mathcal{C}(\Sigma)$	Ground terms. One-hole contexts over Σ.
$\lvert\tau\rvert$, $\textsc{Height}(\tau)$	Term size. Term height.
$\mathrm{pos}(\tau) \subseteq \mathbb{N}^*$	Positions in a term.
$\mathtt{root}(\tau)$	Root symbol of a term.

A.5 Automata, ITD Semantics, and DP-cores

This section records notation for automata, instructive decompositions, and DP-core evaluation.

A.5.1 Automata and transitions

Notation	Meaning / convention
$\mathcal{A}_k.\delta$, $\mathcal{D}[k].\Delta$	Transition relation/family (automata and DP-cores). DP-core transitions use Δ, while maximum degree is $\Delta(G)$.
$\nu(k)$	Witness-level state complexity (number of useful witnesses) of a DP-core at width k.
$\delta_{\mathrm{D}}(k)$	Deterministic witness-set state complexity (number of useful witness sets) of a DP-core at width k.

A.5.2 ITD semantics and boundaried graphs

Notation	Meaning / convention
Σ_k, ITD_k	Width-k instructive alphabet. Well-formed ITD terms.
$\overline{\Sigma}_k$	Bag-annotated instructive alphabet used in the homomorphic semantics of Chapter 9.
$[k+1]$	Active-label universe.
Θ_k	Injective boundary maps into $\mathbb{N}$ used in ITD semantics.
$B(\tau)$	Top bag (join bag) associated with a term.
$(G_1, \theta_1) \oplus_B (G_2, \theta_2)$	Boundary gluing of boundaried graphs along B. In clique-width expressions, $\oplus$ denotes disjoint union.
$\mathcal{G}(\tau)$	Realized graph of τ in the chosen semantics.

A.5.3 DP-cores and dynamization

Notation	Meaning / convention
$\mathcal{D}[k]$	DP-core at width k (alphabet, witness universe, final predicate, transitions, cleaning, invariants).
$\Gamma[\mathcal{D}, k](\tau)$	Witness set computed bottom-up on a term.
$\mathsf{Acc}(\mathcal{D}[k])$	Accepted terms (some root witness is final).

Notation crosswalk. Throughout the thesis, mathematical prose uses graph-theoretic names of properties and invariants. Typical examples are *c-colorability*, *Hamiltonian cycle*, and *maximum degree at least d*. Code-style names such as `ChromaticNumber_AtMost(c)` and `HasMultipleEdges()` are reserved for implementation identifiers, property files, and validation tables. In Chapter 9 and Appendix D, the generic local semantic judgment of a core is written $\mathsf{Interp}_{\mathcal{D}}(\sigma, w)$. Chapter 4 sometimes retains shorter per-core predicate names such as $\mathsf{P}^{\chi}_{c,k}$; these should be read as concrete instances of the same interpretation relation. Likewise, Chapter 4 presents the chromatic core via partition witnesses, whereas Chapter 9 uses a labeled total-map presentation of the same core and makes the translation explicit. We use US spellings in the mathematical prose (*coloring*, *realized*, *realization*); fixed macro names and software identifiers retain the spelling used by the implementation and cited source material.

Finality and bag arguments. Formally, Chapter 4 defines finality as a bag-aware predicate $\texttt{Final}_k(B, w)$ on a current bag $B \subseteq [k+1]$ and a witness w. In prose and in concrete core descriptions, this bag argument is often suppressed when it is fixed by the surrounding term or by the root condition, and one simply states the corresponding witness condition (for example, that every witness is final, or that a designated witness such as $U = \emptyset$ is final). When the current bag is part of the state space, we keep it explicit, as in the bag-aware fly-automaton formulation in Chapter 9, Proposition 7.

A.6 Overloaded Symbols

A handful of letters are reused across the thesis with locally distinct meanings, in line with standard practice in the underlying literature. We list them here so that the reader can resolve the meaning in any given chapter from its surrounding context. In all cases the meaning intended at any point is fixed by the chapter and the typographical surroundings (subscripts, decorations, function-call brackets); the table below is a navigation map, not a redefinition.

Notation	Meaning / convention
ρ	(i) Edge–vertex incidence relation $\rho \subseteq E \times V$ of a graph (Background; Section A.3 of this appendix). (ii) Bag relabeling action / relabel pair (ρ, ρ') on DP-core states (Chapter on Symmetry Breaking). (iii) An accepting run of a tree automaton over a term, written ρ (ATP and Premise Search chapters). (iv) The clique-width relabel operator $\rho_{i\to j}$ (Appendix on Clique-Width). The four uses never appear in the same paragraph.
σ	(i) A subterm of an abstract decomposition / ITD term, often σ_1, σ_2 for the children of a join (most chapters). (ii) The maximum-useful-witness-set complexity measure $\sigma(k, n)$ of a DP-core (Complexity Measures, when written as a function of two arguments). When σ appears bare or with an integer subscript indexing a child, it is a subterm; when it appears as $\sigma(\cdot, \cdot)$ it is a complexity measure.
α	(i) A vertex coloring $\alpha : V \to [c]$ (Chromatic core, Chapter on DP-cores). (ii) The independence number $\alpha(G)$ of a graph (Section A.3 of this appendix; SAT-encodings chapter). (iii) A SAT assignment, used as a generic letter for an assignment to Boolean variables (SAT-encodings chapter, where it never appears as a function of G).
Δ	(i) Maximum degree $\Delta(G)$ of a graph. (ii) DP-core transition family $\Delta_{k,a}$. The two uses are distinguished by the namespace prefix $\mathcal{D}[k].\Delta$ for the transition family and the function-call form $\Delta(G)$ for the graph invariant.

Notation	**Meaning / convention**
δ	(i) Tree-automaton transitions δ_a. (ii) The deterministic witness-set state complexity of a DP-core, written $\delta_{\mathrm{D}}(k)$ where the literal subscript D reads as "deterministic" and is used as a generic name when the core is fixed by context (e.g., the rows of Table 5.1); when several cores are simultaneously in scope we instead write $\delta_{\mathcal{D}}(k)$ with an explicit subscript naming the core. (iii) DP-core symbol-level transitions $\delta_{k,a}$ with two indices (width and constructor). All three uses carry distinguishing subscripts.
Σ_p vs Σ_k	In *ranked-alphabet* contexts, Σ_p denotes the symbols of arity p. In *ITD* contexts, Σ_k denotes the width-k instructive alphabet. The two indexings are distinguished by the variable name in the subscript (arity p vs. width k) and never appear simultaneously.
$\oplus$ vs. $\uplus$	Disjoint union of graphs is written $\uplus$ in the main text (Section A.3); the clique-width appendix follows the convention of that literature and writes the same operation as $\oplus$. In ITD semantics, $\oplus_B$ (with explicit boundary subscript) denotes boundary gluing along B and is a different operation.
k	Default meaning is the width parameter (treewidth or pathwidth bound) throughout the thesis. In the clique-width appendix only, k additionally denotes the label budget of a clique-width expression. The two specialisations agree on the qualitative role of k (a width-style budget) and differ only in their definition.

A.7 SAT Convention

Notation	Meaning / convention
literal	A Boolean variable x or its negation $\neg x$.
clause	A disjunction of literals.
CNF	A conjunction of clauses (conjunctive normal form).
satisfiable / UNSAT	A CNF is satisfiable if some assignment makes all clauses true; otherwise it is unsatisfiable.
3-CNF	Each clause has exactly 3 literals.

Appendix B

Problem Definitions

B.1 Notation

Unless stated otherwise, $G = (V, E)$ denotes a finite, undirected, loopless simple graph. For Multiple Edge Detection we allow multigraph inputs. For $X \subseteq V$, $G[X]$ is the induced subgraph on X, and $G - X := G[V \setminus X]$. For $v \in V$, $N_G(v)$ denotes the (open) neighborhood and $\deg_G(v) := |N_G(v)|$ the degree. For $X \subseteq V$, set $N_G(X) := \bigcup_{x \in X} N_G(x)$ and $N_G[X] := X \cup N_G(X)$. For $q \in \mathbb{N}$, $[q] := \{1, \ldots, q\}$. All paths and cycles are assumed to be simple (no repeated vertices) and their length is the number of edges.

Graph model convention. The SAT encodings and problem statements in this appendix are stated for simple graphs unless explicitly noted otherwise. By contrast, in the ITD semantics the realized graphs $\mathcal{G}(\tau)$ may contain parallel edges (Appendix A). When a property is intended to range only over simple graphs, this is enforced either by restricting the input suite or by conjoining the simplicity predicate `NOT HasMultipleEdges()` in property files. Implementation identifiers used in property files appear in the **Property-file core identifier** column of Table B.1. Table B.1 documents the identifier/encoding crosswalk for the implemented catalog as a whole. Chapter 10 reports empirical conformance results only for the selected benchmark suites used in that chapter.

B.2 Problems used as cores / SAT encodings

Problem / property	Property-file core identifier	SAT encoding
c-Colorability ($\chi(G) \leq c$)	`ChromaticNumber_AtMost(c)`	$\phi_{\mathrm{COL}}(G, c)$ (Appendix E)
Vertex Cover ($\leq t$)	`VertexCover(t)`	$\phi_{\mathrm{VC}}(G, t)$ (Appendix E)
Clique ($\geq \ell$, simple graphs)	`SimpleCliqueNumber_AtLeast(l)`	$\phi_{\mathrm{CLQ}}(G, \ell)$ (Appendix E)
Independence number $\alpha(G)$	`IndependentSet()` (invariant; threshold queries $\alpha(G) \geq t$)	$\phi_{\mathrm{IS}}(G, t)$ decides $\alpha(G) \geq t$ (Appendix E)
Hamiltonian Path / Cycle	`HamiltonianPath()`, `HamiltonianCycle()`	$\phi_{\mathrm{HP}}(G)$, $\phi_{\mathrm{HC}}(G)$ (Appendix E)
Perfect Matching	`PerfectMatching()`	$\phi_{\mathrm{PM}}(G)$ (Appendix E)
Feedback Vertex Set ($\leq t$)	`FeedbackVertexSet(t)`	$\phi_{\mathrm{FVS}}(G, t)$ (Appendix E)
Maximum Degree ($\geq d$)	`MaximumDegree_AtLeast(d)`	$\phi_{\mathrm{DEG}}(G, d)$ (Appendix E)
Multiple Edge Detection	`HasMultipleEdges()`	$\phi_{\mathrm{MULTI}}(G)$ (Appendix E)

Table B.1: Crosswalk between problem names in this appendix, property-file core identifiers, and the SAT encodings used for cross-checking.

Chromatic Number

Instance: A graph G.
Parameter: None.
Task: Compute the minimum integer q such that there exists a coloring $\gamma : V(G) \to [q]$ with $\gamma(u) \neq \gamma(v)$ for every edge $uv \in E(G)$.

c-Colorability

Instance: A graph G.
Parameter: An integer c.
Task: Does there exist a coloring $\gamma : V(G) \to [c]$ with $\gamma(u) \neq \gamma(v)$ for every edge $uv \in E(G)$?

Throughout the thesis, the mathematical property is called c-colorability (equivalently, $\chi(G) \leq c$); `ChromaticNumber_AtMost`(c) is the implementation-facing identifier of the corresponding DP-core in property files and experiments.

Clique

Instance: A graph G.
Parameter: An integer ℓ.
Task: Does there exist $K \subseteq V(G)$ with $|K| \geq \ell$ such that $\{u, v\} \in E(G)$ for all distinct $u, v \in K$?

Note (simplicity guard). The clique DP-core is used under a simplicity promise to avoid false positives caused by parallel edges. In property files, this is enforced by using `SimpleCliqueNumber_AtLeast(l)` or by conjoining `NOT HasMultipleEdges()`. **Note (parameter regime).** The DP-core proof in Appendix D.5 is stated for $\ell \geq 2$. The case $\ell = 1$ is trivial on simple graphs: G has a clique of size at least 1 if and only if

$V(G) \neq \emptyset$.

Vertex Cover

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \leq t$ such that $G - X$ is edgeless (equivalently, every edge has an endpoint in X)?

Independent Set

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \geq t$ such that $G[X]$ is edgeless?

In the implementation and DP-core catalog, `IndependentSet()` is the invariant-valued core computing $\alpha(G)$; the SAT cross-checks in Chapter 10 use fixed threshold queries of the form $\alpha(G) \geq t$.

Hamiltonian Path

Instance: A graph G.
Parameter: None.
Task: Does there exist a simple path P in G such that $V(P) = V(G)$?

Hamiltonian Cycle

Instance: A graph G.
Parameter: None.
Task: Does there exist a simple cycle C in G such that $V(C) = V(G)$?

Note (simple-graph comparison). The Hamiltonian-cycle DP-core in Appendix D.9 is interpreted on the default ITD multigraph semantics. Comparisons with the simple-graph SAT encoding in Appendix E therefore assume simple inputs or include the standard `NOT HasMultipleEdges()` guard.

Perfect Matching

Instance: A graph G.
Parameter: None.
Task: Does there exist $M \subseteq E(G)$ such that every vertex in $V(G)$ is incident to exactly one edge of M?

Feedback Vertex Set

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \leq t$ such that $G - X$ is a forest?

Maximum Degree at least d

Instance: A graph G.
Parameter: An integer $d \geq 1$.
Task: Does there exist $v \in V(G)$ with $\deg_G(v) \geq d$?

Multiple Edge Detection

Instance: An undirected multigraph G.
Parameter: None.
Task: Does G contain two distinct edges with the same unordered endpoints?

B.3 Additional standard problems referenced in the background

Dominating Set

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \leq t$ such that $N_G[X] = V(G)$?

Connected Dominating Set

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \leq t$ such that $G[X]$ is connected and $N_G[X] = V(G)$?

Connected Vertex Cover

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \leq t$ such that $G[X]$ is connected and $G - X$ is edgeless?

Connected Feedback Vertex Set

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \leq t$ such that $G[X]$ is connected and $G - X$ is a forest?

Cycle Packing

Instance: A graph G.
Parameter: An integer t.
Task: Does G contain t pairwise vertex-disjoint cycles?

Longest Path

Instance: A graph G.
Parameter: An integer t.
Task: Does G contain a simple path on at least t vertices?

Longest Cycle

Instance: A graph G.
Parameter: An integer t.
Task: Does G contain a simple cycle on at least t vertices?

MaxCut

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $A, B \subseteq V(G)$ with $A \cap B = \emptyset$ and $A \cup B = V(G)$ such that at least t edges have one endpoint in A and the other in B?

Odd Cycle Transversal

Instance: A graph G.
Parameter: An integer t.
Task: Does there exist $X \subseteq V(G)$ with $|X| \leq t$ such that $G - X$ is bipartite?

Steiner Tree

Instance: A graph G and terminals $T \subseteq V(G)$.
Parameter: An integer t.
Task: Does there exist a tree subgraph H of G such that $T \subseteq V(H)$ and $|E(H)| \leq t$?

B.4 Additional properties used in automated theorem proving

Connectedness (Conn)

Instance: A graph G.
Parameter: None.
Task: Is G connected? In this thesis, the empty graph is counted as connected, matching the convention used by the `Conn` DP-core in Appendix D.2.

Minimum Degree at most c ($\text{MinDeg}_{\leq c}$)

Instance: A graph G.
Parameter: An integer c.
Task: Does G contain a vertex $v \in V(G)$ with $\deg_G(v) \leq c$? (Equivalently, the minimum degree of G is at most c.)

Vertex Connectivity at most c ($\text{VConn}_{\leq c}$)

Instance: A graph G.
Parameter: An integer c.
Task: Does there exist a set $S \subseteq V(G)$ with $|S| \leq c$ such that $G - S$ is disconnected or has at most one vertex? (Equivalently, the vertex connectivity $\kappa(G)$ is at most c.)

Edge Connectivity at most c ($\text{EConn}_{\leq c}$)

Instance: A graph G.
Parameter: An integer c.
Task: Does there exist a set $F \subseteq E(G)$ with $|F| \leq c$ such that $G - F$ is disconnected or has at most one vertex? (Equivalently, the edge connectivity $\lambda(G)$ is at most c.)

Minor Containment (Minor(H))

Instance: A graph G.
Parameter: A fixed graph H.
Task: Is H a minor of G? Equivalently, can H be obtained from a subgraph of G by contracting edges?

Note (scope). The deferred DP-core proof for `Minor(H)` in Appendix D.3 is stated for fixed simple pattern graphs H without isolated vertices.

Nowhere-Zero $\mathbb{Z}_c$-Flow (NZFlow($\mathbb{Z}_c$))

Instance:	A graph $G = (V, E)$.
Parameter:	An integer $c \geq 2$.
Task:	Does there exist an orientation of the edges and a function $\varphi : E \to \mathbb{Z}_c \setminus \{0\}$ such that for every vertex $v \in V$, the net flow at v is zero modulo c (i.e., the sum of outgoing values equals the sum of incoming values in $\mathbb{Z}_c$)?

Appendix C

Clique-Width

This appendix presents a detailed treatment of clique-width, an alternative width measure that applies to broader graph classes than treewidth. The first section introduces the formal definition and key properties, while the second section shows how clique-width can be expressed as a treelike decomposition class in the sense of Chapter 3.

C.1 Definition and Properties

Clique-width is a structural graph parameter that quantifies the complexity of graphs based on labeling and combining operations. It serves as an important alternative to classical parameters such as treewidth, particularly due to its applicability to broader graph classes. Since its introduction, clique-width has attracted considerable attention in both theoretical and practical research. Many algorithms originally designed for graphs of bounded treewidth have been adapted to graphs of bounded clique-width, although these adaptations typically require different decomposition methods [Courcelle and Engelfriet, 2012; Courcelle et al., 2000].

Throughout this appendix, clique-width realization statements are stated for finite nonempty graphs. If one adopts the common convention $\mathrm{cwd}(\emptyset) = 0$, the empty graph is handled separately; it is not represented by the CW treelike class below, whose width-0 language is empty.

Definition 60 (Clique-width [Fomin et al., 2009])**.** *Let G be a graph and $k \in \mathbb{N}^+$. A k-graph is a graph whose vertices are labeled by integers from the set $\{1, 2, \ldots, k\}$. An initial k-graph consists of a single vertex labeled by some $i \in \{1, 2, \ldots, k\}$.*

The clique-width of G, denoted by $\mathrm{cwd}(G)$, is the smallest integer k such that G can be constructed using the following operations:

1. ***Introduce:*** *Create an initial k-graph with one vertex labeled i, denoted $i(v)$.*

2. ***Disjoint union*** ($\oplus$)***:*** *Combine two k-graphs into one graph.*
3. ***Relabel*** ($\rho_{i\to j}$)***:*** *Change all vertices labeled i to label j.*
4. ***Join*** ($\eta_{i,j}$)***:*** *For distinct labels $i \neq j$, connect all vertices labeled i to all vertices labeled j.*

The concept of clique-width provides a structural way to measure the complexity of graphs by describing how a graph can be constructed using labeled operations. To formally represent this constructive approach, we define the notion of an *expression tree*, which explicitly describes the sequence of clique-width operations used to construct a given graph.

Definition 61 (Expression Tree [Fomin et al., 2009])**.** *An expression tree for a graph G is a rooted tree T that describes how G can be built using clique-width operations. The nodes in T represent operations:*

- *Leaves are introduce nodes $i(v)$, representing single-vertex initial k-graphs.*
- *Internal nodes correspond to operations: disjoint union ($\oplus$), relabel ($\rho_{i\to j}$), and join ($\eta_{i,j}$).*
- *The graph represented by the root node (after removing labels) must be isomorphic to G.*

The width of the expression tree is the number of distinct labels used. Thus, if $\mathrm{cwd}(G) \leq k$*, there exists an expression tree of width at most k constructing G.*

It is convenient to view a clique-width expression as a term over a ranked alphabet with symbols for the four operations (introduce, disjoint union, relabel, join). In this way, an expression tree is literally a ground term, and standard automata-theoretic tools apply as in the treewidth setting. This perspective mirrors our use of instructive terms for tree decompositions. Only the signature changes.

Example (building a path on three vertices). Let us construct the path P_3 with vertices v_1, v_2, v_3. Using labels $1, 2, 3$, we can write a clique-width term that first creates v_1 and v_2, joins labels 1 and 2 to add the edge v_1v_2, then creates v_3 and joins labels 2 and 3 to add the edge v_2v_3:

$$\eta_{2,3}\big(\,(\eta_{1,2}(\,1(v_1)\ \oplus\ 2(v_2)\,))\ \oplus\ 3(v_3)\,\big).$$

No edge between labels 1 and 3 is ever introduced, so the resulting graph is exactly a path. One can optionally relabel to reduce the number of active labels once edges are

fixed. This compact term representation is completely analogous to the instructive-term view of tree decompositions and supports bottom-up evaluation by finite devices.

Determining the exact clique-width of an arbitrary graph is computationally challenging. Specifically, given G and k as part of the input, deciding whether $\mathrm{cwd}(G) \leq k$ is NP-complete [Fellows et al., 2009], while polynomial-time algorithms exist for cases where $k \leq 3$ [Corneil et al., 2000]. For fixed $k \geq 4$, polynomial-time recognition is open (in particular for $k = 4$) [Fellows et al., 2009]. Consequently, clique-width meta-theorems should be read as assuming that an appropriate expression is provided as part of the input, or as relying on approximations and related parameters for which decompositions can be obtained more effectively (e.g., via rank-width) [Oum, 2008; Oum and Seymour, 2006].

Clique-width is frequently compared with treewidth and pathwidth because it generalizes structural insights known from these classical measures. An important result linking clique-width and treewidth is the following theorem:

Theorem 24 ([Corneil and Rotics, 2005])**.** *Let G be a graph of treewidth at most k, where $k \geq 1$. Then the clique-width of G satisfies:*

$$\mathrm{cwd}(G) \leq 3 \cdot 2^{k-1}.$$

For $k = 0$, graphs are edgeless and satisfy $\mathrm{cwd}(G) = 1$. Constructive conversions from bounded-treewidth decompositions to bounded-label clique-width expressions are discussed in the same line of work [Corneil and Rotics, 2005].

A fundamental result regarding graphs of bounded clique-width is the MSO_1 model-checking theorem of Courcelle, Makowsky, and Rotics [Courcelle et al., 2000]. Fix an MSO_1 sentence φ and a width bound k. Given, as part of the input, a k-expression (equivalently, an expression tree over $i(v)$, $\oplus$, $\rho_{i \to j}$, and $\eta_{i,j}$) that generates a graph G, one can decide whether $G \models \varphi$ in time $f(k, \|\varphi\|) \cdot |G|$, i.e., linear in $|G|$ with a multiplicative factor depending only on k and φ. This extends the analogous bounded-treewidth meta-theorem, with the same caveat that the decomposition/expression is assumed as part of the input.

However, there remains an important algorithmic gap between clique-width and treewidth. Although many problems parameterized by treewidth are known to be FPT, Fomin et al. [2009] showed that several key problems become hard already under clique-width parameterization: Graph Coloring, Hamiltonian Cycle, and Edge Dominating Set are W[1]-hard when parameterized by clique-width. Thus, algorithmic meta-theorems applicable to graphs of bounded treewidth do not necessarily carry over to graphs of bounded clique-width.

To keep notation uniform with Chapter 3, we recall the basic size parameters of terms

and decompositions. For a ground term τ, we write $|\tau|$ for its size (number of nodes) and $\textsc{Height}(\tau)$ for its height. If $\mathcal{C}$ is a treelike decomposition class with ranked alphabet Σ, the *$\mathcal{C}$-width* of a term is the least k such that $\tau \in \mathsf{L}_k$, and for a graph G it is the least k for which there exists $\tau \in \mathsf{L}_k$ with $\mathcal{G}(\tau) \simeq G$ (Chapter 3). For tree decompositions of graphs we use $tw(G)$ and $pw(G)$. For clique-width expressions, the width of an expression tree is the number of distinct labels that appear. Thus $\mathrm{cwd}(G) \leq k$ if and only if there exists an expression tree of width at most k for G. These conventions let us state algorithmic bounds consistently in terms of a term's size, height, and width.

C.2 Clique-width as a Treelike Decomposition Class

As recalled in Section C.1, clique-width can be described via k-graphs and expression trees (Definitions 60 and 61). An expression tree encodes how a graph is built using the operations $i(v)$ (introduce a vertex with label i), disjoint union $\oplus$, relabel $\rho_{i\to j}$, and join $\eta_{i,j}$. See, for example, Courcelle and Engelfriet [2012]; Fomin et al. [2009]. In this section we make the corresponding treelike decomposition class explicit. This construction is standard (it is essentially the k-expression algebra for clique-width); it is included here only to demonstrate that the interface of Definition 12 is not treewidth-specific.

Remark on notation. The symbol $\oplus$ in clique-width expressions denotes disjoint union, following standard usage. This is unrelated to the boundary-gluing operation $\oplus_B$ on boundaried graphs defined earlier in Chapter 3.

Fix $k \in \mathbb{N}$. For $k = 0$, set

$$\Sigma_0^{\mathsf{cw}} := \{\oplus\}, \qquad L_0^{\mathsf{cw}} := \mathrm{Terms}(\Sigma_0^{\mathsf{cw}}) = \emptyset,$$

and let H_0 and G_0^{cw} be the unique maps with domain L_0^{cw}. Thus width 0 realizes no finite nonempty graph. For the remainder of the construction assume $k \geq 1$, and define a ranked alphabet

$$\Sigma_k^{\mathsf{cw}} = \{\, i \mid i \in [k] \,\} \cup \{\, \oplus \,\} \cup \{\, \rho_{i\to j} \mid i, j \in [k] \,\} \cup \{\, \eta_{i,j} \mid i, j \in [k],\, i \neq j \,\},$$

with arities

$$\mathsf{r}(i) = 0, \qquad \mathsf{r}(\oplus) = 2, \qquad \mathsf{r}(\rho_{i\to j}) = 1, \qquad \mathsf{r}(\eta_{i,j}) = 1.$$

A ground term over Σ_k^{cw} has the shape

$$\tau ::= i \mid \tau_1 \oplus \tau_2 \mid \rho_{i\to j}(\tau) \mid \eta_{i,j}(\tau),$$

which is the standard k-expression grammar with vertex names suppressed. Intuitively,

each occurrence of i creates a fresh vertex with label i. The unary operators $\rho_{i\to j}$ and $\eta_{i,j}$ relabel vertices and insert edges between label classes, while $\oplus$ forms disjoint unions.

We interpret these terms as k-labeled graphs and then forget the labels. Let $\mathcal{G}_k^\ell$ denote the class of finite graphs whose vertices are labeled by integers in $[k]$. Define a map

$$H_k : \mathrm{Terms}(\Sigma_k^{\mathsf{cw}}) \to \mathcal{G}_k^\ell$$

inductively as follows.

- For $i \in [k]$, $H_k(i)$ is the k-graph with a single vertex v labeled i and no edges.
- $H_k(\tau_1 \oplus \tau_2)$ is the disjoint union of $H_k(\tau_1)$ and $H_k(\tau_2)$, with vertex sets made disjoint in the obvious way.
- $H_k(\rho_{i\to j}(\tau))$ is obtained from $H_k(\tau)$ by changing the label of every vertex with label i to j.
- $H_k(\eta_{i,j}(\tau))$ is obtained from $H_k(\tau)$ by adding all edges between vertices labeled i and vertices labeled j (for $i \neq j$).

We then define

$$G_k^{\mathsf{cw}}(\tau) \;=\; \text{the underlying unlabelled graph of } H_k(\tau),$$

obtained by forgetting the vertex labels.

For each k we let L_k^{cw} be the set of all ground terms over Σ_k^{cw}, that is,

$$L_k^{\mathsf{cw}} = \mathrm{Terms}(\Sigma_k^{\mathsf{cw}}).$$

This is a regular tree language, and more restrictive normal forms can be enforced by regular constraints if desired. See Courcelle and Durand [2016]; Courcelle and Olariu [2000] for systematic treatments.

The sequence

$$\mathsf{CW} = \{(\Sigma_k^{\mathsf{cw}}, L_k^{\mathsf{cw}}, G_k^{\mathsf{cw}})\}_{k\in\mathbb{N}}$$

is a treelike decomposition class of arity 2 in the sense of Definition 12: the maximum arity of the symbols is 2, we have $\Sigma_k^{\mathsf{cw}} \subseteq \Sigma_{k+1}^{\mathsf{cw}}$ and $L_k^{\mathsf{cw}} \subseteq L_{k+1}^{\mathsf{cw}}$, and the interpretation is stable under this inclusion since G_{k+1}^{cw} restricts to G_k^{cw} on L_k^{cw}.

Proposition 9 (Clique-width realized by CW)**.** *For every finite nonempty graph G and every $k \in \mathbb{N}$ we have*

$$G \in \mathcal{P}[\mathsf{CW}_k] \quad\Longleftrightarrow\quad \mathrm{cwd}(G) \leq k.$$

Consequently, clique-width on finite nonempty graphs is realized by the treelike decomposition class CW *in the sense of Definition 13.*

Proof. If $k = 0$, then $L_0^{\mathsf{cw}} = \emptyset$, so $\mathcal{P}[\mathsf{CW}_0] = \emptyset$. Since G is finite and nonempty, we also have $\mathrm{cwd}(G) \geq 1$. Hence the claimed equivalence is trivial in this case.

Assume now that $k \geq 1$. By the definition of clique-width and its expression trees (Definitions 60 and 61), a graph G satisfies $\mathrm{cwd}(G) \leq k$ if and only if there exists a k-expression whose expression tree has width at most k and evaluates to a k-labeled graph whose underlying unlabelled graph is isomorphic to G. See, for example, Fomin et al. [2009]. Dropping explicit vertex names yields a term $\tau \in L_k^{\mathsf{cw}}$ with $G_k^{\mathsf{cw}}(\tau) \simeq G$, so $G \in \mathcal{P}[\mathsf{CW}_k]$.

Conversely, given a term $\tau \in L_k^{\mathsf{cw}}$ we can annotate each occurrence of a nullary symbol i with a fresh vertex identifier and interpret the operators $\oplus$, $\rho_{i \to j}$ and $\eta_{i,j}$ as in Definition 60. This produces a k-expression whose expression tree has width at most k and evaluates to a k-graph whose underlying unlabelled graph is $G_k^{\mathsf{cw}}(\tau)$. Hence $G_k^{\mathsf{cw}}(\tau)$ has clique-width at most k, which yields the reverse implication. □

Finally, Courcelle and Olariu [2000] show that for each k there exists a bottom-up finite tree automaton that recognizes the language of k-expressions, and Courcelle and Durand [2016] refine this construction in the setting of fly automata. In particular, CW is automaton-realizable in the sense of Definition 14: there exists a sequence of finite tree automata $(A_k^{\mathsf{cw}})_{k \in \mathbb{N}}$ with $\mathcal{L}(A_k^{\mathsf{cw}}) = L_k^{\mathsf{cw}}$ for each k. We will not need an explicit bound on the number of states of A_k^{cw} here, but the existence of these automata guarantees that the treelike view of clique-width fits into the general automata-based framework of Chapter 3.

Appendix D

Supplementary DP-core Correctness Proofs

This appendix collects detailed instantiations of the DP-core proof framework (Chapter 9) for supplementary cores and deferred proof material from Chapter 4 and Chapter 9.

Scope and how to read this appendix. Unless stated otherwise, correctness statements refer to the realized graph $\mathcal{G}(\tau)$ under the ITD semantics (Appendix A), where parallel edges may arise. When a core is intended to range only over simple graphs (most notably the clique core), we state the simplicity promise explicitly and refer to the standard guard `NOT HasMultipleEdges()`. In particular, the maximum-degree core in Section D.4 counts edge multiplicity, the Hamiltonian-cycle core in Section D.9 uses the multigraph cycle convention stated there, and the feedback-vertex-set core in Section D.10 treats parallel edges as cycles. Therefore agreement with the simple-graph problem statements and SAT encodings of Appendices B and E is asserted only for simple realized graphs, or after conjoining the standard no-multiple-edge guard. Each core proof instantiation follows the same pattern: *witness space* $\rightarrow$ *set-lifted transitions* $\rightarrow$ *denotation and local interpretation predicate* $\rightarrow$ *local law (INT/LSC)* $\rightarrow$ *global correctness and complexity.*

D.1 Witness spaces and bitlength bounds for the cores of Table 5.1

This section explains the asymptotic parameters listed in Table 5.1 by identifying, for each DP-core, the underlying local witness space and the corresponding encoding bound. Its purpose is explanatory rather than original: the witness constructions summarized below are not introduced in this thesis, but are inherited from the witness-space descriptions collected in Appendix H of [de Oliveira Oliveira and Vadiee, 2023],

Core / property	Defined in	Proof here
Maximum degree ($\geq d$)	Section 4.8.2	Section D.4
Clique of size ℓ	Section 4.8.2	Section D.5
Multi-edge detection	Section 4.8.2	Section D.6
Perfect matching	Section 4.8.2	Section D.7
Hamiltonian path	Section 4.8.2	Section D.8
Hamiltonian cycle	Section 4.8.2	Section D.9
Connectedness (`Conn`)	Chapter 5, Table 5.1	Section D.2
Minor containment (`Minor(H)`)	Chapter 5, Table 5.1	Section D.3
Feedback vertex set ($\leq t$)	Section 4.8.2	Section D.10
Maximum independent set	Section 4.8.2	Section D.13
Vertex counting	Section 4.8.2	Section D.14
Witness-space summaries for Table 5.1	Chapter 5, Table 5.1	Section D.1
Framework supplements and lower bounds	Chapter 9	Sections D.11 and D.15

Table D.1: Roadmap for Appendix D.

and are restated here in the language of ITD-coherent DP-cores used in this monograph. Thus, the role of this appendix is to translate those earlier constructions into the complexity vocabulary of Chapter 4 (Definitions 28, 29, 30, and 31). Where a row of Table 5.1 additionally uses an external compression argument, we cite that source explicitly; in particular, the improved multiplicity bound for the Hamiltonian core relies on the rank-based technique of [Bodlaender et al., 2015].

For a finite core, once the witness encoding uses $\beta(k)$ bits, there are at most $2^{O(\beta(k))}$ possible local witnesses, so witness-level state complexity satisfies $\nu(k) = 2^{O(\beta(k))}$. If the core is deterministic, then $\mu(k) = 1$, hence each useful witness set is a singleton and $\delta_D(k) = 2^{O(\beta(k))}$. In the generic non-deterministic case one may use the coarse bound $\mu(k) \leq 2^{O(\beta(k))}$, which yields $\delta_D(k) = 2^{2^{O(\beta(k))}}$. The rows of Table 5.1 are obtained from the witness encodings below together with these generic counting consequences, except where a sharper multiplicity estimate is stated explicitly. For direct follow-up, Section D.2 proves the `Conn` row, Section D.3 proves the `Minor(H)` row, and the remaining catalog cores are handled in the appendix sections listed in Table D.1.

Simple. *Witness space.* A local witness is a subset

$$w \subseteq \binom{[k+1]}{2}.$$

Intuitively, w records which unordered pairs of active labels are joined by a unique edge in the current partial graph. All local witnesses are final, and transitions ensure that a witness exists exactly when the realized partial graph is simple and w records the active boundary pairs with a unique edge [de Oliveira Oliveira and Vadiee, 2023, App. H.1].

Bitlength and complexity. Encoding one Boolean coordinate for each unordered label pair gives $\binom{k+1}{2}$ bits, hence $\beta(k) = O(k^2)$. The construction is deterministic, so $\mu(k) = 1$ and therefore $\nu(k), \delta_D(k) = 2^{O(\beta(k))}$.

MaxDeg$(\geq c)$**.** *Witness space.* A local witness is a pair (x, y), where $x \in \{0, 1\}$ and

$$y \in \{0, 1, \ldots, c+1\}^{k+1}.$$

For each active label u, y_u stores the boundary degree truncated at $c+1$; the flag x records whether a forgotten vertex of degree at least c has already occurred. Finality holds when $x = 1$ or some active boundary vertex has degree at least c [de Oliveira Oliveira and Vadiee, 2023, App. H.2].

Bitlength and complexity. The truncated degree vector requires $O(k \log c)$ bits overall, so $\beta(k) = O(k \log c)$. The core is deterministic, hence $\mu(k) = 1$ and $\nu(k), \delta_D(k) = 2^{O(\beta(k))}$.

MinDeg$(\leq c)$**.** *Witness space.* The witness space is again pairs (x, y) with $x \in \{0, 1\}$ and $y \in \{0, 1, \ldots, c+1\}^{k+1}$. The vector stores truncated active-boundary degrees, while x records whether a forgotten vertex of degree at most c has already been seen. Finality holds when $x = 1$ or some active boundary vertex has degree at most c [de Oliveira Oliveira and Vadiee, 2023, App. H.3].

Bitlength and complexity. The same encoding argument gives $\beta(k) = O(k \log c)$, and the core can again be chosen deterministic, so $\mu(k) = 1$ and $\nu(k), \delta_D(k) = 2^{O(\beta(k))}$.

c-Colorability. *Witness space.* A local witness may be viewed as a vector

$$w \in \{0, 1, \ldots, c\}^{k+1},$$

or equivalently a partial c-coloring of the active bag, where $w_u = 0$ indicates label u is inactive. Thus, the witness stores exactly the boundary coloring induced by a proper c-coloring of the current partial graph. All local witnesses are final [de Oliveira Oliveira and Vadiee, 2023, App. H.4].

Bitlength and complexity. Storing one value in $\{0, \ldots, c\}$ per label yields $\beta(k) = O(k \log c)$. A crude but sufficient multiplicity bound is the number of bag colorings, $(c+1)^{k+1} = 2^{O(k \log c)} = 2^{O(\beta(k))}$. Hence $\nu(k) = 2^{O(\beta(k))}$ and $\delta_D(k) = 2^{2^{O(\beta(k))}}$ under the coarse table-level multiplicity summary.

Conn. *Witness space.* A local witness is a pair (γ, P), where P is a partition of active labels by connected component and $\gamma \in \{0, 1, 2, 3\}$ distinguishes four global situations: the partial graph is empty; nonempty with every vertex reachable from the boundary; bag empty but partial graph connected; or partial graph disconnected. Finality holds when $\gamma \neq 3$ and P has at most one cell [de Oliveira Oliveira and Vadiee, 2023, App. H.5].

Bitlength and complexity. A partition of a bag of size at most $k+1$ can be encoded by canonical block identifiers, using $O(k \log k)$ bits; the constant-size flag γ does not change asymptotics. Hence $\beta(k) = O(k \log k)$. In fact this core can be made deterministic, so $\mu(k) = 1$ (stronger than the coarse $2^{O(\beta(k))}$ summary sometimes used at table level). Consequently $\nu(k) = 2^{O(\beta(k))}$. The table keeps the coarse upper bounds $\mu(k) = 2^{O(\beta(k))}$ and $\delta_D(k) = 2^{2^{O(\beta(k))}}$, but the deterministic implementation sharpens these to $\mu(k) = 1$ and $\delta_D(k) = 2^{O(\beta(k))}$.

VConn$(\leq c)$**.** *Witness space.* A local witness is a triple (r, γ, P), where $r \in \{0, \ldots, c\}$ is the size of a deleted vertex set X, P is the partition of surviving active labels in $\mathcal{G}(\tau) \setminus X$, and $\gamma \in \{0, 1, 2, 3\}$ records the same connectivity status as in `Conn`. Final witnesses are those showing that the graph becomes disconnected after deleting at most c vertices [de Oliveira Oliveira and Vadiee, 2023, App. H.6].

Bitlength and complexity. The counter r contributes $O(\log c)$ bits, while the partition part contributes $O(k \log k)$ bits, so $\beta(k) = O(\log c + k \log k)$. For Table 5.1, the generic multiplicity bound $2^{O(\beta(k))}$ is sufficient. Therefore $\nu(k) = 2^{O(\beta(k))}$ and $\delta_D(k) = 2^{2^{O(\beta(k))}}$.

EConn$(\leq c)$**.** *Witness space.* A local witness again has form (r, γ, P), but now r counts deleted edges rather than deleted vertices. The partition P records active-bag components in $\mathcal{G}(\tau) \setminus Y$, where $|Y| = r$, and γ is the same four-valued connectivity flag as above. Finality means that deleting at most c edges disconnects the graph [de Oliveira Oliveira and Vadiee, 2023, App. H.7].

Bitlength and complexity. Exactly the same encoding argument gives $\beta(k) = O(\log c + k \log k)$. As in the table, one may use the generic multiplicity bound $2^{O(\beta(k))}$. Therefore $\nu(k) = 2^{O(\beta(k))}$ and $\delta_D(k) = 2^{2^{O(\beta(k))}}$.

Hamiltonian. *Witness space.* A local witness is a pair (β, M). Here $\beta : S \to \{0, 1, 2\}$, for some $S \subseteq [k+1]$, records whether each active boundary vertex has degree 0, 1, or 2 inside a partial collection of vertex-disjoint paths, and M is a matching on $\beta^{-1}(1)$ pairing boundary endpoints belonging to the same partial path. Final witnesses are those in which the partial path system closes to a single spanning cycle (Hamiltonian cycle) [de Oliveira Oliveira and Vadiee, 2023, App. H.8].

Bitlength and complexity. The degree profile uses $O(k)$ bits, while the matching can be encoded in $O(k \log k)$ bits overall; hence $\beta(k) = O(k \log k)$. Naive multiplicity is $2^{O(k \log k)}$, but the row in Table 5.1 uses the sharper rank-based cleaning bound of

[Bodlaender et al., 2015], reducing $\mu(k)$ to $2^{O(k)}$. The witness encoding still yields at most $2^{O(\beta(k))}$ possible local witnesses, so $\nu(k) = 2^{O(\beta(k))}$. Combining $\nu(k) = 2^{O(\beta(k))}$ with $\mu(k) = 2^{O(k)}$ gives

$$\delta_D(k) \leq \nu(k)^{\mu(k)} = 2^{O(k \log k) \cdot 2^{O(k)}} = 2^{2^{O(k)}},$$

matching the Hamiltonian row of Table 5.1.

NZFlow($\mathbb{Z}_c$). *Witness space.* The $O(k \log c)$ row in Table 5.1 is justified here by compressing the boundary-to-boundary flow summary of [de Oliveira Oliveira and Vadiee, 2023, App. H.9] to the induced boundary-divergence vector; this is a reformulation in the language of the present framework.

Fix $c \geq 2$. For an orientation with edge-value map $\varphi : E \to \mathbb{Z}_c \setminus \{0\}$, define divergence at vertex x by

$$\partial_\varphi(x) := \sum_{e:\, t(e)=x} \varphi(e) - \sum_{e:\, h(e)=x} \varphi(e) \in \mathbb{Z}_c.$$

Thus x satisfies the flow equation iff $\partial_\varphi(x) = 0$.

For a bag $B \subseteq [k+1]$, let

$$W_k^{\text{flow}}(B) := \{\, q : B \to \mathbb{Z}_c \,\}, \qquad W_k^{\text{flow}} := \bigcup_{B \subseteq [k+1]} W_k^{\text{flow}}(B).$$

A witness $q \in W_k^{\text{flow}}(B)$ records residual imbalance on active labels: forgotten vertices are already balanced, and for each active label u, $q(u)$ equals the current divergence at the corresponding boundary vertex.

Cleaning can be identity. Finality accepts exactly the zero vector on the current bag. Constructor transitions are then the natural divergence updates: leaf gives the empty map; introduce-vertex adds a new zero coordinate; forget-vertex is allowed only when the forgotten coordinate is zero; introduce-edge updates endpoint coordinates by $\pm a$ for chosen nonzero $a \in \mathbb{Z}_c$ and orientation; join sums child vectors coordinatewise on the common bag.

This compressed witness is sufficient because future constructors interact with the realized partial graph only through the current bag; for feasibility, the needed interface information is exactly boundary divergence.

Bitlength and complexity. Each witness stores one value of $\mathbb{Z}_c$ for each of at most $k+1$ active labels, so

$$\beta(k) = O(k \log c).$$

For fixed bag B, there are at most

$$c^{|B|} \leq c^{k+1} = 2^{O(k \log c)} = 2^{O(\beta(k))}$$

possible witnesses. Hence

$$\nu(k) = 2^{O(\beta(k))}, \qquad \mu(k) \le 2^{O(\beta(k))}, \qquad \delta_D(k) \le 2^{2^{O(\beta(k))}},$$

matching the NZFlow row in Table 5.1.

Minor(H)**.** *Witness space.* A local witness is a pair of tuples

$$w = \Big(\big[(\gamma_x, P_x)\big]_{x \in V(H)},\ \big[b_e\big]_{e \in E(H)} \Big).$$

For each pattern vertex x, (γ_x, P_x) is a `Conn`-type summary of the current branch set X_x of a quasimodel, while for each pattern edge e, bit b_e records whether the corresponding model edge has already been realized. Finality requires every model edge to be realized and every branch set to be connected, so that the witness encodes a model of H as a minor [de Oliveira Oliveira and Vadiee, 2023, App. H.10].

Bitlength and complexity. A straightforward encoding gives $O(|V(H)| \cdot k \log k + |E(H)|)$ bits, but Appendix H shows how to compress this to $O(k \log k + |V(H)| + |E(H)|)$, which is exactly the bound used in Table 5.1. The table-level multiplicity summary can be written as $\mu(k) \le 2^{O(\beta(k))}$. Hence $\nu(k) = 2^{O(\beta(k))}$ and $\delta_D(k) = 2^{2^{O(\beta(k))}}$ by the generic counting argument above.

In each case, the row of Table 5.1 is obtained by combining the witness encoding above with the bitlength/multiplicity framework of Chapter 4; the witness-level state complexity $\nu(k)$ and deterministic state complexity $\delta_D(k)$ then follow from counting useful witnesses and useful witness sets. The table entries are therefore not claimed as new in this thesis; rather, they are inherited from earlier DP constructions and recast in the terminology of the present framework.

D.2 Connectedness (`Conn`)

This section instantiates the local-to-global proof framework of Section 9.1 for the connectedness core `Conn`. The target graph property is connectivity of the realized graph $\mathcal{G}(\tau)$.

D.2.1 Witness space, cleaning, and acceptance

For bag $B \subseteq [k+1]$, define

$$W_k^{\mathrm{Conn}}(B) := \{(\gamma, P) \mid \gamma \in \{0,1,2,3\}, \\ P \text{ is a partition of } B\},$$
$$W_k^{\mathrm{Conn}} := \bigcup_{B \subseteq [k+1]} W_k^{\mathrm{Conn}}(B).$$

The flag γ has the standard four-way meaning used in [de Oliveira Oliveira and Vadiee, 2023, App. H.5]: empty graph; nonempty and boundary-reachable; bag-empty connected nonempty; or disconnected/hidden component. Cleaning is identity.

Finality at bag B is

$$\mathrm{Final}_k^{\mathrm{Conn}}(B, (\gamma, P)) = 1 \iff \gamma \neq 3 \text{ and } P \text{ has at most one block.}$$

Under this convention, the empty graph is accepted.

D.2.2 Transitions

Transitions are functional on well-formed witnesses. Let $P \setminus u$ denote deleting u from its block, $P^{u \sim v}$ denote block merge, and for same-domain partitions P_1, P_2, let $P_1 \sqcup P_2$ be the transitive-closure join of component relations.

- **Leaf:**
$$b_{\texttt{leaf}}() = \{(0, \emptyset)\}.$$
- **Introduce vertex u:**
$$b_{\texttt{intro\_}u}(\gamma, P) = \{(\gamma', P \cup \{\{u\}\})\},$$
with $\gamma' = 1$ for $\gamma \in \{0,1\}$ and $\gamma' = 3$ for $\gamma \in \{2,3\}$.
- **Forget vertex u:** write $C = \mathrm{blk}_P(u)$, where $\mathrm{blk}_P(u)$ denotes the block of P containing u, and let $P^- = P \setminus u$. Then
$$b_{\texttt{forget\_}u}(\gamma, P) = \{(\gamma^-, P^-)\},$$
where $\gamma^- = 3$ if $\gamma = 3$, $\gamma^- = 2$ if $\gamma = 1$ and $P^- = \emptyset$, $\gamma^- = 1$ if $\gamma = 1$ and $|C| \geq 2$, and $\gamma^- = 3$ otherwise.

- **Introduce edge uv:**

$$b_{\texttt{intro\_u\_v}}(\gamma, P) = \{(\gamma,\ P^{u\sim v})\}.$$

- **Join:** for child witnesses $(\gamma_1, P_1), (\gamma_2, P_2)$ on common bag B,

$$b_{\texttt{join}}\big((\gamma_1, P_1), (\gamma_2, P_2)\big) = \{(\gamma, P_1 \sqcup P_2)\},$$

  where $\gamma = 3$ if either child has $\gamma_i = 3$, otherwise $\gamma = 1$ for $B \neq \emptyset$; for $B = \emptyset$: $(0,0) \mapsto 0$, $(0,2)$ or $(2,0) \mapsto 2$, $(2,2) \mapsto 3$.

D.2.3 Denotation and local interpretation predicate

For a term $\tau \in ITD_k$, define the bag projection

$$\pi^{\mathrm{Conn}}_{B(\tau)}(\mathcal{G}(\tau)) = (\gamma_\tau, P_\tau) \in W^{\mathrm{Conn}}_k(B(\tau))$$

by:

1. P_τ is the partition of $B(\tau)$ in which $u, v \in B(\tau)$ lie in the same block iff the boundary vertices represented by u, v lie in the same connected component of $\mathcal{G}(\tau)$;

2. γ_τ is the four-way flag from Section D.2; explicitly

$$\gamma_\tau = \begin{cases} 0, & \text{if } \mathcal{G}(\tau) \text{ is empty,} \\ 1, & \text{if } \mathcal{G}(\tau) \neq \emptyset,\ B(\tau) \neq \emptyset, \\ & \text{and every connected component of } \mathcal{G}(\tau) \text{ meets } \theta[\tau](B(\tau)), \\ 2, & \text{if } B(\tau) = \emptyset,\ \mathcal{G}(\tau) \text{ is connected, and } \mathcal{G}(\tau) \neq \emptyset, \\ 3, & \text{otherwise.} \end{cases}$$

We then define

$$\mathsf{Interp}_{\mathrm{Conn}}(\tau, (\gamma, P)) \iff \pi^{\mathrm{Conn}}_{B(\tau)}(\mathcal{G}(\tau)) = (\gamma, P),$$

and

$$\mathrm{Sem}_{\mathrm{Conn}}(\tau) := \{(\gamma, P) \in W^{\mathrm{Conn}}_k(B(\tau)) : \mathsf{Interp}_{\mathrm{Conn}}(\tau, (\gamma, P))\}.$$

Thus $\mathrm{Sem}_{\mathrm{Conn}}(\tau)$ is the singleton containing the unique connectedness summary of $\mathcal{G}(\tau)$ relative to its current bag. This is the Section 9.1 restatement of the (P)-Conn semantics from [de Oliveira Oliveira and Vadiee, 2023, App. H.5].

D.2.4 Local law and global correctness

Lemma 32 (Conn interpretation law)**.** *For every constructor a and child terms $\vec{\sigma}$,*

$$\mathsf{Interp}_{\mathrm{Conn}}(a(\vec{\sigma}), w) \iff \exists \vec{w} \left(\bigwedge_i \mathsf{Interp}_{\mathrm{Conn}}(\sigma_i, w_i) \ \wedge \ w \in b_a(\vec{w}) \right).$$

Proof. **Leaf.** The realized graph of `leaf` is empty, so

$$\pi_{\emptyset}^{\mathrm{Conn}}(\mathcal{G}(\texttt{leaf})) = (0, \emptyset),$$

which is exactly the unique element of $b_{\texttt{leaf}}()$.

Introduce vertex. Let $\tau = \texttt{intro\_}u(\sigma)$. The new vertex represented by u is isolated and belongs to the bag. Therefore its block is the singleton $\{u\}$, while all previous boundary-component relations are unchanged. On the flag side, if the child was empty or already bag-visible ($\gamma \in \{0, 1\}$), the parent is now nonempty and every connected component still meets the bag, so the flag becomes 1. If the child had a hidden component or was bag-empty connected ($\gamma \in \{2, 3\}$), that hidden/non-bag-visible behavior persists after adding an isolated bag vertex, so the parent flag is 3. This is exactly the stated introduce-vertex transition.

Forget vertex. Let $\tau = \texttt{forget\_}u(\sigma)$ and write the child witness as (γ, P). Forgetting u only removes one boundary label, so the parent partition is exactly $P \setminus u$. The flag changes according to whether the connected component meeting u still has another bag vertex after forgetting. If the child was hidden ($\gamma = 3$), it stays hidden. If the child was bag-visible and u was the last bag vertex of its component, then that component becomes hidden; in the special case that the parent bag is empty and the whole graph is connected and nonempty, the flag becomes 2. If the child was bag-visible and the component of u still contains another bag label, the flag remains 1. This is exactly the forget rule above.

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$. The only change in the realized graph is that the two bag vertices represented by u and v may now lie in the same connected component. Thus the parent partition is the block merge $P^{u \sim v}$, while the hidden/non-hidden status of components does not change. Hence the flag is preserved and the partition update is exactly $b_{\texttt{intro\_}u\texttt{\_}v}$.

Join. Let $\tau = \texttt{join}(\tau_1, \tau_2)$ with common bag B, and let the child witnesses be (γ_i, P_i). Because $\mathcal{G}(\tau)$ is the pushout union of the two child graphs along the common bag, two bag labels $u, v \in B$ lie in the same connected component of $\mathcal{G}(\tau)$ iff they can be connected by alternately following child-internal connectedness relations. This is exactly the transitive-closure join $P_1 \sqcup P_2$. For the flag, if either child already has a hidden

component then that component remains hidden in the parent, so $\gamma = 3$. Otherwise, for nonempty bag B, every parent component meets the bag, so $\gamma = 1$. For empty bag, the join is an ordinary disjoint union of two bag-empty child graphs, and the four possible root cases reduce exactly to the table $(0,0) \mapsto 0$, $(0,2)$ or $(2,0) \mapsto 2$, and $(2,2) \mapsto 3$. Hence the join rule matches the parent projection. □

Because cleaning is identity, Lemma 32 yields the local semantic commutation condition, so the generic framework implies dynamic/denotational equality.

Theorem 25 (Connectedness correctness)**.** *For all k and all $\tau \in ITD_k$,*

$$\Gamma[\mathrm{Conn}, k](\tau) = \mathrm{Sem}_{\mathrm{Conn}}(\tau) = \{(\gamma, P) : \mathsf{Interp}_{\mathrm{Conn}}(\tau, (\gamma, P))\},$$

and

$$\tau \in \mathsf{Acc}(\mathrm{Conn}[k]) \iff \mathcal{G}(\tau) \textit{ is connected}$$

under the convention that the empty graph is accepted.

Proof. Lemma 32 gives (INT) for `Conn`. Since cleaning is the identity, Lemma 27 yields (LSC). Therefore Theorems 17 and 19 give

$$\Gamma[\mathrm{Conn}, k](\tau) = \mathrm{Sem}_{\mathrm{Conn}}(\tau) = \{(\gamma, P) : \mathsf{Interp}_{\mathrm{Conn}}(\tau, (\gamma, P))\}.$$

For the acceptance claim, let (γ, P) be the unique witness in $\mathrm{Sem}_{\mathrm{Conn}}(\tau)$.

($\Rightarrow$) Suppose τ is accepted. Then $\gamma \neq 3$ and P has at most one block. If $\gamma = 0$, then $\mathcal{G}(\tau)$ is empty and is accepted by convention. If $\gamma = 2$, then $B(\tau) = \emptyset$ and $\mathcal{G}(\tau)$ is connected and nonempty. If $\gamma = 1$, then every connected component of $\mathcal{G}(\tau)$ meets the bag and all bag vertices lie in at most one block of P. Hence all bag vertices lie in the same connected component, and since every connected component meets the bag, there can be only one connected component in total. Thus $\mathcal{G}(\tau)$ is connected.

($\Leftarrow$) Conversely, suppose $\mathcal{G}(\tau)$ is connected, with the empty graph counted as accepted. If $\mathcal{G}(\tau)$ is empty, then $(\gamma, P) = (0, \emptyset)$, which is final. If $\mathcal{G}(\tau)$ is nonempty and $B(\tau) = \emptyset$, then $(\gamma, P) = (2, \emptyset)$, which is final. If $\mathcal{G}(\tau)$ is nonempty and $B(\tau) \neq \emptyset$, then every connected component meets the bag and all bag labels lie in a single block, so (γ, P) has $\gamma = 1$ and P has at most one block. Hence the root witness is final and τ is accepted. □

D.2.5 Complexity summary

From Appendix D.1, a `Conn` witness uses $\beta_{\mathrm{Conn}}(k) = O(k \log k)$ bits. As recorded there, `Conn` admits deterministic realization with $\mu_{\mathrm{Conn}}(k) = 1$.

D.3 Minor Containment (`Minor(H)`)

This section instantiates the framework for the core `Minor(H)` for a fixed simple pattern graph $H = (V_H, E_H)$ without isolated vertices. We use the quasimodel/model semantics of [de Oliveira Oliveira and Vadiee, 2023, App. H.10], restated in the notation and proof framework of Section 9.1.

D.3.1 Witness space, cleaning, and acceptance

A local witness at bag B has shape

$$w = \Big(\big[(\gamma_x, P_x)\big]_{x \in V_H},\ [b_e]_{e \in E_H} \Big),$$

where each (γ_x, P_x) is a `Conn`-style witness over a domain $B_x \subseteq B$, domains B_x are pairwise disjoint, and $b_e \in \{0, 1\}$ marks whether the model edge for e is already realized. Write $W_k^{\mathrm{Minor}(H)}(B)$ for this set and $W_k^{\mathrm{Minor}(H)} := \bigcup_B W_k^{\mathrm{Minor}(H)}(B)$. For such a witness w, write

$$B_x(w) := \mathrm{dom}(P_x) \subseteq B$$

and define the partial owner map

$$\mathrm{owner}_w(u) := \begin{cases} x, & \text{if } u \in B_x(w) \text{ for some } x \in V_H, \\ \bot, & \text{otherwise.} \end{cases}$$

Since the domains $B_x(w)$ are pairwise disjoint, owner_w is well-defined.

Cleaning is identity. Finality requires

1. all pattern-edge flags are set ($b_e = 1$ for all $e \in E_H$), and
2. every connectivity component (γ_x, P_x) is final for `Conn`, i.e.

$$\mathrm{Final}_k^{\mathrm{Conn}}(B_x(w), (\gamma_x, P_x)) = 1 \qquad \text{for all } x \in V_H.$$

Since H has no isolated vertices, this captures models (not only quasimodels) at the root.

D.3.2 Global objects, bag projection, and interpretation

Recall that a quasimodel of H in a graph G consists of pairwise disjoint branch sets $[X_x]_{x \in V_H}$ together with edge witnesses $[y_e]_{e \in E_H}$, where each y_e is either 0 or an edge of G joining the two branch sets corresponding to the endpoints of e. For a graph G, let

$$\mathcal{X}_{\mathrm{Minor}(H)}(G)$$

denote the set of all such quasimodels in G.

For a term σ with bag $B(\sigma)$ and a quasimodel

$$M = \big([X_x]_{x\in V_H},\ [y_e]_{e\in E_H}\big) \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\sigma)),$$

define its bag projection

$$\pi_{B(\sigma)}(M) = \Big(\big[(\gamma_x, P_x)\big]_{x\in V_H},\ [b_e]_{e\in E_H}\Big) \in W_k^{\mathrm{Minor}(H)}(B(\sigma))$$

as follows. For each $x \in V_H$, let

$$B_x^\sigma(M) := \{u \in B(\sigma) : \theta[\sigma](u) \in X_x\}.$$

Because the branch sets are pairwise disjoint, the sets $B_x^\sigma(M)$ are pairwise disjoint. Define P_x to be the partition of $B_x^\sigma(M)$ in which u, v lie in the same block iff $\theta[\sigma](u)$ and $\theta[\sigma](v)$ lie in the same connected component of $\mathcal{G}(\sigma)[X_x]$. Define γ_x by the same four-way rule as in the connectedness section, applied to $\mathcal{G}(\sigma)[X_x]$ with boundary $B_x^\sigma(M)$:

$$\gamma_x = \begin{cases} 0, & \text{if } X_x = \emptyset, \\ 1, & \text{if } X_x \neq \emptyset,\ B_x^\sigma(M) \neq \emptyset, \\ & \text{and every connected component of } \mathcal{G}(\sigma)[X_x] \text{ meets } \theta[\sigma](B_x^\sigma(M)), \\ 2, & \text{if } B_x^\sigma(M) = \emptyset,\ \mathcal{G}(\sigma)[X_x] \text{ is connected, and } X_x \neq \emptyset, \\ 3, & \text{otherwise.} \end{cases}$$

Finally, for each $e \in E_H$, define

$$b_e := 1 \iff y_e \neq 0.$$

Thus $\pi_{B(\sigma)}(M)$ records, for each branch set, its connectedness summary relative to the current bag, together with the Boolean information of which pattern edges are already witnessed.

We then define

$$\mathsf{Interp}_{\mathrm{Minor}(H)}(\sigma, w) \iff \exists M \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\sigma)) :\ \pi_{B(\sigma)}(M) = w,$$

and

$$\mathrm{Sem}_{\mathrm{Minor}(H)}(\sigma) := \{w \in W_k^{\mathrm{Minor}(H)}(B(\sigma)) : \mathsf{Interp}_{\mathrm{Minor}(H)}(\sigma, w)\}.$$

D.3.3 Transitions

- **Leaf:** the unique all-empty witness

$$\Big(\big[(0,\emptyset)\big]_{x\in V_H},\ [0]_{e\in E_H} \Big).$$

- **Introduce vertex u:** nondeterministically assign u to at most one branch set (or none). If assigned to x, update only (γ_x, P_x) via $b^{\mathrm{Conn}}_{\texttt{intro\_}u}$.

- **Forget vertex u:** if u is owned by x, update only that component via $b^{\mathrm{Conn}}_{\texttt{forget\_}u}$; otherwise unchanged.

- **Introduce edge uv:** if $\mathrm{owner}_w(u) = \mathrm{owner}_w(v) = x \in V_H$, update only component x by $b^{\mathrm{Conn}}_{\texttt{intro\_}u\texttt{\_}v}$; if $\mathrm{owner}_w(u) = x \in V_H$, $\mathrm{owner}_w(v) = y \in V_H$, $x \neq y$, and $e = \{x, y\} \in E_H$, branch between ignoring the edge and setting the corresponding $b_e := 1$ (well-defined because H is simple); otherwise unchanged.

- **Join:** for witnesses $w_i = \big([(\gamma_x^{(i)}, P_x^{(i)})]_{x\in V_H}, [b_e^{(i)}]_{e\in E_H}\big)$ on the common bag B, define

$$\mathrm{Comp}_B(w_1, w_2) \iff \forall u \in B:\ \mathrm{owner}_{w_1}(u) = \mathrm{owner}_{w_2}(u).$$

  When $\mathrm{Comp}_B(w_1, w_2)$ holds, the owner domains agree componentwise, so for each $x \in V_H$ the `Conn` join transition is applied to $((\gamma_x^{(1)}, P_x^{(1)}), (\gamma_x^{(2)}, P_x^{(2)}))$, yielding a unique output (γ_x, P_x). We then set

$$w_1 \otimes_B w_2 := \left\{ \left(\big[(\gamma_x, P_x)\big]_{x\in V_H},\ [\max(b_e^{(1)}, b_e^{(2)})]_{e\in E_H} \right) \right\}.$$

  If $\mathrm{Comp}_B(w_1, w_2)$ fails, we set $w_1 \otimes_B w_2 := \emptyset$.

D.3.4 Join witness composition

Because a minor quasimodel stores only one witness edge y_e per pattern edge $e \in E_H$, the exact object-level gluing clause Definition 57(G2a) is stronger than what the existential interpretation predicate $\mathsf{Interp}_{\mathrm{Minor}(H)}$ needs: two compatible child quasimodels may choose different nonzero witness edges for the same pattern edge while still inducing the same joined witness. For the join case of (INT), it therefore suffices to record the following restriction and reconstruction facts.

Lemma 33 (Minor join restriction and reconstruction)**.** *Let $\tau = \texttt{join}(\tau_1, \tau_2)$ be a join node with common bag B. With $\mathcal{X}_{\mathrm{Minor}(H)}$, π_B, Comp_B, and $\otimes_B$ defined above, the following hold:*

(i) For every $M \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau))$, the canonical restrictions $M_i := M|_{\tau_i}$ belong to $\mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau_i))$ and satisfy

$$\mathrm{Comp}_B(\pi_B(M_1), \pi_B(M_2)) \qquad \textit{and} \qquad \pi_B(M) \in \pi_B(M_1) \otimes_B \pi_B(M_2).$$

(ii) Conversely, suppose that

$$M_i \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau_i)) \quad \textit{for } i \in \{1,2\}, \qquad \mathrm{Comp}_B(\pi_B(M_1), \pi_B(M_2)).$$

Then, for every

$$w \in \pi_B(M_1) \otimes_B \pi_B(M_2)$$

there exists some $M \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau))$ with $\pi_B(M) = w$.

Proof. Write $\iota_i : \mathcal{G}(\tau_i) \hookrightarrow \mathcal{G}(\tau)$ for the pushout embeddings.

Restriction. Let

$$M = \big([X_x]_{x \in V_H}, [y_e]_{e \in E_H}\big) \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau)).$$

For each $i \in \{1,2\}$ and each $x \in V_H$, put

$$X_x^{(i)} := \iota_i^{-1}(X_x) \subseteq V(\mathcal{G}(\tau_i)).$$

For each $e \in E_H$, define

$$y_e^{(i)} := \begin{cases} \iota_i^{-1}(y_e), & \text{if } y_e \text{ is an edge in the image of } \mathcal{G}(\tau_i), \\ 0, & \text{otherwise.} \end{cases}$$

Because the child graphs intersect only on bag vertices and their edge sets are disjoint in the pushout (as multisets), these restrictions are well-defined. Since the branch sets X_x are pairwise disjoint in $\mathcal{G}(\tau)$, the restricted branch sets $X_x^{(i)}$ remain pairwise disjoint in each child. Thus

$$M_i := \big([X_x^{(i)}]_{x \in V_H}, [y_e^{(i)}]_{e \in E_H}\big) \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau_i))$$

for $i \in \{1,2\}$.

Reconstruction from compatible children. Now let

$$M_i = \big([X_x^{(i)}]_{x \in V_H}, [y_e^{(i)}]_{e \in E_H}\big) \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau_i))$$

and write

$$w_i = \pi_B(M_i) = \Big(\big[(\gamma_x^{(i)}, P_x^{(i)})\big]_{x \in V_H}, [b_e^{(i)}]_{e \in E_H}\Big).$$

Assume $\mathrm{Comp}_B(w_1, w_2)$. For each $x \in V_H$, define

$$X_x := \iota_1(X_x^{(1)}) \cup \iota_2(X_x^{(2)}).$$

We claim that the sets X_x are pairwise disjoint. Indeed, if a vertex lies outside the bag image $\theta[\tau](B)$, then it belongs to a unique child and disjointness follows from child disjointness. If it lies in the bag image, then it is represented by some label $u \in B$; compatibility says $\mathrm{owner}_{w_1}(u) = \mathrm{owner}_{w_2}(u)$, so the two children place that common bag vertex in the same branch set (or in none). Hence no bag vertex can be assigned to two different branch sets after gluing.

For each $e \in E_H$, choose

$$y_e \in \begin{cases} \{\iota_1(y_e^{(1)})\}, & \text{if } y_e^{(1)} \neq 0, \\ \{\iota_2(y_e^{(2)})\}, & \text{if } y_e^{(1)} = 0 \text{ and } y_e^{(2)} \neq 0, \\ \{0\}, & \text{if } y_e^{(1)} = y_e^{(2)} = 0. \end{cases}$$

Then

$$M := \big([X_x]_{x \in V_H}, [y_e]_{e \in E_H}\big)$$

is a quasimodel of H in $\mathcal{G}(\tau)$: each chosen nonzero y_e is still an edge joining the appropriate glued branch sets, and cross-child chosen witness edges remain distinct because the child edge sets are disjoint in the pushout.

It remains to identify the projection of M. Fix $x \in V_H$. By construction, $\mathcal{G}(\tau)[X_x]$ is the pushout union of $\mathcal{G}(\tau_1)[X_x^{(1)}]$ and $\mathcal{G}(\tau_2)[X_x^{(2)}]$ along the common boundary labels owned by x. Therefore the connectivity summary of the glued branch set is exactly the `Conn` join of the child summaries: if

$$(\gamma_x, P_x) \in b_{\texttt{join}}\big((\gamma_x^{(1)}, P_x^{(1)}), (\gamma_x^{(2)}, P_x^{(2)})\big),$$

then (γ_x, P_x) is the unique `Conn` summary of $\mathcal{G}(\tau)[X_x]$ relative to the glued bag domain. For the edge flags, we have $b_e = 1$ in the parent iff at least one child already had a nonzero witness for e, i.e. $b_e = \max(b_e^{(1)}, b_e^{(2)})$. Hence

$$\pi_B(M) \in w_1 \otimes_B w_2.$$

Complete restriction. Conversely, let

$$M = \big([X_x]_{x \in V_H}, [y_e]_{e \in E_H}\big) \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau))$$

and let M_i be its restrictions. For any bag label $u \in B$, the common bag vertex $\theta[\tau](u)$

belongs to at most one branch set X_x. Hence the restrictions assign the same owner to u in both children, so $\mathrm{Comp}_B(\pi_B(M_1), \pi_B(M_2))$ holds. The same pushout analysis as above shows that for each x the parent `Conn` summary is the join of the two child summaries, and that the parent edge flag for e is the maximum of the child flags. Therefore

$$\pi_B(M) \in \pi_B(M_1) \otimes_B \pi_B(M_2).$$

This proves (i).

The preceding reconstruction argument proves (ii): starting from compatible child quasimodels M_1, M_2, it constructs a parent quasimodel M with any prescribed witness $w \in \pi_B(M_1) \otimes_B \pi_B(M_2)$ as its projection. We do not claim that this parent restricts back to the original pair (M_1, M_2): if both children choose different nonzero witness edges for the same pattern edge e, then no parent quasimodel can preserve both choices simultaneously because it stores only one coordinate y_e. The existential interpretation predicate used below needs only the witness-level reconstruction stated in (ii). □

D.3.5 Local law

Lemma 34 (Minor interpretation law)**.** *For every constructor a and child terms $\vec{\sigma}$,*

$$\mathsf{Interp}_{\mathrm{Minor}(H)}(a(\vec{\sigma}), w) \iff \exists \vec{w} \left(\bigwedge_i \mathsf{Interp}_{\mathrm{Minor}(H)}(\sigma_i, w_i) \ \wedge \ w \in b_a(\vec{w}) \right).$$

Proof. **Leaf.** ($\Rightarrow$) If $\mathsf{Interp}_{\mathrm{Minor}(H)}(\texttt{leaf}, w)$ holds, then some quasimodel in the empty graph projects to w. The only such quasimodel has all branch sets empty and all edge witnesses equal to 0, so w is exactly the unique leaf witness. ($\Leftarrow$) Conversely, the all-empty quasimodel in the empty graph projects to that same witness, so $\mathsf{Interp}_{\mathrm{Minor}(H)}(\texttt{leaf}, w)$ holds.

Introduce vertex. Let $\tau = \texttt{intro\_}u(\sigma)$ and let $v := \theta[\tau](u)$ be the introduced isolated vertex. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)$ via a quasimodel $M = \big([X_x], [y_e]\big)$ in $\mathcal{G}(\tau)$. If v belongs to no branch set, then the same quasimodel is already a child quasimodel in $\mathcal{G}(\sigma)$ and w is obtained by the "assign to none" branch of the transition. If $v \in X_x$ for some $x \in V_H$, remove v from X_x and keep all other branch sets and edge witnesses unchanged. The resulting child quasimodel still has pairwise disjoint branch sets, and only the x-component connectivity summary changes, exactly by the `Conn` introduce-vertex transition. Hence the right-hand side holds. ($\Leftarrow$) Conversely, let $\mathsf{Interp}_{\mathrm{Minor}(H)}(\sigma, w_1)$ hold via a child quasimodel M_1. If the transition leaves u unassigned, reuse M_1 as a parent quasimodel. If it assigns u to branch set x, extend only X_x by the new isolated vertex v and leave all edge witnesses unchanged. The induced summary of branch set x changes exactly by the `Conn` introduce-vertex update, so the resulting

parent quasimodel projects to w.

Forget vertex. Let $\tau = \texttt{forget\_}u(\sigma)$. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)$ via a quasimodel M in $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$. Let $w_1 := \pi_{B(\sigma)}(M)$ be the child witness. If $\mathrm{owner}_{w_1}(u) = \bot$, then the parent witness is unchanged. Otherwise, say $\mathrm{owner}_{w_1}(u) = x$. The quasimodel itself does not change when passing from child to parent; only the visible boundary shrinks. Consequently, only the x-component connectivity summary changes, and it changes exactly by the `Conn` forget transition on that component. Hence w lies in the specified forget output of the child witness w_1. ($\Leftarrow$) Conversely, let $\mathsf{Interp}_{\mathrm{Minor}(H)}(\sigma, w_1)$ hold via a child quasimodel M. Since the realized graph is unchanged, the same M is a parent quasimodel for τ. Since $w_1 = \pi_{B(\sigma)}(M)$ is the child witness, if $\mathrm{owner}_{w_1}(u) = \bot$ then the projection is unchanged; if $\mathrm{owner}_{w_1}(u) = x$, the projection changes only by forgetting u from the x-component summary. Thus the parent projection is exactly the witness produced by the transition.

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ and let $f = \{\theta[\tau](u), \theta[\tau](v)\}$ be the introduced graph edge. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)$ via a quasimodel $M = \big([X_x], [y_e]\big)$. If both endpoints have the same owner x, then the quasimodel object itself is unchanged when we pass to the child graph: the branch set X_x is the same subset of vertices, but its induced subgraph loses exactly the edge f. Therefore only the x-component summary changes, exactly by the `Conn` introduce-edge transition. If the endpoints have distinct owners $x \neq y$ and $\{x, y\} \in E_H$, then there are two subcases. If the parent quasimodel does not use f as the witness edge for pattern edge $e = \{x, y\}$, then the same quasimodel already lives in the child graph and the witness is obtained by the skip branch. If instead the parent uses f as y_e, replace y_e by 0 and keep all branch sets and all other witness edges unchanged. The resulting child object is a quasimodel, and the parent witness is obtained from it by the branch that sets $b_e := 1$. In all remaining owner configurations, the quasimodel is unchanged and the edge is ignored. ($\Leftarrow$) Conversely, suppose $\mathsf{Interp}_{\mathrm{Minor}(H)}(\sigma, w_1)$ holds via a child quasimodel M_1. If the transition is the skip branch, keep M_1 as the parent quasimodel. If the transition updates an internal branch-set summary for owner x, again keep the same branch sets and witness edges: only the induced subgraph on X_x gains f, and its summary changes exactly by the `Conn` introduce-edge update. If the transition sets an inter-branch edge flag b_e for $e = \{x, y\}$, keep the branch sets and all other edge witnesses unchanged, and let the parent quasimodel use f as the witness edge for e whenever needed. Since f joins the two appropriate branch sets, the result is a valid parent quasimodel whose projection is w.

Join. Let $\tau = \texttt{join}(\tau_1, \tau_2)$ be a join node with common bag B. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)$ via a parent quasimodel $M \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau))$. Let $M_i := M|_{\tau_i}$ be the canonical restrictions and put $w_i := \pi_B(M_i)$. Then $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau_i, w_i)$ holds for

$i \in \{1, 2\}$, and Lemma 33(i) gives

$$\mathrm{Comp}_B(w_1, w_2) \qquad \text{and} \qquad w \in w_1 \otimes_B w_2.$$

Hence the right-hand side of (INT) holds.

($\Leftarrow$) Conversely, suppose $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau_i, w_i)$ holds for $i \in \{1, 2\}$ and $w \in w_1 \otimes_B w_2$. Choose child quasimodels

$$M_i \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau_i)) \qquad \text{with} \qquad \pi_B(M_i) = w_i.$$

Since $w_1 \otimes_B w_2 = \emptyset$ whenever $\mathrm{Comp}_B(w_1, w_2)$ fails, the assumption $w \in w_1 \otimes_B w_2$ implies $\mathrm{Comp}_B(w_1, w_2)$. Therefore Lemma 33(ii) yields a parent quasimodel

$$M \in \mathcal{X}_{\mathrm{Minor}(H)}(\mathcal{G}(\tau))$$

with $\pi_B(M) = w$. Hence $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)$ holds. □

D.3.6 Global correctness

Theorem 26 (Minor containment correctness)**.** *For fixed simple H without isolated vertices, all k, and all $\tau \in ITD_k$,*

$$\Gamma[\mathrm{Minor}(H), k](\tau) = \mathrm{Sem}_{\mathrm{Minor}(H)}(\tau) = \{w : \mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)\},$$

and

$$\tau \in \mathsf{Acc}(\mathrm{Minor}(H)[k]) \iff H \preceq \mathcal{G}(\tau).$$

Proof. Lemma 34 gives (INT) for the minor core. Since cleaning is the identity, Lemma 27 yields (LSC). Therefore Theorems 17 and 19 give

$$\Gamma[\mathrm{Minor}(H), k](\tau) = \mathrm{Sem}_{\mathrm{Minor}(H)}(\tau) = \{w : \mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)\}.$$

For the acceptance claim, write $G := \mathcal{G}(\tau)$.

($\Rightarrow$) Suppose τ is accepted. Then some final witness $w \in \mathrm{Sem}_{\mathrm{Minor}(H)}(\tau)$ is realized by a quasimodel

$$M = \big([X_x]_{x \in V_H}, [y_e]_{e \in E_H}\big) \in \mathcal{X}_{\mathrm{Minor}(H)}(G).$$

Finality says that $b_e = 1$ for every pattern edge e, so every y_e is a nonzero edge joining the appropriate two branch sets. Finality also says that each component (γ_x, P_x) is final for $\mathtt{Conn}$, hence $\mathcal{G}(\tau)[X_x]$ is either empty or connected. Since H has no isolated vertices, every $x \in V_H$ is incident with some $e \in E_H$; because $b_e = 1$, the witness edge y_e has an

endpoint in X_x, so $X_x \neq \emptyset$. Thus every branch set is nonempty and connected, and the quasimodel is actually a minor model of H in G. Hence $H \preceq G$.

($\Leftarrow$) Conversely, suppose $H \preceq G$. Fix a minor model

$$M = \big([X_x]_{x \in V_H}, [y_e]_{e \in E_H}\big)$$

of H in G, so each X_x is nonempty and connected and every y_e is a nonzero edge joining the corresponding two branch sets. Let

$$w := \pi_{B(\tau)}(M).$$

Then $\mathsf{Interp}_{\mathrm{Minor}(H)}(\tau, w)$ holds, so $w \in \mathrm{Sem}_{\mathrm{Minor}(H)}(\tau)$. Because every X_x is connected and nonempty, the corresponding `Conn` summary is final. Because every y_e is nonzero, every edge flag satisfies $b_e = 1$. Hence w is a final witness, so τ is accepted. □

D.3.7 Complexity summary

As recorded in Table 5.1 and Appendix D.1,

$$\beta_{\mathrm{Minor}(H)}(k) = O(k \log k + |V(H)| + |E(H)|), \qquad \mu_{\mathrm{Minor}(H)}(k) \leq 2^{O(\beta_{\mathrm{Minor}(H)}(k))}.$$

For fixed H, this yields the standard finite-core bound $n \cdot 2^{O(k \log k)}$ for fixed-term evaluation. The local witness language and all constructor operations are also polynomial-time for fixed H; see Lemma 35 below.

Lemma 35 (Appendix catalog cores are internally polynomial)**.** *Under the witness encodings fixed in this appendix, the cores* Conn, Minor(H) *for fixed H, $\mathcal{C}_d^{\Delta}$, $\mathcal{C}_\ell^{\omega}$, $\mathcal{C}^{\mathrm{ME}}$, $\mathcal{C}^{\mathrm{PM}}$, $\mathcal{C}^{\mathrm{HP}}$, $\mathcal{C}^{\mathrm{HC}}$, $\mathcal{C}_t^{\mathrm{FVS}}$, $\mathcal{C}^{\alpha}$, and $\mathcal{C}^{\mathrm{VCNT}}$ are internally polynomial.*

Proof. For each listed core, a witness is encoded by a tuple of bag-indexed finite data (bitmasks, truncated counters, partial matchings, partitions, endpoint types, or flags) together with at most one integer parameter encoded in binary. Membership in the witness language is therefore decidable by polynomial-time checks on these encodings.

The constructor transitions in the corresponding core definitions are explicit local updates of that data: they inspect the current bag, modify finitely many coordinates, perform bounded set/partition/matching bookkeeping on structures of size at most $k+1$, and apply at most polynomial-time arithmetic on the binary counters. Cleaning is either the identity, a deterministic functional update, or a signature-wise minimum/maximum selection over a finite witness set; each of these is computable in time polynomial in the total input size. Finality and invariant evaluation are likewise polynomial-time tests or scans over the encoded witness set. Finally, for the numeric invariant cores $\mathcal{C}^{\alpha}$ and $\mathcal{C}^{\mathrm{VCNT}}$,

the invariant output is obtained by scanning the stored binary counters and returning a binary integer bounded by the term size, hence has polynomial bitlength in the witness encoding. Therefore each listed core satisfies Definition 33.

For the two auxiliary cores used inside the minor construction, the same analysis is completely explicit. For Conn, a witness is just a partition of a set of size at most $k+1$ together with a constant-size flag, and every transition is a bounded update of that partition data; in particular, the join transition computes a transitive-closure merge on a universe of size $O(k)$. For Minor(H) with fixed H, a witness consists of $|V(H)|$ many `Conn`-summaries together with $|E(H)|$ bits. Membership, finality, and root acceptance are polynomial-time scans of this explicit tuple. The introduce/forget/edge constructors update at most one `Conn` component and finitely many edge bits, while the join constructor performs one `Conn`-join for each $x \in V(H)$ and a bitwise maximum over the $|E(H)|$ pattern-edge flags. Since H is fixed, all of this is polynomial in the total witness encoding size.

The only nontrivial point is the join transition for the path/cycle/feedback catalog cores. For $\mathcal{C}^{\mathrm{HP}}$ and $\mathcal{C}^{\mathrm{HC}}$, a witness consists of a degree profile on the current bag, a partial pairing of bag endpoints, and a constant-size flag. The join routine builds an auxiliary interface graph/diagram whose vertices and edges are indexed by the at most $k+1$ bag labels and by the finitely many endpoint stubs encoded in the two child witnesses. Checking degree feasibility, suppressing internalized bag vertices, testing whether a cycle is created, and extracting the resulting pairing are all standard graph operations on an object of size polynomial in the witness encoding, hence run in polynomial time. For $\mathcal{C}_t^{\mathrm{FVS}}$, the join routine similarly combines two bag-indexed annotations and partitions on a universe of size $O(k)$, then performs compatibility and acyclicity checks via union-find / transitive-closure style computations on the resulting auxiliary graph; these are again polynomial in the total witness encoding size. Thus the join-heavy cores do not hide any superpolynomial subroutine. □

D.4 Maximum Degree ($\geq d$)

This section discharges the correctness proof for the maximum-degree DP-core $\mathcal{C}_d^{\Delta}$ from Section 4.8.2 and instantiates the proof framework of Section 9.1. Throughout this section we work in the natural regime $d \geq 1$, since the threshold $d = 0$ is degenerate.

D.4.1 Witness space, cleaning, and acceptance

At bag B a witness is either the flag `Found` or a degree map $\phi : B \to \{0, \ldots, d-1\}$. Transitions maintain degree profiles along the decomposition and promote to `Found` as soon as a vertex of degree at least d is detected. Cleaning is the identity. Acceptance at

the root is the presence of `Found`.

D.4.2 Transitions as set-valued functions

We restate the transitions here (with the catalog anchor at Section 4.8.2). In set-lifted form, every constructor transition is functional: Leaf returns the empty degree map, introduce-vertex extends by $u \mapsto 0$, forget-vertex restricts to the smaller bag, introduce-edge increments the two endpoint counters (or promotes to `Found` if some counter reaches $\geq d$), and join adds the two child degree profiles pointwise on the common bag (again promoting if some coordinate reaches $\geq d$).

D.4.3 Denotation and the local interpretation predicate

For $\tau \in \mathsf{ITD}_k$ with realized graph $\mathcal{G}(\tau)$ and top bag $B(\tau)$, set

$$\mathsf{Interp}_{\mathcal{C}_d^\Delta}(\tau, w) \iff \mathsf{P}_{d,k}^\Delta(\tau, w), \qquad \mathrm{Sem}_{\mathcal{C}_d^\Delta}(\tau) := \{w : \mathsf{Interp}_{\mathcal{C}_d^\Delta}(\tau, w)\}.$$

Unwinding $\mathsf{P}_{d,k}^\Delta$: either $w = \texttt{Found}$ and some vertex of $\mathcal{G}(\tau)$ has degree $\geq d$, or w is the unique map $B(\tau) \to \{0, \ldots, d-1\}$ giving the (strictly smaller) bag degrees.

D.4.4 Local law (INT / LSC)

We prove the constructor-wise interpretation law for the maximum-degree core.

Lemma 36 (Max degree INT)**.** *For each constructor a and children $(\sigma_i)_i$ such that $\tau := a(\vec{\sigma}) \in \mathsf{ITD}_k$,*

$$\mathsf{Interp}_{\mathcal{C}_d^\Delta}\big(a(\vec{\sigma}), w\big) \iff \exists \vec{w} \, \Big(\bigwedge_i \mathsf{Interp}_{\mathcal{C}_d^\Delta}(\sigma_i, w_i) \ \wedge \ w \in \widehat{a}(\vec{w}) \Big).$$

Proof. We prove the equivalence by cases on a, writing $\tau := a(\vec{\sigma})$. By the definition of $\mathsf{Interp}_{\mathcal{C}_d^\Delta}(\tau, w)$, either $w = \texttt{Found}$ and $\Delta(\mathcal{G}(\tau)) \geq d$, or w is the unique degree map on $B(\tau)$ when $\Delta(\mathcal{G}(\tau)) < d$.

Leaf. $\mathcal{G}(\texttt{leaf})$ has no vertices, so $\Delta(\mathcal{G}(\texttt{leaf})) = 0 < d$ and the unique witness is the empty degree map $\emptyset : B(\texttt{leaf}) = \emptyset \to \{0, \ldots, d-1\}$. Thus $\mathsf{Interp}(\texttt{leaf}, w)$ holds iff $w = \emptyset$, which is equivalent to $w \in \widehat{\texttt{leaf}}()$.

Introduce vertex. Let $\tau = \texttt{intro\_}u(\sigma)$ and let $x := \theta[\tau](u)$ be the introduced vertex. Then x is isolated in $\mathcal{G}(\tau)$, so $\deg_{\mathcal{G}(\tau)}(x) = 0$ and all other degrees are unchanged. ($\Rightarrow$) If $w = \texttt{Found}$ then $\Delta(\mathcal{G}(\tau)) \geq d$, hence $\Delta(\mathcal{G}(\sigma)) \geq d$ and $\mathsf{Interp}(\sigma, \texttt{Found})$ holds. By definition the introduce-vertex transition propagates the flag, so $w \in \widehat{\texttt{intro\_}u}(\texttt{Found})$. If $w = \phi$ is a degree map on $B(\tau)$, then $\Delta(\mathcal{G}(\tau)) < d$, so also $\Delta(\mathcal{G}(\sigma)) < d$ and the unique

child degree map is $\phi|_{B(\sigma)}$. Moreover $\phi(u) = 0$ because x is isolated, so w is exactly the extension produced by the introduce-vertex transition. ($\Leftarrow$) Conversely, if $\mathsf{Interp}(\sigma, w_1)$ and $w \in \widehat{\texttt{intro\_}u}(w_1)$, then either $w_1 = \texttt{Found}$ and the flag remains correct in $\mathcal{G}(\tau)$, or w_1 is the child degree map and w is its extension by $u \mapsto 0$, which matches degrees in $\mathcal{G}(\tau)$.

Forget vertex. Let $\tau = \texttt{forget\_}u(\sigma)$, so $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$ and $B(\tau) = B(\sigma) \setminus \{u\}$. ($\Rightarrow$) If $w = \texttt{Found}$ then $\Delta(\mathcal{G}(\sigma)) = \Delta(\mathcal{G}(\tau)) \geq d$, so $\mathsf{Interp}(\sigma, \texttt{Found})$ and the transition propagates the flag. If $w = \phi$ is a degree map on $B(\tau)$, then $\Delta(\mathcal{G}(\tau)) < d$ and hence also $\Delta(\mathcal{G}(\sigma)) < d$. Let ϕ' be the unique degree map on $B(\sigma)$. Then $\phi = \phi'|_{B(\tau)}$, so $w \in \widehat{\texttt{forget\_}u}(\phi')$. ($\Leftarrow$) Conversely, if $\mathsf{Interp}(\sigma, w_1)$ and $w \in \widehat{\texttt{forget\_}u}(w_1)$, then either $w_1 = \texttt{Found}$ and $\mathsf{Interp}(\tau, \texttt{Found})$ holds because $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$, or $w_1 = \phi'$ is the child degree map and $w = \phi'|_{B(\tau)}$, which is the unique correct degree map on $B(\tau)$.

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ and let $e := \{\theta[\tau](u), \theta[\tau](v)\}$ be the introduced edge. Only the degrees of the endpoints of e change, each increasing by 1 (counting edge multiplicity). ($\Rightarrow$) If $w = \texttt{Found}$ then either $\Delta(\mathcal{G}(\sigma)) \geq d$ (so the child is already $\texttt{Found}$) or else $\Delta(\mathcal{G}(\sigma)) < d$ and the new edge causes one of the endpoint degrees to become $\geq d$. In the latter case, letting ϕ be the unique child degree map, the introduce-edge transition on ϕ detects the threshold and returns $\texttt{Found}$. If $w = \phi'$ is a degree map, then $\Delta(\mathcal{G}(\tau)) < d$ and hence $\Delta(\mathcal{G}(\sigma)) < d$. Writing ϕ for the unique child degree map, we have $\phi' = \phi$ except that $\phi'(u) = \phi(u) + 1$ and $\phi'(v) = \phi(v) + 1$, and this is exactly the introduce-edge update. ($\Leftarrow$) Conversely, if $\mathsf{Interp}(\sigma, w_1)$ and $w \in \widehat{\texttt{intro\_}u\texttt{\_}v}(w_1)$, then either $w_1 = \texttt{Found}$ and $w = \texttt{Found}$, or $w_1 = \phi$ is the child degree map and the transition either produces an updated map ϕ' (when both endpoint degrees stay $< d$) or produces $\texttt{Found}$ (when some endpoint reaches $\geq d$). In all cases, the produced w matches $\mathsf{P}^{\Delta}_{d,k}$ on $\mathcal{G}(\tau)$.

Join. Let $\tau = \texttt{join}(\sigma_1, \sigma_2)$ be a join node with common bag B. By the semantics of join nodes, $\mathcal{G}(\tau)$ is the pushout of $\mathcal{G}(\sigma_1)$ and $\mathcal{G}(\sigma_2)$ along the bag vertices, so every non-bag vertex belongs to exactly one child graph, while degrees of bag vertices add: for each bag label $b \in B$,

$$\deg_{\mathcal{G}(\tau)}(\theta[\tau](b)) = \deg_{\mathcal{G}(\sigma_1)}(\theta[\sigma_1](b)) + \deg_{\mathcal{G}(\sigma_2)}(\theta[\sigma_2](b)).$$

($\Rightarrow$) If $w = \texttt{Found}$, then either some child already has maximum degree $\geq d$ (so some $w_i = \texttt{Found}$) or else both child maxima are $< d$ but some bag vertex reaches degree $\geq d$ only after summing the two child degrees. In the first case, the join transition propagates the flag. In the second case, both children have degree maps φ_1, φ_2 and the join transition detects $\exists b \in B : \varphi_1(b) + \varphi_2(b) \geq d$ and returns $\texttt{Found}$. If $w = \varphi$ is a degree map, then $\Delta(\mathcal{G}(\tau)) < d$, so both children have degree maps φ_1, φ_2 and $\varphi = \varphi_1 + \varphi_2$ pointwise

on B, which is exactly the join transition output. ($\Leftarrow$) Conversely, if $\mathsf{Interp}(\sigma_i, w_i)$ and $w \in \widehat{\texttt{join}}(w_1, w_2)$, then the join transition returns $w = \texttt{Found}$ exactly when some $w_i = \texttt{Found}$ or when both $w_i = \varphi_i$ are degree maps with a bag coordinate sum $\geq d$. In either case $\Delta(\mathcal{G}(\tau)) \geq d$ by the degree-additivity equation above. Otherwise it returns the sum map $\varphi_1 + \varphi_2$, and all degrees remain $< d$, so this sum is the unique correct degree map on B. □

Thus (INT) holds. Since cleaning is the identity, (LSC) follows, and Theorem 17 gives equality of dynamic and denotational semantics.

D.4.5 From local law to global correctness

We now derive global correctness and record the resulting witness bounds.

Theorem 27. *For every $d \geq 1$ and all k, $\Gamma[\mathcal{C}_d^\Delta, k] = \mathrm{Sem}_{\mathcal{C}_d^\Delta}$. Moreover, for every term $\tau \in \mathsf{ITD}_k$, we have*

$$\tau \in \mathsf{Acc}(\mathcal{C}_d^\Delta[k]) \quad \Longleftrightarrow \quad \Delta(\mathcal{G}(\tau)) \geq d.$$

Each witness can be encoded in $\beta = O((k{+}1)\log(d+1))$ bits. All transitions are functional, so every node's witness set is a singleton ($\mu = 1$). As a coarse range bound, there are at most $d^{k+1} + 1$ possible witnesses at a bag of size $\leq k{+}1$.

Proof. By Lemma 36 we have (INT) for every constructor. Since cleaning is the identity, (LSC) follows and Theorem 17 yields $\Gamma[\mathcal{C}_d^\Delta, k] = \mathrm{Sem}_{\mathcal{C}_d^\Delta}$. Fix $\tau \in \mathsf{ITD}_k$. At the root, $\mathcal{C}_d^\Delta$ accepts τ iff $\texttt{Found} \in \Gamma[\mathcal{C}_d^\Delta, k](\tau)$, which holds exactly when some vertex reaches degree $\geq d$, i.e. $\Delta(\mathcal{G}(\tau)) \geq d$.

For encoding, a degree map $\phi : B \to \{0, \ldots, d{-}1\}$ on a bag of size $m \leq k{+}1$ is determined by m values in $\{0, \ldots, d-1\}$, each representable in $O(\log(d+1))$ bits. Adding one flag bit for $\texttt{Found}$ gives $\beta = O((k+1)\log(d+1))$.

Each constructor updates the witness deterministically. Hence every transition returns a singleton set, and every cleaned witness set is a singleton. In particular, $\mu = 1$. Finally, there are at most d^m degree maps on a bag of size $m \leq k+1$, plus the flag $\texttt{Found}$. Therefore $d^{k+1} + 1$ is a coarse upper bound on the number of possible witnesses. □

D.4.6 Equivariance and coherence

Equivariance is immediate: relabelings act by renaming bag coordinates in degree maps and fix the flag $\texttt{Found}$, and each transition commutes with this action. Hence, by Corollary 14, the induced property is well-defined (independent of the chosen decomposition).

D.4.7 Bitlength, multiplicity, and running time

The encoding and multiplicity bounds are recorded in Theorem 27: $\beta = O((k+1)\log(d+1))$ and $\mu = 1$ (all transitions are functional). By Lemma 35, Corollary 2 applies, and the generic runtime bound follows by instantiating it with these parameters.

D.5 Clique of Size ℓ

We recall the clique DP-core $\mathcal{C}_\ell^\omega$ from Section 4.8.2 and now verify its correctness within the framework of Section 9.1. Throughout, fix an integer $k \in \mathbb{N}$, a threshold $\ell \geq 2$, and a bag $B \subseteq [k+1]$. The degenerate case $\ell = 1$ is excluded from this DP-core proof: on simple graphs it is trivial, since a graph contains a clique of size at least 1 if and only if it has a vertex.

D.5.1 Witness space, cleaning, and acceptance

A local witness is either the flag `Found` or a pair (ψ, t) where $t \in \{0, \ldots, \ell\}$ and ψ is a partial map

$$\psi : B \rightharpoonup \{0, \ldots, \ell - 1\}.$$

Intuitively, `Found` records that an ℓ-clique has already been found. Otherwise, (ψ, t) represents a *clique candidate* C of size t inside the subgraph built below the current node:

- the domain $\mathrm{dom}(\psi)$ consists of those bag vertices that already belong to C, and
- for $u \in \mathrm{dom}(\psi)$, the counter $\psi(u)$ records how many neighbors of the vertex labeled by u inside C have been connected so far (capped at $\ell - 1$).

Cleaning collapses to $\{$`Found`$\}$ whenever `Found` is present, and is the identity otherwise. The invariant map $\mathtt{Inv}_k \equiv 0$ is constant. Acceptance is the presence, at the root, of `Found`.

D.5.2 Transitions as set-valued functions

Let B be the current bag. Restating the transition component of $\mathcal{C}^{\omega}_{\ell}[k]$ in the set-lifted form $\widehat{a} : W_k^r \to \mathcal{P}_{\mathrm{fin}}(W_k)$, we have:

$$\begin{aligned}
\mathcal{C}^{\omega}_{\ell}[k].\texttt{LeafSet} &= \{(\emptyset, 0)\},\\
\mathcal{C}^{\omega}_{\ell}[k].\texttt{intro_}u(\texttt{Found}) &= \{\texttt{Found}\},\\
\mathcal{C}^{\omega}_{\ell}[k].\texttt{intro_}u(\psi, t) &= \{(\psi, t)\}\\
&\quad \cup \{(\psi \cup \{u \mapsto 0\},\, t+1) \mid t < \ell \text{ and } t = |\operatorname{dom}(\psi)|\},\\
\mathcal{C}^{\omega}_{\ell}[k].\texttt{forget_}u(\texttt{Found}) &= \{\texttt{Found}\},\\
\mathcal{C}^{\omega}_{\ell}[k].\texttt{forget_}u(\psi, t) &= \begin{cases} \{(\psi|_{B\setminus\{u\}},\, t)\}, & u \notin \operatorname{dom}(\psi),\\ \{(\psi|_{B\setminus\{u\}},\, t)\}, & t = \ell \text{ and } \psi(u) = \ell - 1,\\ \emptyset, & \text{otherwise,} \end{cases}\\
\mathcal{C}^{\omega}_{\ell}[k].\texttt{intro_}u_v(\texttt{Found}) &= \{\texttt{Found}\},\\
\mathcal{C}^{\omega}_{\ell}[k].\texttt{intro_}u_v(\psi, t) &= \begin{cases} \{\texttt{Found}\}, & \substack{t=\ell \\ \forall b \in \operatorname{dom}\psi', \\ \psi'(b) = \ell-1}\\ \{(\psi', t)\}, & \text{otherwise,} \end{cases}
\end{aligned}$$

where $\psi' = \psi$ if $u \notin \operatorname{dom}(\psi)$ or $v \notin \operatorname{dom}(\psi)$, and otherwise ψ' agrees with ψ outside $\{u, v\}$ and increments both counters $\psi(u)$ and $\psi(v)$ by 1, capped at $\ell - 1$. Finally, joins:

$$\begin{aligned}
\mathcal{C}^{\omega}_{\ell}[k].\texttt{join}(\texttt{Found}, w) &= \{\texttt{Found}\} \quad \text{(and symmetrically)},\\
\mathcal{C}^{\omega}_{\ell}[k].\texttt{join}\big((\psi, t), (\psi', t')\big) &= \begin{cases} \{\texttt{Found}\}, & \text{if } \substack{\operatorname{dom}(\psi)=\operatorname{dom}(\psi') \\ (t=|\operatorname{dom}(\psi)| \vee t'=|\operatorname{dom}(\psi)|) \\ t_{\mathrm{new}}=\ell \\ \forall b \in \operatorname{dom}(\psi):\ \psi''(b)=\ell-1},\\ \{(\psi'',\, t_{\mathrm{new}})\}, & \text{if } \substack{\operatorname{dom}(\psi)=\operatorname{dom}(\psi') \\ (t=|\operatorname{dom}(\psi)| \vee t'=|\operatorname{dom}(\psi)|)},\\ \emptyset, & \text{otherwise,} \end{cases}
\end{aligned}$$

where $t_{\mathrm{new}} := t + t' - |\operatorname{dom}(\psi)|$. The guard ($t = |\operatorname{dom}(\psi)|$ or $t' = |\operatorname{dom}(\psi)|$) ensures that at least one candidate is entirely contained within the bag (no forgotten vertices), enforcing that the clique cannot pick up non-bag vertices from both sides (which would be disconnected in the pushout). Whenever the join transition returns a nonempty set we have $\operatorname{dom}(\psi) = \operatorname{dom}(\psi') \subseteq B$. The merged counter map ψ'' is defined on this common domain by

$$\psi''(b) = \min\{\ell - 1,\ \psi(b) + \psi'(b)\} \quad \text{for } b \in \operatorname{dom}(\psi).$$

D.5.3 Denotation and the local interpretation predicate

For a graph G let

$$X_\ell^{\mathrm{cand}}(G) := \{\, C \subseteq V(G) : |C| \le \ell \,\}.$$

Let σ be a term with bag $B(\sigma)$ and labeling $\theta[\sigma]$. For $C \in X_\ell^{\mathrm{cand}}(\mathcal{G}(\sigma))$ with $|C| = t \le \ell$, define

$$\pi_{B(\sigma)}(C) = (\psi_C, t),$$

where $\mathrm{dom}(\psi_C) = \theta[\sigma]^{-1}(C) \cap B(\sigma)$ and for u in this domain,

$$\psi_C(u) = \big|\{\, v \in C \setminus \{\theta[\sigma](u)\} : \{\theta[\sigma](u), v\} \in E(\mathcal{G}(\sigma)) \,\}\big|.$$

Set

$$\mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\sigma) := \{\, w : \mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\sigma, w) \,\}$$

and define $\mathsf{Interp}_{\mathcal{C}_\ell^\omega}$ by cases:

$$\mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\sigma, \mathtt{Found}) \iff \mathcal{G}(\sigma) \text{ contains a clique of size } \ell,$$

and for pairs (ψ, t),

$$\begin{aligned}
\mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\sigma, (\psi, t)) \iff\ & \mathcal{G}(\sigma) \text{ contains no clique of size } \ell \text{ and} \\
& \exists C \in X_\ell^{\mathrm{cand}}(\mathcal{G}(\sigma)) \text{ such that} \\
& \quad \pi_{B(\sigma)}(C) = (\psi, t), \\
& \quad t < \ell \ \Rightarrow\ t = |\,\mathrm{dom}(\psi)|, \\
& \quad \forall v \in C \setminus \theta[\sigma](B(\sigma)) : \\
& \qquad \forall w \in C \setminus \{v\} :\ \{v, w\} \in E(\mathcal{G}(\sigma)).
\end{aligned}$$

Comparing with the semantic predicate $\mathsf{P}_{\ell,k}^\omega$ from Section 4.8.2, we see that for $\tau \in \mathsf{ITD}_k$,

$$\mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\tau, w) \quad \iff \quad \mathsf{P}_{\ell,k}^\omega(\tau, w).$$

Convention 3 (Clique core is scoped to simple realized graphs). In this subsection we interpret "clique" in the standard (simple-graph) sense. Since an instructive term may in general realize a multigraph with parallel edges, the correctness statement for $\mathcal{C}_\ell^\omega$ is understood under the promise that $\mathcal{G}(\tau)$ is simple (i.e. has no parallel edges). Equivalently, when we want an unconditional graph property, we evaluate the clique core only on inputs that also satisfy the "no multi-edge" property. This promise can be enforced by the multi-edge detector core $\mathcal{C}^{\mathrm{ME}}$ (Section D.6), e.g. accept iff $\mathcal{C}_\ell^\omega$ accepts and $\mathcal{C}^{\mathrm{ME}}$ rejects.

D.5.4 Local law (INT / LSC)

We prove the constructor-wise local law for the clique core within the simple-graph regime of Convention 3.

Lemma 37 (Clique LSC)**.** *For each constructor a of the instructive alphabet and all children $(\sigma_i)_i$, writing $\tau := a(\vec{\sigma})$, if $\mathcal{G}(\tau)$ is simple then*

$$\mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\tau) = \mathsf{F}_{a,B(\tau)}\big(\mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\sigma_1), \dots, \mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\sigma_r)\big),$$

where $r = \mathsf{r}(a)$.

Proof. Assume throughout that $\mathcal{G}(\tau)$ is simple. Then each child graph $\mathcal{G}(\sigma_i)$ is also simple, since it is a subgraph of $\mathcal{G}(\tau)$.

Write

$$S_i := \mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\sigma_i) \qquad (1 \le i \le r)$$

and

$$U_a := \bigcup_{\vec{w} \in \prod_i S_i} \widehat{a}(\vec{w}).$$

By definition of the cleaned algebra operation,

$$\mathsf{F}_{a,B(\tau)}(S_1, \dots, S_r) = \mathtt{Clean}_k(U_a).$$

So it is enough to prove

$$\mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\tau) = \mathtt{Clean}_k(U_a).$$

Case 1: $\mathcal{G}(\tau)$ contains an ℓ-clique. Then, by definition of $\mathsf{Interp}_{\mathcal{C}_\ell^\omega}$,

$$\mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\tau) = \{\mathtt{Found}\}.$$

We show that $\mathtt{Found} \in U_a$. Since cleaning collapses any set containing `Found` to the singleton $\{\mathtt{Found}\}$, this will imply the required equality.

For `leaf` this case cannot occur, and $\widehat{\mathtt{leaf}}()$ does not produce `Found`. For $\ell \ge 2$, the constructors `intro_`u and `forget_`u do not create new adjacencies among pre-existing vertices, and a newly introduced isolated vertex cannot by itself form an ℓ-clique. Hence these constructors can only preserve an already existing ℓ-clique, which is exactly why their found branch is pure propagation.

For `intro_`u`_`v, there are two possibilities. Either the ℓ-clique was already present in the child graph, in which case the child denotation is $\{\mathtt{Found}\}$ and the flag is propagated. Or the new edge uv is part of some ℓ-clique C in the parent. Since uv is the only new adjacency created at this node, all vertices of C were already present in the child term

and every edge of C other than uv was already realized there. Thus the child must already contain the corresponding size-ℓ candidate, with all counters complete except for the adjacency between u and v; adding uv raises the missing endpoint counters to $\ell - 1$, and this is exactly the condition under which the transition promotes to `Found`.

For `join`, either some child already contains an ℓ-clique, so again the flag is propagated, or the clique is assembled only after gluing the two child subgraphs along the bag. In the latter case the clique cannot use forgotten vertices from both sides of the join: vertices outside the bag belong to different pushout summands and are therefore nonadjacent across the join. Let $\iota_i : \mathcal{G}(\sigma_i) \hookrightarrow \mathcal{G}(\tau)$ be the pushout embeddings, let C be an ℓ-clique of the parent, and define $C_i := \iota_i^{-1}(C) \subseteq V(\mathcal{G}(\sigma_i))$. Then at least one of C_1, C_2 is contained in the bag, so the guard $t_1 = |\operatorname{dom}(\psi_1)|$ or $t_2 = |\operatorname{dom}(\psi_2)|$ is satisfied for the projected witnesses. Moreover, the merged candidate has size $t_{\text{new}} = |C| = \ell$, and every selected bag vertex has exactly $\ell - 1$ neighbors inside C, so the merged counter map reaches the saturated value $\ell - 1$ on every bag coordinate. Hence the join transition produces `Found`.

Case 2: $\mathcal{G}(\tau)$ contains no ℓ-clique. Then $\mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\tau)$ contains only pair witnesses. Also, no child denotation contains `Found`, since every child graph is a subgraph of the parent simple graph. Thus every tuple in $\prod_i S_i$ is pair-valued. For a pair witness (ψ, t), we claim that

$$\mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\tau, (\psi, t)) \iff \exists \vec{w} \in \prod_i S_i \text{ such that } (\psi, t) \in \widehat{a}(\vec{w}).$$

We verify this by cases on a.

Leaf: G empty, projection $(\emptyset, 0)$ matches `LeafSet`.

Introduce: if the new vertex is not used in a candidate set C, the projection is unchanged. If it is used, a fresh counter 0 is added for the corresponding bag label and t increases by 1. This is allowed exactly when the candidate currently contains no forgotten vertices (the guard $t = |\operatorname{dom}(\psi)|$).

Edge: introducing an edge between two selected bag vertices increments both counters by 1 and leaves t unchanged.

Forget: if $u \notin \operatorname{dom}(\psi)$, forgetting u only changes the bag interface, so the projection restricts to $B \setminus \{u\}$. If $u \in \operatorname{dom}(\psi)$, then necessarily $t = \ell$ (since for $t < \ell$ we require $t = |\operatorname{dom}(\psi)|$), and u becomes a non-bag vertex. Hence u must already be adjacent in $\mathcal{G}(\sigma)$ to every other vertex of C (no future edges incident to u can be introduced), which is equivalent to requiring $\psi(u) = \ell - 1$, exactly as in the forget guard.

Join (pairs). Let $\tau = \texttt{join}(\sigma_1, \sigma_2)$ be a join node with common bag B and let $\iota_i : \mathcal{G}(\sigma_i) \hookrightarrow \mathcal{G}(\tau)$ be the pushout embeddings. We prove the equivalence directly.

($\Rightarrow$) Assume $\mathsf{Interp}_{\mathcal{C}^\omega_\ell}(\tau, (\psi'', t''))$, witnessed by a candidate set $C \subseteq V(\mathcal{G}(\tau))$ of size t''. Let $C_i := \iota_i^{-1}(C) \subseteq V(\mathcal{G}(\sigma_i))$ and write $\pi_B(C_i) = (\psi_i, t_i)$. By definition of π_B we have $\mathrm{dom}(\psi_1) = \mathrm{dom}(\psi_2) = \theta[\tau]^{-1}(C) \cap B$, and $\pi_B(C)$ has the same domain. Moreover, C cannot contain vertices outside $\theta[\tau](B)$ from *both* sides: if $x \in C \setminus \iota_1(V(\mathcal{G}(\sigma_1)))$ and $y \in C \setminus \iota_2(V(\mathcal{G}(\sigma_2)))$, then x and y lie in different pushout summands and are nonadjacent in $\mathcal{G}(\tau)$, contradicting the clique condition in $\mathsf{Interp}_{\mathcal{C}^\omega_\ell}$. Hence at least one of C_1, C_2 is contained in the bag, equivalently $t_1 = |\,\mathrm{dom}(\psi_1)|$ or $t_2 = |\,\mathrm{dom}(\psi_2)|$. Finally, fix $b \in \mathrm{dom}(\psi_1)$ and write $v := \theta[\tau](b)$. Every neighbor of v in C is either a non-bag vertex (hence belongs to exactly one join child, so the corresponding incident edge is counted in exactly one of $\psi_1(b), \psi_2(b)$) or another bag vertex. Because $\mathcal{G}(\tau)$ is simple, no edge between two bag vertices can occur in both join children (otherwise the pushout would contain a parallel edge). Therefore the neighbor counter in C splits additively across the two restrictions:

$$\psi''(b) = \min\{\ell - 1, \psi_1(b) + \psi_2(b)\}.$$

Since C is the pushout union of C_1 and C_2 with overlap exactly the bag part, we also have $t'' = t_1 + t_2 - |\,\mathrm{dom}(\psi_1)|$. Thus $(\psi'', t'') \in \widehat{\texttt{join}}\big((\psi_1, t_1), (\psi_2, t_2)\big)$.

($\Leftarrow$) Conversely, assume there exist witnesses (ψ_1, t_1), (ψ_2, t_2) with $\mathsf{Interp}_{\mathcal{C}^\omega_\ell}(\sigma_i, (\psi_i, t_i))$ and $(\psi'', t'') \in \widehat{\texttt{join}}((\psi_1, t_1), (\psi_2, t_2))$. Choose candidate sets $C_i \subseteq V(\mathcal{G}(\sigma_i))$ witnessing $\mathsf{Interp}_{\mathcal{C}^\omega_\ell}(\sigma_i, (\psi_i, t_i))$. The join transition requires $\mathrm{dom}(\psi_1) = \mathrm{dom}(\psi_2)$ and $t_1 = |\,\mathrm{dom}(\psi_1)|$ or $t_2 = |\,\mathrm{dom}(\psi_2)|$, so all vertices of (at least) one C_i lie in the bag. Let $C := \iota_1(C_1) \cup \iota_2(C_2) \subseteq V(\mathcal{G}(\tau))$ be the pushout union. Then $|C| = t_1 + t_2 - |\,\mathrm{dom}(\psi_1)| = t''$, and C contains no non-bag vertices from both sides, so the clique condition in $\mathsf{Interp}_{\mathcal{C}^\omega_\ell}(\tau, (\psi'', t''))$ holds (all non-bag vertices come from the same side and are adjacent to every other vertex in C by the child predicate). The counters satisfy $\psi''(b) = \min\{\ell - 1, \psi_1(b) + \psi_2(b)\}$ as above, so $\pi_B(C) = (\psi'', t'')$. It remains to check the side condition $t'' < \ell \Rightarrow t'' = |\,\mathrm{dom}(\psi'')|$. By symmetry we may assume $t_1 = |\,\mathrm{dom}(\psi_1)|$, so $t'' = t_2$. If $t'' < \ell$, then also $t_2 < \ell$, and since $\mathsf{Interp}_{\mathcal{C}^\omega_\ell}(\sigma_2, (\psi_2, t_2))$ holds, the child side condition gives $t_2 = |\,\mathrm{dom}(\psi_2)| = |\,\mathrm{dom}(\psi'')|$. Hence $\mathsf{Interp}_{\mathcal{C}^\omega_\ell}(\tau, (\psi'', t''))$.

In each case we obtain the stated equivalence for pairs. Finally, the `Found`-branch is impossible in the no-clique case: every flag transition either propagates a flagged child (already ruled out) or certifies a newly completed ℓ-clique in the parent graph, contradicting the present assumption. Therefore U_a contains exactly the pair witnesses in $\mathrm{Sem}_{\mathcal{C}^\omega_\ell}(\tau)$, so $\texttt{Clean}_k(U_a) = U_a = \mathrm{Sem}_{\mathcal{C}^\omega_\ell}(\tau)$. □

D.5.5 From local law to global correctness

Theorem 28 (Clique correctness)**.** *For every k, every $\ell \geq 2$, and every term $\tau \in \mathsf{ITD}_k$ with $\mathcal{G}(\tau)$ simple,*

$$\Gamma[\mathcal{C}_\ell^\omega, k](\tau) = \mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\tau) = \{w : \mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\tau, w)\},$$

and $\tau \in \mathsf{Acc}(\mathcal{C}_\ell^\omega[k])$ iff $\mathcal{G}(\tau)$ contains a clique of size ℓ.

Proof. Every subterm σ of τ also has $\mathcal{G}(\sigma)$ simple, since realized subterm graphs are subgraphs of $\mathcal{G}(\tau)$. Hence Lemma 37 establishes (LSC) for every constructor occurrence in the evaluation of $\mathcal{C}_\ell^\omega$ on τ. Moreover, by definition,

$$\mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\sigma) = \{w : \mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\sigma, w)\}$$

for every instructive subterm σ. Therefore Theorems 17 and 19 yield

$$\Gamma[\mathcal{C}_\ell^\omega, k](\tau) = \mathrm{Sem}_{\mathcal{C}_\ell^\omega}(\tau) = \{w : \mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\tau, w)\}.$$

Since the finality predicate accepts exactly `Found`, the term τ is accepted if and only if $\texttt{Found} \in \Gamma[\mathcal{C}_\ell^\omega, k](\tau)$, which is equivalent to $\mathsf{Interp}_{\mathcal{C}_\ell^\omega}(\tau, \texttt{Found})$, i.e. to $\mathcal{G}(\tau)$ containing a clique of size ℓ. □

D.5.6 Equivariance and coherence

Equivariance is as in the chromatic case (Section 9.2): relabelings act by renaming bag coordinates in ψ and preserve the flag `Found`, and the transition rules commute with this action. Hence, within the simple-graph regime fixed by Convention 3, Corollary 14 shows that the induced acceptance predicate is well-defined (independent of the chosen decomposition).

D.5.7 Bitlength, multiplicity, and running time

We now bound the encoding size and per-node multiplicity of clique witnesses.

Theorem 29 (Clique bitlength and multiplicity)**.** *Let B be a bag of size $m := |B| \leq k{+}1$. Then $\beta_{\mathrm{clq}}(k) = O((k{+}1)\log \ell)$ and $\mu_{\mathrm{clq}}(k) \leq (\ell{+}1)^{k+2}$. Hence*

$$T_{\mathrm{clq}}(k, n) = n \cdot k^{O(1)} \cdot (\log \ell)^{O(1)} \cdot \left((\ell+1)^{k+2}\right)^{O(1)}$$

by Corollary 2.

Proof. At a bag B of size $m \leq k+1$, the counter map $\psi : B \rightharpoonup \{0, \ldots, \ell-1\}$ can be

encoded as a total map $B \to \{\bot, 0, \dots, \ell - 1\}$, where $\bot$ denotes "undefined". Thus ψ requires $m \cdot O(\log(\ell + 1)) = O((k + 1) \log \ell)$ bits, and $t \in \{0, \dots, \ell\}$ needs $O(\log \ell)$ bits. Hence $\beta_{\mathrm{clq}}(k) = O((k + 1) \log \ell)$.

For multiplicity, there are $(\ell + 1)^m$ choices for ψ and $(\ell + 1)$ choices for t, so the number of pair witnesses (ψ, t) is at most $(\ell + 1)^{m+1} \leq (\ell + 1)^{k+2}$. The additional witness `Found` does not increase multiplicity, since cleaning collapses any set containing `Found` to the singleton $\{\texttt{Found}\}$. Hence $\mu_{\mathrm{clq}}(k) \leq (\ell + 1)^{k+2}$. By Lemma 35, Corollary 2 applies. Instantiating it with these β, μ bounds gives the displayed runtime. □

D.6 Multi-Edge Detection

This section instantiates the proof framework of Section 9.1 for the multi-edge detection core $\mathcal{C}^{\mathrm{ME}}$ from Section 4.8.2. Our goal is to show that acceptance coincides with the existence of a parallel edge in the realized multigraph, and to record the resulting witness parameters. Since cleaning is the identity, the main work is a constructor-wise verification of the local interpretation law, after which global correctness and the generic complexity bound follow immediately.

D.6.1 Witness space, cleaning, and acceptance

We recall the multi-edge DP-core $\mathcal{C}^{\mathrm{ME}}$ from Section 4.8.2. At a node σ with top bag $B(\sigma)$, a witness is either the flag `Found` or a set $F \subseteq \binom{B(\sigma)}{2}$ storing the unordered bag pairs $\{u, v\}$ such that $\mathcal{G}(\sigma)$ contains at least one edge between $\theta[\sigma](u)$ and $\theta[\sigma](v)$. Cleaning is the identity. Acceptance at the root is the presence of `Found`.

D.6.2 Transitions as set-valued functions

Transitions add a pair $\{u, v\}$ when an edge between the corresponding bag vertices is introduced, forget removes all pairs incident with the forgotten label, and promotion to `Found` occurs either when an edge is introduced twice within a subtree or when a join combines two subgraphs that both contain an edge on the same bag pair (creating a parallel edge in the pushout multigraph).

D.6.3 Denotation and the local interpretation predicate

For denotation we take $\mathrm{Sem}_{\mathcal{C}^{\mathrm{ME}}}$ to be the characteristic set of the catalog predicate $\mathsf{P}_k^{\mathrm{ME}}$: for $\tau \in \mathsf{ITD}_k$ and a witness w,

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{ME}}}(\tau, w) \iff \mathsf{P}_k^{\mathrm{ME}}(\tau, w), \qquad \mathrm{Sem}_{\mathcal{C}^{\mathrm{ME}}}(\tau) := \{w : \mathsf{Interp}_{\mathcal{C}^{\mathrm{ME}}}(\tau, w)\}.$$

Unfolding the definition of $\mathsf{P}_k^{\mathrm{ME}}$ shows that `Found` denotes "a parallel edge exists in $\mathcal{G}(\tau)$" and a set F denotes the set of bag pairs $\{u, v\}$ such that $\mathcal{G}(\tau)$ contains at least one edge between $\theta[\tau](u)$ and $\theta[\tau](v)$.

D.6.4 Local law (INT / LSC)

We verify the local interpretation law for the multi-edge detector.

Lemma 38 (Multi-edge INT)**.** *For each constructor a and children $(\sigma_i)_i$,*

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{ME}}}\big(a(\vec{\sigma}), w\big) \iff \exists \vec{w} \, \Big(\bigwedge_i \mathsf{Interp}_{\mathcal{C}^{\mathrm{ME}}}(\sigma_i, w_i) \ \wedge \ w \in \widehat{a}(\vec{w}) \Big).$$

Proof. We prove the equivalence by cases on a, writing $\tau := a(\vec{\sigma})$ and unfolding $\mathsf{P}_k^{\mathrm{ME}}$ as described above: either $w = \texttt{Found}$ and $\mathcal{G}(\tau)$ contains a multi-edge, or else $w = F \subseteq \binom{B(\tau)}{2}$ records exactly which bag pairs already carry at least one edge in $\mathcal{G}(\tau)$.

Leaf. $\mathcal{G}(\texttt{leaf})$ has no edges, hence no multi-edge. The unique set witness is $F = \emptyset$, and $\widehat{\texttt{leaf}}() = \{\emptyset\}$, so the equivalence holds.

Introduce vertex. Let $\tau = \texttt{intro\_}v(\sigma)$. Introducing an isolated vertex does not change the realized edge multiset, hence does not create or destroy multi-edges and does not change which pairs of bag vertices are adjacent by an edge. Therefore both `Found` and every set witness F are propagated unchanged, establishing both directions of (INT).

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ and write e for the introduced edge between $\theta[\tau](u)$ and $\theta[\tau](v)$. The parent graph is obtained from the child graph by adding one copy of e to the edge multiset. ($\Rightarrow$) If $w = \texttt{Found}$, then either a multi-edge already existed in $\mathcal{G}(\sigma)$ (so the child witness is `Found` and the transition propagates it), or else the new copy of e creates a second parallel edge on the same endpoints. In this second case, the child witness must be some F with $\{u, v\} \in F$, and the transition returns `Found` as required. If $w = F'$ is a set witness, then $\mathcal{G}(\tau)$ has no multi-edge and F' agrees with the child set witness F except that $\{u, v\}$ is present after introducing e, i.e. $F' = F \cup \{\{u, v\}\}$. This is exactly the non-flag branch of the introduce-edge transition. ($\Leftarrow$) Conversely, if $\mathsf{Interp}(\sigma, w_1)$ and $w \in \widehat{\texttt{intro\_}u\texttt{\_}v}(w_1)$, then: if $w_1 = \texttt{Found}$ the flag remains correct. If $w_1 = F$ and $\{u, v\} \in F$, the second copy of e creates a multi-edge and the transition returns `Found`. Otherwise the transition returns $F \cup \{\{u, v\}\}$, which is the correct updated bag-pair set in $\mathcal{G}(\tau)$.

Forget vertex. Let $\tau = \texttt{forget\_}v(\sigma)$ so $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$ and the parent bag is $B(\tau) = B(\sigma) \setminus \{v\}$. ($\Rightarrow$) If $w = \texttt{Found}$ then a multi-edge exists in $\mathcal{G}(\sigma)$ as well, so the child witness is `Found` and the transition propagates it. If $w = F$ is a set witness on $B(\tau)$, then the child witness is the corresponding set F' on $B(\sigma)$, and F is obtained by deleting

from F' all pairs incident to v. This is exactly the forget transition on set witnesses. ($\Leftarrow$) Conversely, the forget transition never changes the realized graph, so propagating `Found` preserves its meaning, and restricting F' by removing all pairs incident to v yields exactly the correct bag-pair set on the smaller bag $B(\tau)$.

Join. Let $\tau = \texttt{join}(\sigma_1, \sigma_2)$ be a join node with common bag B. By pushout semantics, $E(\mathcal{G}(\tau))$ is the multiset union of $E(\mathcal{G}(\sigma_1))$ and $E(\mathcal{G}(\sigma_2))$. ($\Rightarrow$) If $w = \texttt{Found}$, then either a multi-edge already exists in some child graph, in which case the corresponding child witness is `Found` and the join transition propagates it, or else the multi-edge in $\mathcal{G}(\tau)$ consists of one edge contributed by each child between the same bag endpoints. In the latter case, there exists a bag pair $\{u, v\}$ such that both child witnesses are sets F_1, F_2 with $\{u, v\} \in F_1 \cap F_2$, and the join transition returns `Found`. If $w = F$ is a set witness, then $\mathcal{G}(\tau)$ has no multi-edge, so neither child has a multi-edge and no bag pair can occur as an edge in both children. Hence the child witnesses are sets F_1, F_2 with $F_1 \cap F_2 = \emptyset$, and the correct parent set is $F = F_1 \cup F_2$, which is exactly the join transition output. ($\Leftarrow$) Conversely, if the join transition returns `Found`, then either some child already has a multi-edge, or there exists $\{u, v\} \in F_1 \cap F_2$, which yields two parallel edges between $\theta[\tau](u)$ and $\theta[\tau](v)$ in the multiset union. If it returns $F_1 \cup F_2$, then $F_1 \cap F_2 = \emptyset$ and both children are multi-edge free, so the pushout multigraph is also multi-edge free and the union set is exactly the set of bag pairs that carry an edge. □

Thus (INT) holds for $\mathcal{C}^{\mathrm{ME}}$. Since cleaning is the identity, (LSC) follows by Lemma 27, and Theorem 17 applies.

D.6.5 From local law to global correctness

We now derive global correctness and the witness-complexity bounds.

Theorem 30. $\Gamma[\mathcal{C}^{\mathrm{ME}}, k] = \mathrm{Sem}_{\mathcal{C}^{\mathrm{ME}}}$. *Acceptance holds iff a multi-edge exists.* $\beta = O(k^2)$, $\mu = 1$.

Proof. Lemma 38 establishes (INT) for every constructor. Since cleaning is the identity, (LSC) follows and Theorem 17 yields $\Gamma[\mathcal{C}^{\mathrm{ME}}, k] = \mathrm{Sem}_{\mathcal{C}^{\mathrm{ME}}}$. At the root, $\mathcal{C}^{\mathrm{ME}}$ accepts iff the flag `Found` is present, which by the definition of $\mathsf{P}_k^{\mathrm{ME}}$ is equivalent to the existence of a parallel edge in the realized multigraph.

For encoding, let B be a bag of size $m := |B| \leq k{+}1$. A set $F \subseteq \binom{B}{2}$ can be represented as a bitvector of length $\binom{m}{2}$, so $\beta = O(\binom{m}{2}) = O(k^2)$. For multiplicity, every denotational state is a singleton: if $\mathcal{G}(\sigma)$ contains a multi-edge then $\mathrm{Sem}_{\mathcal{C}^{\mathrm{ME}}}(\sigma) = \{\texttt{Found}\}$, while otherwise

$$\mathrm{Sem}_{\mathcal{C}^{\mathrm{ME}}}(\sigma) = \Big\{\{\{u, v\} \subseteq B(\sigma) : \mathcal{G}(\sigma) \text{ contains an edge between } \theta[\sigma](u) \text{ and } \theta[\sigma](v)\}\Big\}.$$

Hence every useful set has size 1, so $\mu = 1$. □

D.6.6 Equivariance and coherence

Equivariance is immediate: relabelings act by permuting the elements of $\binom{B}{2}$ (and fix the flag), and all transitions commute with this action. Coherence follows by Corollary 14.

D.6.7 Bitlength, multiplicity, and running time

The bounds $\beta = O(k^2)$ and $\mu = 1$ are recorded in Theorem 30. By Lemma 35, Corollary 2 applies, so the generic runtime bound follows by instantiating it.

D.7 Perfect Matching

We now instantiate the framework of Section 9.1 for the perfect-matching core $\mathcal{C}^{\mathrm{PM}}$ from Section 4.8.2.

D.7.1 Witness space, cleaning, and acceptance

At a bag $B \subseteq [k{+}1]$ a witness is a subset $U \subseteq B$, interpreted as the set of vertices in the bag that are still *unmatched*, with the invariant that every vertex outside the bag is already matched once. Cleaning is the identity, and the finality predicate accepts exactly the empty set $U = \emptyset$, expressing that every vertex of the realized graph is matched.

D.7.2 Transitions as set-valued functions

We restate the transition rules here (with the catalog anchor at Section 4.8.2). Writing B for the current bag:

$$\mathcal{C}^{\mathrm{PM}}[k].\texttt{LeafSet} = \{\emptyset\},$$

$$\mathcal{C}^{\mathrm{PM}}[k].\texttt{intro\_}u(U) = \{U \cup \{u\}\},$$

$$\mathcal{C}^{\mathrm{PM}}[k].\texttt{forget\_}u(U) = \begin{cases} \{U\}, & u \notin U, \\ \emptyset, & u \in U, \end{cases}$$

$$\mathcal{C}^{\mathrm{PM}}[k].\texttt{intro\_}u\texttt{\_}v(U) = \begin{cases} \{U,\ U \setminus \{u, v\}\}, & u, v \in U, \\ \{U\}, & \text{otherwise}, \end{cases}$$

$$\mathcal{C}^{\mathrm{PM}}[k].\texttt{join}(U_1, U_2) = \begin{cases} \{U_1 \cap U_2\}, & U_1 \cup U_2 = B, \\ \emptyset, & \text{otherwise}. \end{cases}$$

D.7.3 Denotation and the local interpretation predicate

For a graph G, let

$$X_{\mathrm{PM}}(G) := \{\, M \subseteq E(G) : M \text{ is a matching in } G \,\}.$$

For a term σ we will use the *saturated* matchings

$$X^{\mathrm{sat}}_{\mathrm{PM}}(\sigma) := \{\, M \in X_{\mathrm{PM}}(\mathcal{G}(\sigma)) : \forall v \in V(\mathcal{G}(\sigma)) \setminus \theta[\sigma](B(\sigma)) :\ \deg_M(v) = 1 \,\},$$

i.e. matchings in which every non-bag vertex is already matched once. For a term σ with realized graph $\mathcal{G}(\sigma)$, top bag $B(\sigma)$, and labeling $\theta[\sigma] : B(\sigma) \to V(\mathcal{G}(\sigma))$, define the bag projection

$$\pi_{B(\sigma)}(M) := \{u \in B(\sigma) : \deg_M(\theta[\sigma](u)) = 0\}.$$

Thus $\pi_{B(\sigma)}(M)$ is precisely the set of bag vertices that remain unmatched in the matching M.

For $\sigma \in \mathsf{ITD}_k$ and $U \subseteq [k{+}1]$ set

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}(\sigma, U) \iff \mathsf{P}^{\mathrm{PM}}_k(\sigma, U), \qquad \mathrm{Sem}_{\mathcal{C}^{\mathrm{PM}}}(\sigma) := \{U : \mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}(\sigma, U)\},$$

where $\mathsf{P}^{\mathrm{PM}}_k$ is the perfect-matching catalog predicate from Chapter 4. Unfolding the definition, $\mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}(\sigma, U)$ asserts that there exists a matching $M \in X_{\mathrm{PM}}(\mathcal{G}(\sigma))$ such that

$$\forall u \in B(\sigma) :\ \theta[\sigma](u) \text{ is } \textit{unmatched} \text{ in } M \iff u \in U,$$

and all non-bag vertices are fully matched in M, i.e. $M \in X^{\mathrm{sat}}_{\mathrm{PM}}(\sigma)$.

D.7.4 Local law (INT / LSC)

We begin with the join-gluing lemma needed for perfect matching and then state the full interpretation law.

Lemma 39 (PM join gluing)**.** *Let $\tau = \texttt{join}(\tau_1, \tau_2)$ be a join node with common bag B. For saturated matchings, $(X^{\mathrm{sat}}_{\mathrm{PM}}, \pi_B)$ satisfies Definition 57 with*

$$\mathrm{Comp}_B(U_1, U_2) \iff U_1 \cup U_2 = B, \qquad U_1 \otimes_B U_2 := \{U_1 \cap U_2\}.$$

Proof. Let $M \in X^{\mathrm{sat}}_{\mathrm{PM}}(\tau)$ and let $M_i := M \cap E(\mathcal{G}(\tau_i))$ be the edge-set restrictions. Then each M_i is a matching in $\mathcal{G}(\tau_i)$, and since $V(\mathcal{G}(\tau)) \setminus \theta[\tau](B)$ is the disjoint union of $V(\mathcal{G}(\tau_i)) \setminus \theta[\tau_i](B)$, each M_i is saturated on non-bag vertices, i.e. $M_i \in X^{\mathrm{sat}}_{\mathrm{PM}}(\tau_i)$. Let $U_i := \pi_B(M_i)$. If a bag vertex b is matched in M_i, then it cannot be matched in the other child (otherwise it would have degree 2 in M), so for every $b \in B$ at least one of

M_1, M_2 leaves b unmatched, i.e. $U_1 \cup U_2 = B$. Moreover, a bag vertex is unmatched in $M = M_1 \cup M_2$ iff it is unmatched in both M_1 and M_2, so $\pi_B(M) = U_1 \cap U_2$.

Conversely, let $M_i \in X^{\mathrm{sat}}_{\mathrm{PM}}(\tau_i)$ and write $U_i := \pi_B(M_i)$. If $U_1 \cup U_2 = B$, then no bag vertex is matched in both M_1 and M_2, so $M := M_1 \cup M_2$ is a matching in the pushout graph. Every non-bag vertex lies in exactly one child and is matched there, hence $M \in X^{\mathrm{sat}}_{\mathrm{PM}}(\tau)$. Finally, $\pi_B(M) = U_1 \cap U_2$, which is exactly the witness-level composition above. □

Lemma 40 (PM interpretation law)**.** *For each constructor a and children $(\sigma_i)_i$,*

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}\big(a(\vec{\sigma}), U\big) \iff \exists \vec{U} \, \Big(\bigwedge_i \mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}(\sigma_i, U_i) \ \wedge \ U \in \widehat{a}(\vec{U}) \Big).$$

Proof. We verify the equivalence by cases on the constructor a.

Leaf. The realized graph is empty and the only matching is $M = \emptyset$, whose projection is $U = \emptyset$. This matches the leaf transition $\widehat{\texttt{leaf}}() = \{\emptyset\}$.

Introduce vertex. Let $\tau = \texttt{intro\_}v(\sigma)$ and write $x := \theta[\tau](v)$ for the new vertex. Since x is isolated in $\mathcal{G}(\tau)$, every matching M in $\mathcal{G}(\tau)$ has $\deg_M(x) = 0$, so $v \in \pi_{B(\tau)}(M)$. Restricting M to $\mathcal{G}(\sigma)$ yields a matching witnessing the child predicate, and conversely any child matching extends to $\mathcal{G}(\tau)$ by leaving x unmatched. This is exactly the witness update $U \mapsto U \cup \{v\}$ of the introduce transition.

Forget vertex. Let $\tau = \texttt{forget\_}v(\sigma)$. Since $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$, a matching M witnesses $\mathsf{Interp}(\tau, U)$ iff it witnesses $\mathsf{Interp}(\sigma, U')$ for $U' := \pi_{B(\sigma)}(M)$ and the forgotten vertex $x := \theta[\sigma](v)$ is matched in M, i.e. $v \notin U'$. The forget transition discards witnesses with $v \in U'$ and otherwise restricts U' to $B(\tau)$, so (INT) holds.

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ and write $e := \{\theta[\tau](u), \theta[\tau](v)\}$ for the new edge. If a matching M witnessing $\mathsf{Interp}(\tau, U)$ does not use e, then M is a matching of $\mathcal{G}(\sigma)$ with the same projection. If it uses e, then removing e yields a matching of $\mathcal{G}(\sigma)$ in which both endpoints are unmatched. Conversely, whenever both endpoints are unmatched in a child matching one may add e to match them. This is exactly the skip/use branching of $\widehat{\texttt{intro\_}u\texttt{\_}v}$.

Join. This is an instance of Theorem 18 (gluing saturated matchings along the pushout) instantiated with $X^{\mathrm{sat}}_{\mathrm{PM}}$ and π_B, where $\mathrm{Comp}_B(U_1, U_2) \iff U_1 \cup U_2 = B$ and $U_1 \otimes_B U_2 := \{U_1 \cap U_2\}$. □

D.7.5 From local law to global correctness

Hence $\mathcal{C}^{\mathrm{PM}}$ satisfies (LSC), and Theorem 17 yields

$$\Gamma[\mathcal{C}^{\mathrm{PM}}, k](\sigma) = \mathrm{Sem}_{\mathcal{C}^{\mathrm{PM}}}(\sigma) = \{U : \mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}(\sigma, U)\}.$$

By construction of $\mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}$ this equality is equivalent to the perfect-matching catalog statement in Section 4.8.2. Since the finality predicate accepts exactly $U = \emptyset$, a term σ is accepted iff $\emptyset \in \Gamma[\mathcal{C}^{\mathrm{PM}}, k](\sigma)$. Unfolding $\mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}$, this holds iff there exists a saturated matching $M \in X^{\mathrm{sat}}_{\mathrm{PM}}(\sigma)$ with $\pi_{B(\sigma)}(M) = \emptyset$, i.e. a matching in $\mathcal{G}(\sigma)$ that matches every vertex. Equivalently, $\mathcal{G}(\sigma)$ has a perfect matching.

Theorem 31 (Perfect matching correctness and complexity)**.** *This is the chapter-level correctness anchor for the perfect-matching core referenced from Section 4.8.2. For every k and term $\sigma \in \mathsf{ITD}_k$,*

$$\Gamma[\mathcal{C}^{\mathrm{PM}}, k](\sigma) = \mathrm{Sem}_{\mathcal{C}^{\mathrm{PM}}}(\sigma),$$

and σ is accepted by $\mathcal{C}^{\mathrm{PM}}[k]$ if and only if $\mathcal{G}(\sigma)$ has a perfect matching. Moreover, at a bag of size $m \le k{+}1$ witnesses are subsets of B, so

$$\beta_{\mathrm{PM}}(k) = O(k), \qquad \mu_{\mathrm{PM}}(k) \le 2^{k+1},$$

and thus

$$T_{\mathrm{PM}}(k, n) = n \cdot k^{O(1)} \cdot 2^{O(k)}$$

by Corollary 2.

Proof. Lemma 40 establishes (INT) for every constructor of $\mathcal{C}^{\mathrm{PM}}$. Since cleaning is the identity, (LSC) follows. Thus Theorem 17 yields $\Gamma[\mathcal{C}^{\mathrm{PM}}, k] = \mathrm{Sem}_{\mathcal{C}^{\mathrm{PM}}}$. Since the finality predicate accepts exactly $U = \emptyset$, σ is accepted iff $\emptyset \in \Gamma[\mathcal{C}^{\mathrm{PM}}, k](\sigma)$, which by the definition of $\mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}$ holds exactly when $\mathcal{G}(\sigma)$ has a perfect matching.

For complexity, a witness is a subset $U \subseteq B$ with $|B| \le k + 1$, so it can be encoded as a bitmask of length $|B|$, giving $\beta_{\mathrm{PM}}(k) = O(k)$. There are at most $2^{|B|} \le 2^{k+1}$ such subsets, and cleaning does not merge distinct subsets, so $\mu_{\mathrm{PM}}(k) \le 2^{k+1}$. By Lemma 35, Corollary 2 applies. The runtime bound follows. □

Lemma 41 (PerfectMatching implementation refines $\mathcal{C}^{\mathrm{PM}}$)**.** *The* TreeWidzard *DP-core* `PerfectMatching` *uses witnesses consisting of a partial matching on the current bag (encoded as a set of disjoint pairs) together with a set of bag vertices already matched to forgotten vertices. Under the projection*

$$U = \{\, b \in B : b \text{ is not incident to a stored pair and } b \notin \mathrm{sat} \,\},$$

the set of refined witnesses realized at a bag projects exactly to the unmatched-set witness semantics of $\mathcal{C}^{\mathrm{PM}}$ from Section 4.8.2. In particular, the implementation's bag-dependent finality test (`can_clear_bag`) is equivalent to requiring $U = \emptyset$, so the implementation accepts exactly the graphs with a perfect matching.

Proof. Fix a current term σ and write $B := B(\sigma)$. A vertex label $b \in B$ is in one of three disjoint states: *free* (unmatched), *paired* with another bag label via the stored partial matching, or *satisfied* (already matched to a forgotten vertex). The set U is exactly the set of free labels.

Conversely, let U be a canonical witness with $\mathsf{Interp}_{\mathcal{C}^{\mathrm{PM}}}(\sigma, U)$, witnessed by a saturated matching $M \in X^{\mathrm{sat}}_{\mathrm{PM}}(\sigma)$. Define a refined state by

$$\begin{aligned} P_M &:= \big\{\{b, c\} \subseteq B : \{\theta[\sigma](b), \theta[\sigma](c)\} \in M\big\}, \\ \mathrm{sat}_M &:= \big\{b \in B : \theta[\sigma](b) \text{ is matched by } M \\ &\qquad\qquad \text{to a vertex outside } \theta[\sigma](B)\big\}. \end{aligned}$$

Because M is a matching, the pairs in P_M are disjoint and no label appearing in a pair from P_M belongs to sat_M. Moreover, every bag vertex is either free, paired inside the bag, or matched to a forgotten vertex, so the projection of (P_M, sat_M) is exactly U. Hence every realized canonical witness has a refined representative.

Introduce-vertex does not add edges, so the refined witness does not change and U gains the new label, matching the canonical transition $\mathcal{C}^{\mathrm{PM}}[k].\texttt{intro\_}u$. Introduce-edge may either skip the edge or use it to pair two free bag vertices, which removes both from U, matching $\mathcal{C}^{\mathrm{PM}}[k].\texttt{intro\_}u\texttt{\_}v$. Forgetting a label is allowed only when it is not free. If it is satisfied, it is removed. If it is paired within the bag, the pair is removed and the partner becomes satisfied. In all cases, the projected set U evolves exactly as in the canonical forget transition (rejecting precisely when the forgotten label is unmatched). Finally, at joins the implementation unions the two child matchings and satisfied sets subject to the usual matching compatibility (no bag vertex matched twice). This projects to the canonical guard $U_1 \cup U_2 = B$ and composition $U_1 \cap U_2$.

The finality test `can_clear_bag` succeeds exactly when every remaining bag vertex is not free (either satisfied or paired inside the bag), i.e. when $U = \emptyset$. Therefore the implementation accepts exactly when $\mathcal{C}^{\mathrm{PM}}$ accepts, which by Theorem 31 holds exactly when the realized graph has a perfect matching. □

D.7.6 Equivariance and coherence

Equivariance is immediate: relabelings act by renaming bag elements in U, and the transitions (which only use set operations and the join guard $U_1 \cup U_2 = B$) commute with this action. Coherence follows by Corollary 14.

D.7.7 Bitlength, multiplicity, and running time

The complexity bounds are recorded in Theorem 31: $\beta_{\mathrm{PM}}(k) = O(k)$ and $\mu_{\mathrm{PM}}(k) \le 2^{k+1}$. By Lemma 35, Corollary 2 applies, yielding $T_{\mathrm{PM}}(k, n) = n \cdot k^{O(1)} \cdot 2^{O(k)}$.

D.8 Hamiltonian Path: Endpoint Matchings (Involutions)

This section discharges the correctness proof for the Hamiltonian-path core $\mathcal{C}^{\mathrm{HP}}$ described in Chapter 4 (Section 4.8.2). We spell out the DP-core in the framework of Section 9.1, at the same level of detail as the vertex-cover example.

D.8.1 Witness space, cleaning, and acceptance

Fix $k \in \mathbb{N}$ and a bag $B \subseteq [k{+}1]$.

Witness space. At B a witness is a triple (d, M, e) where

- $d \in \{0,1,2\}^B$ is a degree map.
- M is a partial fixed-point-free involution $M : E_B \rightharpoonup E_B$ (equivalently, a matching) on $E_B := \{v \in B : d(v) = 1\}$.
- $e \in \{0,1,2,\dots\}$ counts path endpoints that have been forgotten so far; a forgotten isolated vertex contributes two coincident endpoints.

Intuitively, d records degrees in the chosen path subgraph, M pairs those degree-1 vertices whose two endpoints both lie in the bag, and e accounts for forgotten endpoints, including the two coincident endpoints of a forgotten isolated vertex. We write $W_k^{\mathrm{HP}}(B)$ for the set of such triples. The ambient witness space is

$$W_k^{\mathrm{HP}} := \bigcup_{B \subseteq [k+1]} W_k^{\mathrm{HP}}(B).$$

The implementation keeps two additional bookkeeping fields: a cycle counter cc (any state with $\mathrm{cc} > 0$ is discarded) and the number comps of open components intersecting the bag, obtained as

$$\mathrm{comps} = \frac{|\operatorname{dom}(M)|}{2} + \big|E_B \setminus \operatorname{dom}(M)\big|.$$

Since M is fixed-point-free, $|\operatorname{dom}(M)|$ is even. These fields are used only to implement the transition relation efficiently and are not part of the logical witness type.

Cleaning. Cleaning $\texttt{Clean}_k$ removes duplicate witnesses and any state whose (d, M) fails to encode a partial matching on degree-1 vertices. On reachable states it is idempotent and we will not rely on extensivity.

Acceptance. At the root bag B we accept (d, M, e) iff either

(i) $B = \{v\}$, $d(v) = 0$, $M = \varnothing$, and $e = 0$ (the degenerate K_1 case), or

(ii) $d(v) \in \{1, 2\}$ for all $v \in B$ and $e + |E_B(d)| = 2$.

That is, every remaining bag vertex lies on the chosen subgraph, and the total number of endpoints is 2 (counting forgotten endpoints via e and bag endpoints via E_B). We treat the degenerate graph K_1 as a length-0 path with two coincident endpoints; this is why forgetting an isolated vertex contributes two endpoints, while (i) accepts the corresponding open-term root state.

D.8.2 Transitions as set-valued functions

Let B be the current bag. We write $E_B(d) := \{v \in B : d(v) = 1\}$.

Leaf. At a leaf the graph is empty and the only path subgraph is empty, so

$$\mathcal{C}^{\mathrm{HP}}[k].\texttt{LeafSet} = \{(\emptyset, \emptyset, 0)\}.$$

Introduce vertex. Let $\tau = \texttt{intro\_}v(\sigma)$. We have $\mathcal{G}(\tau)$ and $\mathcal{G}(\sigma)$ with the same edges. Introducing v extends the degree map by $v \mapsto 0$:

$$\mathcal{C}^{\mathrm{HP}}[k].\texttt{intro\_}v(d, M, e) = \{(d[v \mapsto 0], M, e)\}.$$

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ with $u, v \in B$. For $(d, M, e) \in W_k^{\mathrm{HP}}(B)$ define

$$\mathcal{C}^{\mathrm{HP}}[k].\texttt{intro\_}u\texttt{\_}v(d, M, e) = S_{\mathrm{skip}} \cup S_{\mathrm{use}}$$

where $S_{\mathrm{skip}} := \{(d, M, e)\}$ and S_{use} is defined as follows. Set $d_u := d(u)$, $d_v := d(v)$. If $d_u \geq 2$ or $d_v \geq 2$ then $S_{\mathrm{use}} := \varnothing$. Otherwise let $d'_u := d_u + 1$, $d'_v := d_v + 1$ and d' be d updated at u, v.

The matching M' is obtained by the standard path-concatenation rule:

- if $d_u = d_v = 0$ then u, v become endpoints of a new path component, so we add the pair (u, v) to M.
- if $d_u = 1, d_v = 0$, then u is an endpoint of a component and v is new. If u is matched to p in M, remove (p, u) and insert (p, v). Otherwise u was already unmatched and we leave M unchanged (the unmatched endpoint moves from u to v).
- the case $d_u = 0, d_v = 1$ is symmetric.

- if $d_u = d_v = 1$ then both are endpoints of components. If u and v are matched to each other, adding uv would form a cycle, so $S_{\text{use}} := \varnothing$. Otherwise, if u is matched to p and v to q with $p \neq v$, $q \neq u$, remove both pairs $(p, u), (q, v)$ and insert (p, q). If exactly one of u, v is matched, we remove that pair, leaving the other endpoint unmatched. If neither is matched, both endpoints were already external and M stays unchanged.

In all admissible cases we set $S_{\text{use}} := \{(d', M', e)\}$.

Forget vertex. Let $\tau = \texttt{forget\_}v(\sigma)$. For $(d, M, e) \in W_k^{\text{HP}}(B)$:

- If $d(v) = 0$, then v is an isolated length-0 path component. Forgetting it removes v from the bag and adds its two coincident endpoints to the forgotten endpoint counter: $\mathcal{C}^{\text{HP}}[k].\texttt{forget\_}v(d, M, e) = \{(d|_{B\setminus\{v\}}, M, e{+}2)\}$.
- If $d(v) = 1$, then v is an endpoint. We remove v from d, delete any pair involving v from M, and increment e: $\mathcal{C}^{\text{HP}}[k].\texttt{forget\_}v(d, M, e) = \{(d|_{B\setminus\{v\}}, M', e{+}1)\}$.
- If $d(v) = 2$, then v is internal. We drop v from d and leave M, e unchanged: $\mathcal{C}^{\text{HP}}[k].\texttt{forget\_}v(d, M, e) = \{(d|_{B\setminus\{v\}}, M, e)\}$.

Join. Let $\tau = \texttt{join}(\tau_1, \tau_2)$ with common bag B. For witnesses (d_i, M_i, e_i) on the children, we combine the child degree profiles pointwise and require that the resulting bag degrees stay at most 2; explicitly, we put

$$d(v) := d_1(v) + d_2(v)$$

for $v \in B$, rejecting if some $d(v) > 2$. The endpoint counter is $e := e_1 + e_2$. To define M we proceed as follows.

The guiding picture is that each child witness records how a collection of path components meets the common bag: a matched pair in M_i means that the same child component has two bag endpoints, while an unmatched degree-1 bag vertex means that the component leaves the bag through only one endpoint. The join operation therefore has to answer a purely interface-level question: after gluing the two children along B, which of these child path fragments concatenate through the bag into larger path fragments, and when does such a concatenation accidentally create a cycle or a branch?

Consider the auxiliary graph whose vertices are

- all $v \in B$ with $d_1(v) = 1$ or $d_2(v) = 1$, and
- for each side $i \in \{1, 2\}$ and each $v \in B$ with $d_i(v) = 1$ that is unmatched in M_i, a "stub" vertex $\star_{i,v}$.

Input: child witnesses (d_1, M_1, e_1) and (d_2, M_2, e_2) on B
Output: $\{(d, M, e)\}$ or $\varnothing$
$d \leftarrow d_1 + d_2$ and $e \leftarrow e_1 + e_2$
if $\exists v \in B$ *with* $d(v) > 2$ **then**
  reject
end
Build the auxiliary graph on bag vertices with $d_1(v) = 1$ or $d_2(v) = 1$ and stubs $\star_{i,v}$ induced by M_1, M_2
while *there exists a bag vertex* v *with* $d_1(v) = d_2(v) = 1$ **do**
  if v *has* $\neq 2$ *incident auxiliary edges* **then**
    reject
  end
  Splice the two incident edges through v into a single auxiliary edge and delete v
end
if *some auxiliary component has no endpoint, or a bag endpoint has degree* > 1 **then**
  reject
end
For each auxiliary component, pair its two bag endpoints (if any) to define M. Endpoints that are stubs remain unmatched
return $\{(d, M, e)\}$

Algorithm 4: HP join transition at a common bag B (summary)

We add an edge between two bag vertices u, v whenever M_i pairs them, and we add an edge between v and $\star_{i,v}$ whenever v is an unmatched endpoint in child i. This auxiliary graph encodes how child components touch the bag.

We then eliminate every bag vertex v with $d_1(v) = d_2(v) = 1$ in this auxiliary graph: v must have exactly two incident auxiliary edges, and we splice them together, replacing them by a single auxiliary edge between their other endpoints and deleting v. If such a vertex has more than two incident auxiliary edges, we reject (this would force branching). After all such eliminations, each connected component of the auxiliary graph is either a path (with endpoints stubs or degree-1 bag vertices) or a cycle (no endpoint).

We define M by pairing two degree-1 bag vertices u, v iff they are the two bag endpoints of the same auxiliary component. If an auxiliary path has one or two stub endpoints, the bag endpoints remain unmatched. If a degree-1 bag vertex has more than one real neighbor in the auxiliary graph, we reject. Finally we reject if some auxiliary component has no endpoint (i.e., forms a cycle after splicing).

The resulting matching M is a partial fixed-point-free involution on $E_B(d)$. We set $\mathcal{C}^{\mathrm{HP}}[k].\texttt{join}\big((d_1, M_1, e_1), (d_2, M_2, e_2)\big) = \{(d, M, e)\}$ when the construction succeeds, and $\varnothing$ otherwise.

Algorithm 4 summarizes the join construction.

D.8.2.1 Endpoint diagrams as a partial monoid

Fix a bag B. For a witness $w = (d, M, e) \in W_k^{HP}(B)$ define its *endpoint diagram* $\texttt{Diag}(w)$ to be the finite graph constructed as follows:

- the *bag-endpoint vertices* are $E_B(d) = \{v \in B : d(v) = 1\}$.
- for each $v \in E_B(d)$ that is *unmatched* in M, add a fresh *stub* vertex $\star_v$ and an edge $\{v, \star_v\}$.
- for each matched pair $\{u, v\}$ in M, add an edge $\{u, v\}$.

Thus every connected component of $\texttt{Diag}(w)$ is a single edge or a length-2 path $u - \star_u$. It records, abstractly, how the chosen path-subgraph components touch the bag. The important point for joins is that $\texttt{Diag}(w)$ forgets everything about the interior of the child graph except this boundary behavior: two child path components that induce the same diagram are interchangeable for the join construction.

Composition. Given two HP witnesses $w_i = (d_i, M_i, e_i)$ on the same bag B, define $w_1 \otimes_B w_2$ (if possible) by the following *diagrammatic* procedure:

1. Set $d := d_1 + d_2$ pointwise on B and reject if $d(v) > 2$ for some v. Set $e := e_1 + e_2$.
2. Form the disjoint union $\texttt{Diag}(w_1) \uplus \texttt{Diag}(w_2)$, keeping stubs distinct.
3. For every bag vertex $v \in B$ with $d_1(v) = d_2(v) = 1$ (so $d(v) = 2$), perform a *splice elimination*: v must have exactly two incident diagram edges. Delete v and replace those two edges by one edge between their other endpoints. If v has $\neq 2$ incident edges, reject.
4. After all splice eliminations, reject if any resulting connected component is a cycle (i.e. has no endpoint). Otherwise, define M by pairing two bag vertices $u, v \in B$ iff they are the two bag-endpoints of the same remaining component. All other bag vertices in $E_B(d)$ remain unmatched.

If no rejection occurs, set $w_1 \otimes_B w_2 := (d, M, e)$.

Equivalently, $\otimes_B$ first superimposes the two child interface diagrams on the common bag, then suppresses precisely those bag vertices that become internal after gluing (the degree pattern $(1, 1)$ case), and finally reads off the surviving path endpoints from the reduced diagram. This is why the same construction simultaneously defines the join transition and the semantic gluing rule.

Lemma 42 (Associativity and well-definedness)**.** *The operation $\otimes_B$ is well-defined (independent of the order of splice eliminations), and is associative on all triples for which both sides are defined:*

$$(w_1 \otimes_B w_2) \otimes_B w_3 \;=\; w_1 \otimes_B (w_2 \otimes_B w_3).$$

Moreover, for any join node $\tau = \texttt{join}(\tau_1, \tau_2)$ and any path subgraphs $H_i \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau_i))$, if $H := H_1 \cup H_2$ is a path subgraph in $\mathcal{G}(\tau)$ then

$$\pi_B(H) \;=\; \pi_B(H_1) \otimes_B \pi_B(H_2),$$

and if H is not a path subgraph then the diagram composition rejects.

Proof. Fix $w_1 = (d_1, M_1, e_1)$ and $w_2 = (d_2, M_2, e_2)$ on bag B and form the union diagram $D := \texttt{Diag}(w_1) \uplus \texttt{Diag}(w_2)$ by identifying bag vertices and keeping stubs distinct. Let $C := \{v \in B : d_1(v) = d_2(v) = 1\}$. In D, every $v \in C$ has degree exactly 2 (one incident diagram edge from each side), so splice elimination at v is the standard *suppression* of a degree-2 vertex (delete v and connect its two neighbors). The result of suppressing all vertices in C is independent of the order: it can be characterized as the graph on vertex set $V(D) \setminus C$ in which two vertices x, y are adjacent iff they are connected by a D-path whose internal vertices lie in C (i.e. a maximal C-path is replaced by a single edge between its endpoints). Hence $\otimes_B$ is well-defined.

For associativity, let w_1, w_2, w_3 be witnesses on B such that both $(w_1 \otimes_B w_2) \otimes_B w_3$ and $w_1 \otimes_B (w_2 \otimes_B w_3)$ are defined. Form the three-way union diagram $D := \texttt{Diag}(w_1) \uplus \texttt{Diag}(w_2) \uplus \texttt{Diag}(w_3)$ (bag vertices are identified and stubs remain distinct). Since composition is defined, for every $v \in B$ we have $d_1(v) + d_2(v) + d_3(v) \le 2$, so v is a bag-endpoint vertex in at most two of the diagrams. Let C be the set of bag vertices that appear as endpoints in *exactly two* of the three witnesses. Equivalently, those bag vertices of degree 2 in the union diagram D. In either parenthesization, the splice eliminations delete precisely the vertices in C (possibly in two stages), and by the well-definedness argument the resulting reduced diagram depends only on (D, C). Thus both parenthesizations produce the same reduced diagram and hence the same pairing M of surviving bag endpoints. Since the numeric fields d and e are combined by pointwise addition and integer addition, respectively, the resulting witness is the same and $\otimes_B$ is associative where defined.

For the semantic clause, let $\tau = \texttt{join}(\tau_1, \tau_2)$ and let $H_i \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau_i))$. Write $\pi_B(H_i) = (d_i, M_i, e_i)$ and let $H := H_1 \cup H_2 \subseteq \mathcal{G}(\tau)$ be the glued subgraph in the pushout graph. Then $d = d_1 + d_2$ records bag degrees in H, and $e = e_1 + e_2$ counts endpoints of H outside the bag (endpoints outside the bag remain outside after gluing). The endpoint diagram

$\texttt{Diag}(\pi_B(H_i))$ records exactly how the path components of H_i meet the bag: matched pairs in M_i correspond to components with two bag endpoints, and stubs correspond to components with one bag endpoint. If $d_1(v) = d_2(v) = 1$, then v is an endpoint in both H_1 and H_2 and becomes internal (degree 2) in H. Splicing v in the union diagram is exactly concatenation of the two incident path pieces through v. After splicing all such v, the reduced diagram can be read component-by-component:

- a component with two bag endpoints represents a path component of H whose two endpoints remain in the bag, so those two bag labels must be paired in M;
- a component with one bag endpoint and one stub represents a path component of H with one endpoint in the bag and one endpoint outside the bag, so the bag endpoint remains unmatched and the outside endpoint contributes to e;
- a component with two stubs represents a path component of H whose two endpoints are both outside the bag, so it contributes no pair to M and contributes two external endpoints to e;
- a component with no endpoint is a cycle, which is forbidden in HP and therefore forces rejection.

In other words, every connected component of the reduced diagram is exactly the bag trace of one connected component of H, and the reduced diagram contains all information needed to reconstruct $\pi_B(H)$. Therefore $\pi_B(H) = \pi_B(H_1) \otimes_B \pi_B(H_2)$ when H is a path subgraph, and the diagram composition rejects exactly when H is not. □

D.8.3 Denotation and the local interpretation predicate

We define the global object family for Hamiltonian path and the corresponding bag projection.

Global objects. Let σ be a term with bag $B(\sigma)$. We let $\mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$ be the set of spanning selected-edge subgraphs $H \subseteq \mathcal{G}(\sigma)$ whose connected components are finite simple paths, where an isolated vertex is regarded as a length-0 path. Thus every vertex outside the bag is already committed to one of these path components; if such a forgotten component is isolated, it contributes two coincident endpoints to the endpoint counter below.

Projection. For a term σ with bag $B(\sigma)$ and labeling $\theta[\sigma]$ define $\pi_{B(\sigma)} : \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma)) \to W_k^{\mathrm{HP}}(B(\sigma))$ as follows. For $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$:

- $d(v) := \deg_H(\theta[\sigma](v))$ for $v \in B(\sigma)$.

- M pairs distinct $u, v \in B(\sigma)$ exactly when H has a non-degenerate path component whose two endpoints both lie in $\theta[\sigma](B(\sigma))$ and are $\theta[\sigma](u)$ and $\theta[\sigma](v)$.
- e is the number of endpoints of H that do not lie in $\theta[\sigma](B(\sigma))$, with the convention that a length-0 path contributes two coincident endpoints at its unique vertex.

Denotation and interpretation. Set

$$\mathrm{Sem}_{\mathcal{C}^{\mathrm{HP}}}(\sigma) := \mathtt{Clean}_k\big(\{\pi_{B(\sigma)}(H) : H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))\}\big)$$

and

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma, (d, M, e)) \iff \exists H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma)) : \ \pi_{B(\sigma)}(H) = (d, M, e).$$

Lemma 43 (Well-formed projections for HP)**.** *Let σ be a term with bag $B := B(\sigma)$ and let $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$. Write $\pi_{B(\sigma)}(H) = (d, M, e)$. Then (d, M, e) is a well-formed HP witness:*

1. *$d : B \to \{0, 1, 2\}$ (i.e. all bag degrees in H are at most 2).*
2. *M is a partial fixed-point-free involution on $E_B(d)$, pairing exactly those degree-1 bag labels that are endpoints of the same path component of H.*
3. *e equals the number of endpoints of H outside $\theta[\sigma](B)$, counting an isolated outside vertex twice.*

In particular, every realized projection lies in the witness universe $W_k^{\mathrm{HP}}(B)$, so the cleaning operator $\mathtt{Clean}_k$ is the identity on the set of realized projections.

Proof. Since H is a disjoint union of simple paths (possibly including the degenerate length-0 path), every vertex of H has degree at most 2, hence $d(v) = \deg_H(\theta[\sigma](v)) \in \{0, 1, 2\}$ for each $v \in B$.

Consider the path components of H. Every non-degenerate path component has exactly two endpoints, whereas a length-0 path has degree 0 at its unique vertex and therefore contributes no degree-1 bag label. If that unique vertex is outside the bag, its two coincident endpoints are counted in e; if it is in the bag, they remain uncounted until the vertex is forgotten. Intersecting a non-degenerate path component with $\theta[\sigma](B)$ yields one of three cases: no bag endpoint, exactly one bag endpoint, or two distinct bag endpoints. In the last case, the two bag endpoints are uniquely determined and define a unique pair $\{u, v\}$, which we put into M. This produces a partial matching on the degree-1 bag labels, hence a partial fixed-point-free involution on $E_B(d)$. The value e is, by definition, the number of endpoints not lying in $\theta[\sigma](B)$.

Finally, $\texttt{Clean}_k$ in the HP core discards only ill-formed encodings (and set semantics already identifies duplicates). Since every realized projection is well-formed, cleaning does not remove any realized witness. □

Consequently,

$$\mathrm{Sem}_{\mathcal{C}^{\mathrm{HP}}}(\sigma) = \{(d, M, e) : \mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma, (d, M, e))\}.$$

D.8.4 Local law (INT / LSC)

We prove the local interpretation law for Hamiltonian path, using the endpoint-diagram analysis of Lemma 42 for joins.

Lemma 44 (HP interpretation law)**.** *For each constructor a of the instructive alphabet and children $(\sigma_i)_i$,*

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}\big(a(\vec{\sigma}), (d, M, e)\big) \iff \exists \vec{w} \, \Big(\bigwedge_i \mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma_i, w_i) \ \wedge \ (d, M, e) \in \widehat{a}(\vec{w}) \Big).$$

Hence $\mathcal{C}^{\mathrm{HP}}$ satisfies (LSC)*.*

Proof. We prove the equivalence by cases on the constructor a, writing $\tau := a(\vec{\sigma})$.

Leaf. ($\Rightarrow$) If $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\texttt{leaf}, (d, M, e))$ holds, then there exists $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\texttt{leaf}))$ with $\pi_\emptyset(H) = (d, M, e)$. Since $\mathcal{G}(\texttt{leaf})$ is empty, the only such H is the empty subgraph, and $\pi_\emptyset(H) = (\emptyset, \emptyset, 0)$. Hence $(d, M, e) \in \widehat{\texttt{leaf}}()$. ($\Leftarrow$) Conversely, if $(d, M, e) \in \widehat{\texttt{leaf}}()$, then $(d, M, e) = (\emptyset, \emptyset, 0)$, and the empty subgraph H witnesses $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\texttt{leaf}, (d, M, e))$.

Introduce vertex. Let $\tau = \texttt{intro\_}v(\sigma)$ and write $x := \theta[\tau](v)$ for the introduced vertex. Then x is isolated in $\mathcal{G}(\tau)$. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau, (d, M, e))$ via some $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau))$. Let $H' := H \cap \mathcal{G}(\sigma)$ be the restriction to the child graph (equivalently, delete the isolated vertex x). Then $H' \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$ and $\pi_{B(\sigma)}(H') = (d|_{B(\sigma)}, M, e)$, while $d(v) = \deg_H(x) = 0$. Hence $(d, M, e) \in \widehat{\texttt{intro\_}v}(d|_{B(\sigma)}, M, e)$ and the right-hand side of (INT) holds. ($\Leftarrow$) Conversely, suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma, w_1)$ via some $H' \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$ and $(d, M, e) \in \widehat{\texttt{intro\_}v}(w_1)$. Extend H' to a subgraph $H \subseteq \mathcal{G}(\tau)$ by adding the isolated vertex x and no edges incident to it. Then $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau))$ and $\pi_{B(\tau)}(H) = (d, M, e)$, so $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau, (d, M, e))$ holds.

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ with common bag B. Define

$$f := \{\theta[\tau](u), \theta[\tau](v)\}.$$

($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau,(d,M,e))$ via some $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau))$. If $f \notin E(H)$, then H is also a subgraph of $\mathcal{G}(\sigma)$, so with $w_1 := \pi_B(H)$ we have

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma, w_1) \quad \text{and} \quad (d,M,e) = w_1 \in \widehat{\texttt{intro\_}u\texttt{\_}v}(w_1)$$

(skip branch). If instead $f \in E(H)$, let $H' := H \setminus \{f\}$, which lies in $\mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$ because deleting an edge preserves the property "components are simple paths." Set $w_1 := \pi_B(H')$. The only changes from H' to H occur at the endpoints $\theta[\tau](u), \theta[\tau](v)$, and since H is a path subgraph, adding f does not create a cycle and does not increase any degree beyond 2. By the definition of the introduce-edge transition (the path-concatenation update of (d,M,e) described above), the witness $\pi_B(H) = (d,M,e)$ belongs to $\widehat{\texttt{intro\_}u\texttt{\_}v}(w_1)$ (use branch). This yields the right-hand side. ($\Leftarrow$) Conversely, suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma, w_1)$ via some $H' \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$ and $(d,M,e) \in \widehat{\texttt{intro\_}u\texttt{\_}v}(w_1)$. If $(d,M,e) = w_1$, take $H := H'$ (skip branch) to witness

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau,(d,M,e)).$$

Otherwise, (d,M,e) is the unique witness produced by the use branch, so w_1 encodes degrees ≤ 1 at u, v and adding f is allowed by the transition (it rejects exactly when this would create a cycle or degree > 2). Let $H := H' \cup \{f\} \subseteq \mathcal{G}(\tau)$. Then H is still a spanning selected-edge subgraph whose connected components are simple paths: the introduced edge is incident only to bag vertices, and the transition rejects cycle or degree-> 2 creations. By construction of the transition, $\pi_B(H) = (d,M,e)$, so $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau,(d,M,e))$.

Forget vertex. Let $\tau = \texttt{forget\_}v(\sigma)$ with child bag $B(\sigma) = B(\tau) \cup \{v\}$. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau,(d,M,e))$ via some $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau))$. Since $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$, the same H is a subgraph of $\mathcal{G}(\sigma)$ and we may set $w_1 := \pi_{B(\sigma)}(H)$. Then $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma, w_1)$ holds. If $\deg_H(\theta[\sigma](v)) = 2$, the forget transition restricts d and leaves (M,e) unchanged, matching $\pi_{B(\tau)}(H)$. If $\deg_H(\theta[\sigma](v)) = 1$, then $\theta[\sigma](v)$ is an endpoint of its path component, and after forgetting v it becomes an endpoint outside the bag. The endpoint counter increases by 1, and the restriction deletes v from M. This is exactly the degree-1 forget branch. If $\deg_H(\theta[\sigma](v)) = 0$, then $\theta[\sigma](v)$ is an isolated length-0 component. After forgetting v, its two coincident endpoints lie outside the bag, so the endpoint counter increases by 2. This is exactly the degree-0 forget branch. Hence $(d,M,e) \in \widehat{\texttt{forget\_}v}(w_1)$. ($\Leftarrow$) Conversely, suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\sigma, w_1)$ via some $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\sigma))$ and $(d,M,e) \in \widehat{\texttt{forget\_}v}(w_1)$. Since $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$, H is also a subgraph of $\mathcal{G}(\tau)$. If the forget transition produces (d,M,e), then $\deg_H(\theta[\sigma](v)) \in \{0,1,2\}$. In the degree-0 case, the isolated component contributes two newly forgotten endpoints; in the degree-1 case, one endpoint is newly forgotten; and in the degree-2 case, no endpoint counter changes. In each case, $\pi_{B(\tau)}(H) = (d,M,e)$ by the definition of the forget

transition, hence $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau, (d, M, e))$.

Join. Let $\tau = \mathtt{join}(\tau_1, \tau_2)$ be a join node with common bag B. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau, (d, M, e))$ via some $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau))$. Let

$$H_i := H \cap \mathcal{G}(\tau_i) \qquad (i \in \{1, 2\}).$$

Since vertices outside the common bag belong to exactly one child graph and every connected component of H is a simple path, each H_i lies in $\mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau_i))$. Write

$$w_i := \pi_B(H_i) \qquad (i \in \{1, 2\}).$$

Then $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau_i, w_i)$ holds for both children. By Lemma 42, the diagram composition does not reject and

$$\pi_B(H) = w_1 \otimes_B w_2.$$

Since the join transition is defined exactly by this composition, we obtain

$$(d, M, e) = \pi_B(H) \in \widehat{\mathtt{join}}(w_1, w_2).$$

($\Leftarrow$) Conversely, suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau_i, w_i)$ via $H_i \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau_i))$ for $i \in \{1, 2\}$, and suppose $(d, M, e) \in \widehat{\mathtt{join}}(w_1, w_2)$. By definition of the join transition, $w_1 \otimes_B w_2$ is defined and equals (d, M, e). Let

$$H := H_1 \cup H_2 \subseteq \mathcal{G}(\tau).$$

Lemma 42 now implies that $H \in \mathcal{X}_{\mathrm{HP}}(\mathcal{G}(\tau))$ and

$$\pi_B(H) = w_1 \otimes_B w_2 = (d, M, e).$$

Hence $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau, (d, M, e))$. □

D.8.5 From local law to global correctness

We now derive global correctness of the Hamiltonian-path core.

Theorem 32 (Correctness of $\mathcal{C}^{\mathrm{HP}}$)**.** *For every k and term $\tau \in \mathsf{ITD}_k$,*

$$\Gamma[\mathcal{C}^{\mathrm{HP}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}^{\mathrm{HP}}}(\tau) = \{(d, M, e) : \mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau, (d, M, e))\},$$

and $\tau \in \mathsf{Acc}(\mathcal{C}^{\mathrm{HP}}[k])$ iff $\mathcal{G}(\tau)$ has a Hamiltonian path.

Proof. Lemma 44 gives the interpretation law (INT) for HP. Lemma 27 then yields

(LSC). By Theorems 17 and 19 we obtain, for every instructive term $\tau \in \mathsf{ITD}_k$,

$$\Gamma[\mathcal{C}^{\mathrm{HP}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}^{\mathrm{HP}}}(\tau) = \{(d, M, e) : \mathsf{Interp}_{\mathcal{C}^{\mathrm{HP}}}(\tau, (d, M, e))\}.$$

For the acceptance claim, write $G := \mathcal{G}(\tau)$ and $B := B(\tau)$. Unfolding the definition of acceptance, τ is accepted iff there exists a witness $(d, M, e) \in \mathrm{Sem}_{\mathcal{C}^{\mathrm{HP}}}(\tau)$ satisfying the root condition from above. By the semantic characterization, such a witness is realized by some $H \in \mathcal{X}_{\mathrm{HP}}(G)$ with $\pi_B(H) = (d, M, e)$. The path-subgraph invariant ensures that H is a disjoint union of simple paths.

If the accepted witness satisfies case (i), then $B = \{v\}$, $d(v) = 0$, $M = \varnothing$, and $e = 0$. Thus no degree-1 endpoint is visible in the root bag and no endpoint has been forgotten. Every non-degenerate path component of H has two endpoints, and every isolated component outside the bag contributes two forgotten endpoints. Because $B = \{v\}$, $d(v) = 0$, and $e = 0$, neither kind of component can occur outside v. Hence G has no vertices outside B, so $G \simeq K_1$, which has a Hamiltonian path.

Otherwise we are in case (ii). Then every bag vertex has degree 1 or 2 in H. Each component of H is a simple path, with a length-0 component contributing two endpoints. The endpoints in the root bag are exactly the degree-1 bag vertices $E_B(d)$, while each forgotten endpoint increments e at its forget step. Hence the total number of endpoints of H equals $e + |E_B(d)| = 2$ by case (ii), so H has exactly one component and is a single spanning path. Thus G has a Hamiltonian path.

Conversely, if G has a Hamiltonian path P, then $P \in \mathcal{X}_{\mathrm{HP}}(G)$ and its projections along the decomposition yield witnesses at every node that are preserved by the transitions above. At the root, if $G \simeq K_1$ and $B = \{v\}$, then $\pi_B(P) = (\{v \mapsto 0\}, \varnothing, 0)$ and case (i) applies. If $G \simeq K_1$ and $B = \emptyset$, then the length-0 endpoint convention gives $\pi_B(P) = (\emptyset, \varnothing, 2)$, so case (ii) applies. Otherwise every vertex of G has degree 1 or 2 in P and P has exactly two endpoints, so the induced root witness satisfies case (ii). In either case, the run accepts. □

D.8.6 Equivariance and coherence

Equivariance is inherited from the symmetry arguments in Section 9.1: relabelings act by renaming bag labels in the degree profile d and the endpoint matching M (and preserve the numeric counter e), and the constructor transitions commute with this action. Coherence follows by Corollary 14.

D.8.7 Bitlength, multiplicity, and running time

We record the bitlength and multiplicity bounds for Hamiltonian-path witnesses.

Theorem 33 (HP bitlength and multiplicity)**.** *For a term of size n,*

$$\beta_{\mathrm{HP}}(k,n) = O(k\log k + \log n), \qquad \mu_{\mathrm{HP}}(k,n) \;\le\; (2n+1)\cdot 3^{k+1}\cdot I(k+1),$$

where $I(m) = \sum_{j=0}^{\lfloor m/2\rfloor} \frac{m!}{2^j\, j!\,(m-2j)!}$ is the number of partial matchings on an m-element set. Consequently,

$$T_{\mathrm{HP}}(k,n) \;=\; n\cdot k^{O(1)}\cdot (k\log k+\log n)^{O(1)}\cdot \big((2n+1)\cdot 3^k I(k)\big)^{O(1)}.$$

Here we simplify $3^{k+1}I(k+1)$ to $3^kI(k)$ up to a factor polynomial in k, absorbed into the $k^{O(1)}$ term.

Proof. Let B be a bag of size $m \le k+1$. The degree profile $d : B \to \{0,1,2\}$ has 3^m possibilities and can be encoded in $O(m)$ bits. For a fixed d, the matching component M is a partial matching on $E_B(d) \subseteq B$, hence on at most m vertices. The number of such matchings is at most $I(m)$. Encoding M by listing its pairs uses $O(m\log m)$ bits, since each bag label lies in $[k+1]$. Finally, the endpoint counter satisfies $0 \le e \le 2n$ on a term of size n, so e contributes $O(\log n)$ bits. Thus $\beta_{\mathrm{HP}}(k,n) = O(m\log m + \log n) = O(k\log k + \log n)$.

For multiplicity, a witness is determined by (d, M, e). There are at most 3^m choices for d, at most $I(m)$ choices for M, and at most $(2n+1)$ choices for e, so the total number of possible witnesses on a bag of size m is at most $(2n+1)\cdot 3^m \cdot I(m) \le (2n+1)\cdot 3^{k+1}\cdot I(k+1)$. Cleaning only removes ill-formed or dominated encodings, hence μ_{HP} is bounded by the same quantity. By Lemma 35, Theorem 4 applies. Substituting these β, μ yields the runtime bound. □

D.9 Hamiltonian Cycle: Endpoint Matchings

This section discharges the correctness proof for the Hamiltonian-cycle core $\mathcal{C}^{\mathrm{HC}}$ described in Chapter 4 (Section 4.8.2). It adapts the Hamiltonian-path construction of Section D.8 to allow closing exactly one cycle component and strengthens the forget/root conditions accordingly. We spell out the modified witnesses and show that the updated transition rules satisfy the local law, yielding global correctness and coherence for Hamiltonian cycle.

As elsewhere in Appendix D, the semantic statements below are made in the default ITD realization model, where $\mathcal{G}(\tau)$ may be a loopless multigraph. In this subsection, a Hamiltonian cycle in a loopless multigraph means a spanning connected 2-regular selected edge subgraph; in particular, two parallel edges between the same two vertices form a multigraph 2-cycle. On simple realized graphs this agrees with the ordinary simple-cycle

definition. When the intended target is the ordinary simple-graph Hamiltonian-cycle problem from Appendix B, one can pair $\mathcal{C}^{\mathrm{HC}}$ with the standard simplicity guard `NOT HasMultipleEdges()`.

D.9.1 Witness space, cleaning, and acceptance

Concretely, compared to $\mathcal{C}^{\mathrm{HP}}$:

- witnesses are triples (d, M, cc), where $\mathrm{cc} \in \{0, 1\}$ indicates whether a cycle component has been closed so far (states that would create a second cycle are discarded).
- unlike HP, an introduce-edge step that closes a component into a cycle sets $\mathrm{cc} \leftarrow \mathrm{cc} + 1$ (rather than rejecting).
- forget transitions require $d(v) = 2$, i.e. vertices are removed only once both incident Hamiltonian edges have been chosen.
- root acceptance requires $\mathrm{cc} = 1$ and $d(v) = 2$ for all remaining bag vertices v, ensuring exactly one cycle component overall and that the selected edges form a spanning 2-regular subgraph.

The remaining transitions reuse the $\mathcal{C}^{\mathrm{HP}}$ rules, with one additional update at joins: the auxiliary-graph splicing construction may produce a component with no endpoint (a cycle formed by gluing), which is rejected in $\mathcal{C}^{\mathrm{HP}}$ but instead increments cc in $\mathcal{C}^{\mathrm{HC}}$ (again discarding the state if this would create a second cycle component).

D.9.2 Transitions as set-valued functions

We write the HC transitions explicitly in terms of the HP rules on (d, M) together with the cycle counter.

- **Leaf / introduce-vertex:** use the HP transition on (d, M) and keep cc unchanged.
- **Introduce-edge uv:** keep the skip branch (d, M, cc) as in HP. For the use branch:
  - if the HP use branch on (d, M) is defined and does not create a cycle, output the same updated pair (d', M') with unchanged counter cc;
  - if $d(u) = d(v) = 1$ and u, v are paired together in M, then adding uv closes that path component into a cycle, so the use branch outputs

$$\big(d',\ M \setminus \{\{u, v\}\},\ \mathrm{cc} + 1\big),$$

where d' is obtained from d by setting $d'(u) = d'(v) = 2$ and leaving all other entries unchanged; this branch is rejected when $\mathrm{cc} = 1$.

- **Forget vertex v:** unlike HP, forgetting is allowed only when $d(v) = 2$; then restrict d to $B \setminus \{v\}$ and keep (M, cc) unchanged.
- **Join:** perform the same reduced-diagram composition on (d, M) as in HP, let c_{join} be the number of resulting endpoint-free components, and output

$$\big(d, M, \mathrm{cc}_1 + \mathrm{cc}_2 + c_{\mathrm{join}}\big)$$

exactly when the path-part composition is defined and the displayed sum is at most 1.

D.9.3 Denotation and the local interpretation predicate

We define the Hamiltonian-cycle object family and the corresponding bag projection.

Global objects. For a graph G, let $\mathcal{X}_{\mathrm{HC}}(G)$ be the set of subgraphs $H \subseteq G$ whose connected components are finite simple paths or finite simple cycles, with at most one cycle component. For a term σ with bag $B(\sigma)$, we restrict to the *saturated* objects

$$\mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma) := \Big\{ H \in \mathcal{X}_{\mathrm{HC}}(\mathcal{G}(\sigma)) : \ \forall v \in V(\mathcal{G}(\sigma)) \setminus \theta[\sigma](B(\sigma)) : \ \deg_H(v) = 2 \Big\}.$$

Thus every vertex outside the current bag is already of degree 2 in H, in accordance with the HC forget rule.

Projection. For σ and $H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma)$ define $\pi_{B(\sigma)}(H) = (d, M, \mathrm{cc})$ by:

- $d(v) := \deg_H(\theta[\sigma](v))$ for $v \in B(\sigma)$.
- M pairs $u, v \in B(\sigma)$ exactly when H has a *path* component whose two endpoints both lie in $\theta[\sigma](B(\sigma))$ and are $\theta[\sigma](u)$ and $\theta[\sigma](v)$.
- $\mathrm{cc} := 1$ iff H has a cycle component, and $\mathrm{cc} := 0$ otherwise.

Denotation and interpretation. Set

$$\mathrm{Sem}_{\mathcal{C}^{\mathrm{HC}}}(\sigma) := \mathtt{Clean}_k\big(\{\pi_{B(\sigma)}(H) : H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma)\}\big)$$

and

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\sigma, (d, M, \mathrm{cc})) \iff \exists H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma) : \ \pi_{B(\sigma)}(H) = (d, M, \mathrm{cc}).$$

Lemma 45 (Well-formed projections for HC)**.** *Let σ be a term with bag $B := B(\sigma)$ and let $H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma)$. Write $\pi_{B(\sigma)}(H) = (d, M, \mathrm{cc})$. Then (d, M, cc) is a well-formed HC witness:*

1. $d : B \to \{0, 1, 2\}$.
2. *M is a partial fixed-point-free involution on $E_B(d)$, pairing exactly those degree-1 bag labels that are endpoints of the same path component of H.*
3. $\mathrm{cc} \in \{0, 1\}$ *equals* 1 *iff H has a cycle component.*

In particular, every realized projection lies in the witness universe $W_k^{\mathrm{HC}}(B)$, so the cleaning operator is the identity on the set of realized projections.

Proof. Identical to Lemma 43 for the d and M clauses: components of H are paths or cycles, hence degrees are ≤ 2, and path components have either 0 or 2 endpoints, inducing the matching M on degree-1 bag labels. The flag cc is well-defined because $\mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma)$ contains subgraphs with at most one cycle component by definition. Since every realized projection is well-formed, the cleaning operator does not remove any realized witness. □

Consequently,

$$\mathrm{Sem}_{\mathcal{C}^{\mathrm{HC}}}(\sigma) = \{(d, M, \mathrm{cc}) : \mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\sigma, (d, M, \mathrm{cc}))\}.$$

D.9.4 Local law (INT / LSC)

We establish the local law for Hamiltonian cycle, highlighting the join case and the cycle counter update.

Lemma 46 (HC join semantics)**.** *Let $\tau = \texttt{join}(\tau_1, \tau_2)$ be a join node with common bag B. For $i \in \{1, 2\}$, let $H_i \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\tau_i)$ and write*

$$\pi_B(H_i) = (d_i, M_i, \mathrm{cc}_i).$$

Let $H := H_1 \cup H_2 \subseteq \mathcal{G}(\tau)$. Set

$$d(v) := d_1(v) + d_2(v) \qquad (v \in B),$$

and form the reduced union diagram exactly as in the HP join construction (splice bag vertices with degree pattern $(1, 1)$ and keep the remaining components). If some $d(v) > 2$ or a surviving bag endpoint has more than one incident real neighbor, the path-part composition is undefined. Otherwise define M by pairing the two bag endpoints of each reduced component that has two bag endpoints, leaving bag endpoints unmatched when a reduced component has a stub endpoint. Let c_{join} be the number of reduced components that have no endpoint. Then

$$H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\tau) \iff \textit{the path-part composition is defined and } \mathrm{cc}_1 + \mathrm{cc}_2 + c_{\mathrm{join}} \leq 1.$$

When these equivalent conditions hold,

$$\pi_B(H) = \big(d, M, \mathrm{cc}_1 + \mathrm{cc}_2 + c_{\mathrm{join}}\big).$$

In particular, the HC join transition is exactly the HP join transition on (d, M) together with the update

$$\mathrm{cc} := \mathrm{cc}_1 + \mathrm{cc}_2 + c_{\mathrm{join}},$$

rejecting iff $\mathrm{cc} > 1$.

Proof. For each $i \in \{1, 2\}$, let C_i be the unique cycle component of H_i if $\mathrm{cc}_i = 1$, and let $C_i := \emptyset$ otherwise. Put $P_i := H_i - C_i$. Then every connected component of P_i is a simple path.

The proof has two logically separate parts: existing cycle components coming from the children, and new cycle components that may be created by gluing path fragments through the bag.

First consider the child cycle components C_i. Because every vertex of H_i has degree at most 2, a bag vertex lying on C_i has degree 2 already inside child i. Hence if the other child contributed an incident edge at that bag vertex, the union H would have degree greater than 2 there. Therefore, whenever the join witness is defined, each child cycle component survives unchanged in H and contributes exactly cc_i cycle components to H.

It remains to analyze the union $P := P_1 \cup P_2$. Here the same endpoint-diagram analysis as in the proof of Lemma 42 applies: the reduced union diagram records how path components from the two children are spliced through bag vertices with degree pattern $(1, 1)$. The key output of that analysis is that every reduced-diagram component corresponds to exactly one connected component of P, with endpoints interpreted in the same way as in HP. In particular:

- reduced components with endpoints represent path components of P;
- reduced components with no endpoint represent cycle components of P.

If some $d(v) > 2$ or a surviving bag endpoint has more than one incident real neighbor, then the splicing creates branching or a bag degree greater than 2, so $H \notin \mathcal{X}_{\mathrm{HC}}(\tau)$. Otherwise every connected component of P is either a simple path or a simple cycle, and the components of P that are cycles are exactly those reduced-diagram components with no endpoint. Hence P contributes exactly c_{join} cycle components, and the bag projection of P has degree profile d and endpoint matching M defined above.

Since every vertex outside the common bag belongs to exactly one child, saturation of H_1 and H_2 implies that every vertex outside B has degree 2 in H. Therefore H lies in

$\mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\tau)$ exactly when the path-part composition is defined and the total number of cycle components

$$\mathrm{cc}_1 + \mathrm{cc}_2 + c_{\mathrm{join}}$$

is at most 1. In that case, the previous paragraph shows that the projection of H is precisely

$$\pi_B(H) = \big(d, M, \mathrm{cc}_1 + \mathrm{cc}_2 + c_{\mathrm{join}}\big).$$

This is exactly the HC join transition. □

Lemma 47 (HC interpretation law)**.** *For each constructor a of the instructive alphabet and children $(\sigma_i)_i$, we have*

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}\big(a(\vec{\sigma}), (d, M, \mathrm{cc})\big) \iff \exists \vec{w}\Big(\bigwedge_i \mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\sigma_i, w_i) \ \wedge \ (d, M, \mathrm{cc}) \in \widehat{a}(\vec{w})\Big).$$

Hence $\mathcal{C}^{\mathrm{HC}}$ satisfies (LSC).

Proof. We argue constructor-wise. The proof is parallel to Lemma 44 (HP), with the additional cycle-component bookkeeping.

Leaf and introduce-vertex. The proofs are exactly the same as in Lemma 44, with the additional observation that no edge is added, so no cycle component can be created and the witness updates preserve $\mathrm{cc} = 0$.

Introduce edge uv. As in HP, either the edge is not selected (skip branch) or it is selected (use branch). In the use branch, adding uv may connect two distinct path components or extend one path component, updating (d, M) exactly as in HP. The only new case is when u and v are the two endpoints of the *same* path component: then adding uv closes that component into a cycle. In this case we set $\mathrm{cc} \leftarrow 1$, delete the pair $\{u, v\}$ from M, update both endpoint degrees to 2, and reject if cc was already 1 (as this would create a second cycle component).

Forget vertex v. The HC forget rule requires $d(v) = 2$, so no endpoint can disappear into the forgotten region. Therefore the endpoint matching M and the cycle flag cc are preserved under restriction to $B \setminus \{v\}$.

Join. Let $\tau = \mathtt{join}(\sigma_1, \sigma_2)$ be a join node with common bag B. ($\Rightarrow$) Suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\tau, (d, M, \mathrm{cc}))$ via some $H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\tau)$. For $i \in \{1, 2\}$ let $H_i := H \cap \mathcal{G}(\sigma_i)$ and write $\pi_B(H_i) = w_i$. Restricting a subgraph whose components are paths or cycles can only turn components into paths or cycles, and every vertex outside B belongs to a unique child; hence each H_i lies in $\mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma_i)$. Thus $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\sigma_i, w_i)$ holds. Applying Lemma 46 to H_1 and H_2 yields $(d, M, \mathrm{cc}) \in \widehat{\mathtt{join}}(w_1, w_2)$.

($\Leftarrow$) Conversely, suppose $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\sigma_i, w_i)$ via $H_i \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\sigma_i)$ for $i \in \{1, 2\}$ and $(d, M, \mathrm{cc}) \in \widehat{\mathtt{join}}(w_1, w_2)$. Let $H := H_1 \cup H_2 \subseteq \mathcal{G}(\tau)$. By Lemma 46, $H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\tau)$ and $\pi_B(H) =$

(d, M, cc). Hence $\mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\tau, (d, M, \mathrm{cc}))$. □

D.9.5 From local law to global correctness

We now derive global correctness of the Hamiltonian-cycle core.

Theorem 34 (Correctness of $\mathcal{C}^{\mathrm{HC}}$)**.** *For every k and term $\tau \in \mathsf{ITD}_k$,*

$$\Gamma[\mathcal{C}^{\mathrm{HC}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}^{\mathrm{HC}}}(\tau) = \{(d, M, \mathrm{cc}) : \mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\tau, (d, M, \mathrm{cc}))\}.$$

Moreover, $\tau \in \mathsf{Acc}(\mathcal{C}^{\mathrm{HC}}[k])$ iff $\mathcal{G}(\tau)$ has a Hamiltonian cycle.

Proof. Lemma 47 gives (INT), implying (LSC) via Lemma 27. By Theorems 17 and 19 we obtain, for every instructive term $\tau \in \mathsf{ITD}_k$,

$$\Gamma[\mathcal{C}^{\mathrm{HC}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}^{\mathrm{HC}}}(\tau) = \{(d, M, \mathrm{cc}) : \mathsf{Interp}_{\mathcal{C}^{\mathrm{HC}}}(\tau, (d, M, \mathrm{cc}))\}.$$

For the acceptance claim, write $G := \mathcal{G}(\tau)$ and $B := B(\tau)$.

Suppose τ is accepted. Then there exists $(d, M, \mathrm{cc}) \in \mathrm{Sem}_{\mathcal{C}^{\mathrm{HC}}}(\tau)$ satisfying the root condition ($\mathrm{cc} = 1$ and $d(v) = 2$ for all $v \in B$). Unwinding the denotation, this witness is realized by some $H \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\tau)$ with $\pi_B(H) = (d, M, \mathrm{cc})$. By saturation, every vertex outside the bag has degree 2 in H, and by the root condition the same holds for every bag vertex. Hence H is a spanning 2-regular subgraph of G, i.e., a disjoint union of cycles. Since $\mathcal{X}_{\mathrm{HC}}$ allows at most one cycle component and $\mathrm{cc} = 1$ records that a cycle component exists, H is a single spanning cycle. Thus G has a Hamiltonian cycle.

Conversely, if G has a Hamiltonian cycle C, then $C \in \mathcal{X}_{\mathrm{HC}}(G)$ and every vertex has degree 2 in C, so in particular $C \in \mathcal{X}^{\mathrm{sat}}_{\mathrm{HC}}(\tau)$. Its projection yields a root witness with $\mathrm{cc} = 1$ and $d(v) = 2$ for all $v \in B$, so τ is accepted. □

D.9.6 Equivariance and coherence

Equivariance is inherited from HP: relabelings act by permuting bag labels, and the HC transitions commute with this action (including the cc update). Hence coherence follows by Corollary 14.

D.9.7 Bitlength, multiplicity, and running time

We record the bitlength, multiplicity, and runtime bounds for Hamiltonian cycle.

Theorem 35 (HC bitlength and multiplicity)**.**

$$\beta_{\mathrm{HC}}(k, n) = O(k \log k),$$

$$\mu_{\mathrm{HC}}(k,n) \le 2 \cdot 3^{k+1} \cdot I(k{+}1),$$

and

$$T_{\mathrm{HC}}(k,n) = n \cdot k^{O(1)} (\log k)^{O(1)} \cdot (3^k I(k))^{O(1)}.$$

In the runtime bound we simplify $3^{k+1} I(k+1)$ to $3^k I(k)$ up to a factor polynomial in k, absorbed into the $k^{O(1)}$ term.

Proof. At a bag B of size $m \le k+1$, the degree profile $d : B \to \{0,1,2\}$ has 3^m possibilities and is encodable in $O(m)$ bits. The matching component M is a partial matching on $E_B(d) \subseteq B$, hence on at most m vertices. There are at most $I(m)$ possibilities. Encoding M by listing its pairs uses $O(m \log m)$ bits. The cycle counter $\mathrm{cc} \in \{0,1\}$ is constant-size. Thus $\beta_{\mathrm{HC}}(k,n) = O(m \log m) = O(k \log k)$.

For multiplicity, a witness is determined by (d, M, cc), so there are at most $2 \cdot 3^m \cdot I(m)$ candidates at bag size m. Since $m \le k+1$, we obtain

$$\mu_{\mathrm{HC}}(k,n) \le 2 \cdot 3^{k+1} \cdot I(k+1).$$

By Lemma 35, Corollary 2 applies. The runtime bound follows. □

D.10 Feedback Vertex Set ($\le t$)

This section discharges the correctness proof for the feedback-vertex-set core $\mathcal{C}_t^{\mathrm{FVS}}$ from Section 4.8.2 and instantiates the proof framework of Section 9.1.

D.10.1 Witness space, cleaning, and acceptance

Fix $k, t \in \mathbb{N}$ and a bag $B \subseteq [k{+}1]$.

Witness space. At bag B a witness is a triple (D, Π, s) where $D \subseteq B$ is the set of deleted bag labels, Π is a partition of $B \setminus D$, and $s \in \{0,1,2,\dots\}$ counts deleted vertices outside the bag. We write $W_k^{\mathrm{FVS}}(B)$ for the set of such triples and $W_k^{\mathrm{FVS}} := \bigcup_{B \subseteq [k+1]} W_k^{\mathrm{FVS}}(B)$ for the ambient witness space.

Cleaning and acceptance. Cleaning retains, for each fixed signature (D, Π), only the witness with minimal counter s. The finality predicate accepts (D, Π, s) iff $s + |D| \le t$ and the invariant returns $\min(s + |D|)$.

D.10.2 Transitions as set-valued functions

We restate the transition rules here (with the catalog anchor at Section 4.8.2).

Notation (partitions). For a partition Π write $\mathrm{dom}(\Pi)$ for its domain and $\mathrm{blk}_\Pi(u)$ for the block containing $u \in \mathrm{dom}(\Pi)$. For $u \in \mathrm{dom}(\Pi)$, let $\Pi \setminus u$ be the partition obtained by removing u from its block (deleting the block if it becomes empty). For $u \neq v$ with $u, v \in \mathrm{dom}(\Pi)$ and $\mathrm{blk}_\Pi(u) \neq \mathrm{blk}_\Pi(v)$, let $\Pi\mathrm{merge}_{u,v}$ be the partition obtained by merging the two blocks. For partitions Π_1, Π_2 on a common domain U, let $\Pi_1 \sqcup \Pi_2$ be the partition induced by the transitive closure of the equivalence relation generated by $\Pi_1 \cup \Pi_2$.

Transitions. Writing B for the current bag and $U := B \setminus D$ for the surviving bag labels:

$$\begin{aligned}
\mathcal{C}_t^{\mathrm{FVS}}[k].\texttt{LeafSet} &= \{(\emptyset, \emptyset, 0)\},\\
\mathcal{C}_t^{\mathrm{FVS}}[k].\texttt{intro_}u(D, \Pi, s) &= \{(D \cup \{u\}, \Pi, s),\ (D, \Pi \cup \{\{u\}\}, s)\},\\
\mathcal{C}_t^{\mathrm{FVS}}[k].\texttt{forget_}u(D, \Pi, s) &= \begin{cases} \{(D \setminus \{u\}, \Pi, s+1)\}, & u \in D,\\ \{(D, \Pi \setminus u, s)\}, & u \notin D, \end{cases}\\
\mathcal{C}_t^{\mathrm{FVS}}[k].\texttt{intro_}u\texttt{_}v(D, \Pi, s) &= \begin{cases} \{(D, \Pi, s)\}, & u \in D \text{ or } v \in D,\\ \emptyset, & \substack{u,v \notin D\\ \mathrm{blk}_\Pi(u) = \mathrm{blk}_\Pi(v)},\\ \{(D, \Pi\mathrm{merge}_{u,v}, s)\}, & \text{otherwise}, \end{cases}\\
\mathcal{C}_t^{\mathrm{FVS}}[k].\texttt{join}\big((D, \Pi_1, s_1), (D', \Pi_2, s_2)\big) &= \begin{cases} \{(D, \Pi_1 \sqcup \Pi_2, s_1 + s_2)\}, & \substack{D = D'\\ H(\Pi_1, \Pi_2) \text{ is acyclic}},\\ \emptyset, & \text{otherwise}. \end{cases}
\end{aligned}$$

where $H(\Pi_1, \Pi_2)$ is the component-intersection multigraph whose left vertices are the blocks of Π_1, whose right vertices are the blocks of Π_2, and which has, for every $x \in U$, an edge labeled x between $\mathrm{blk}_{\Pi_1}(x)$ and $\mathrm{blk}_{\Pi_2}(x)$.

D.10.3 Denotation and the local interpretation predicate

We now define global objects for feedback vertex set and the bag projection used in denotation.

Global objects. For a graph G let

$$X_{\mathrm{FVS}}(G) := \{\, X \subseteq V(G) : G - X \text{ is a forest} \,\}.$$

Projection. For a term σ with bag $B(\sigma)$ and labeling $\theta[\sigma] : B(\sigma) \to V(\mathcal{G}(\sigma))$ define the bag projection

$$\pi_{B(\sigma)}(X) := (D, \Pi, s), \qquad X \in X_{\mathrm{FVS}}(\mathcal{G}(\sigma)),$$

where

(i) $D := \theta[\sigma]^{-1}(X) \cap B(\sigma)$,

(ii) $s := |X \setminus \theta[\sigma](B(\sigma))|$, and

(iii) Π is the partition of $B(\sigma) \setminus D$ induced by connected components of the forest $\mathcal{G}(\sigma) - X$.

Denotation and interpretation. Set

$$\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\sigma) := \mathtt{Clean}_k\big(\{\pi_{B(\sigma)}(X) : X \in X_{\mathrm{FVS}}(\mathcal{G}(\sigma))\}\big),$$

and define

$$\begin{aligned}\mathsf{Interp}_{\mathrm{FVS}}(\sigma,(D,\Pi,s)) \iff \exists X \in X_{\mathrm{FVS}}(\mathcal{G}(\sigma)) \text{ such that} &\text{(i) } \pi_{B(\sigma)}(X) = (D,\Pi,s),\\ &\text{(ii) } \forall X' \in X_{\mathrm{FVS}}(\mathcal{G}(\sigma)) :\\ &\quad \pi_{B(\sigma)}(X') = (D,\Pi,s')\\ &\quad \Rightarrow\ s' \geq s.\end{aligned}$$

Then $\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\sigma) = \{(D,\Pi,s) : \mathsf{Interp}_{\mathrm{FVS}}(\sigma,(D,\Pi,s))\}$ by the definition of cleaning. Unpacking the definitions shows that $\mathsf{Interp}_{\mathrm{FVS}}$ coincides with the feedback-vertex-set catalog predicate $\mathsf{P}_{t,k}^{\mathrm{FVS}}$ from Chapter 4.

D.10.4 A join lemma for gluing forests along a bag

We start with a join lemma characterizing when two forests glue to a forest.

Lemma 48 (Forest gluing via $H(\Pi_1,\Pi_2)$)**.** *Let $\tau = \mathtt{join}(\tau_1,\tau_2)$ be a join node with common bag B. Fix $X_i \in X_{\mathrm{FVS}}(\mathcal{G}(\tau_i))$ and set $F_i := \mathcal{G}(\tau_i) - X_i$. Assume the deleted bag vertices agree, i.e.*

$$\theta[\tau_1]^{-1}(X_1) \cap B = \theta[\tau_2]^{-1}(X_2) \cap B.$$

Let D denote this common set and put $U := B \setminus D$. For $i \in \{1,2\}$ let Π_i be the partition of U induced by connected components of F_i. Let $X := \iota_1(X_1) \cup \iota_2(X_2) \subseteq V(\mathcal{G}(\tau))$ and $F := \mathcal{G}(\tau) - X$. Then:

(a) F is a forest if and only if $H(\Pi_1,\Pi_2)$ is acyclic (in the multigraph sense, so parallel edges form a cycle).

(b) If $H(\Pi_1,\Pi_2)$ is acyclic, then the partition of U induced by connected components of F is exactly $\Pi_1 \sqcup \Pi_2$.

Proof. Each F_i is a forest, so every connected component of F_i is a tree. Moreover, by the ITD pushout semantics the forests F_1 and F_2 intersect only on bag vertices in U.

(a) $\Rightarrow$. Assume $H(\Pi_1, \Pi_2)$ contains a cycle. Take a simple cycle in H; it alternates between blocks of Π_1 and blocks of Π_2:

$$C_1 \xrightarrow{x_1} D_1 \xrightarrow{x_2} C_2 \xrightarrow{x_3} \cdots \xrightarrow{x_m} D_m \xrightarrow{x_{m+1}} C_1,$$

where $C_j \in \Pi_1$, $D_j \in \Pi_2$, and each $x_j \in U$ labels the corresponding edge. For each block $C_j \in \Pi_1$, the vertices x_j and x_{j+1} lie in the same tree component of F_1, so there is a unique F_1-path P_j^1 between them. Similarly, for each $D_j \in \Pi_2$ there is a unique F_2-path P_j^2 between x_{j+1} and x_{j+2} (indices modulo $m+1$). Concatenating these paths yields a closed walk in F. Any closed walk contains a cycle, hence F is not acyclic and therefore not a forest.

(a) $\Leftarrow$. Assume F contains a cycle Z. Since each F_i is a forest, Z cannot lie entirely inside F_1 or entirely inside F_2, so it must use edges from both sides. Whenever Z switches from an edge of F_1 to an edge of F_2 (or vice versa), it does so at a vertex of U. Traversing Z and recording which tree-components of F_1 and F_2 are visited between successive vertices of U yields a closed walk in the incidence structure of blocks of Π_1 and Π_2 with edges labeled by vertices of U, i.e. a closed walk in $H(\Pi_1, \Pi_2)$. Every closed walk contains a cycle, so $H(\Pi_1, \Pi_2)$ is cyclic.

(b). Assume $H(\Pi_1, \Pi_2)$ is acyclic. Two vertices $x, y \in U$ lie in the same component of F iff there exists a path in F from x to y. Such a path alternates between subpaths inside components of F_1 and components of F_2, and therefore corresponds exactly to an alternating sequence of blocks of Π_1 and Π_2 connecting x to y. Equivalently, x and y are related in the transitive closure of the relations "same block in Π_1" and "same block in Π_2", which is precisely $\Pi_1 \sqcup \Pi_2$. □

D.10.5 Local law (INT / LSC)

We prove the LSC equation for FVS by showing that transitions compute the minimum outside-count per signature.

Lemma 49 (FVS (LSC))**.** *Fix k. Let a be an instructive constructor of arity r, and let $\sigma_1, \ldots, \sigma_r \in \mathsf{ITD}_k$ be child terms such that $\tau := a(\vec{\sigma}) \in \mathsf{ITD}_k$. Then*

$$\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\tau) = \mathsf{F}_a\Big(\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\sigma_1), \ldots, \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\sigma_r)\Big).$$

Proof. For a term σ with bag $B(\sigma)$ and a signature (D, Π) with $D \subseteq B(\sigma)$ and Π a

partition of $B(\sigma) \setminus D$, define

$$m_\sigma(D,\Pi) := \min\big\{|X \setminus \theta[\sigma](B(\sigma))| :\ X \in X_{\mathrm{FVS}}(\mathcal{G}(\sigma)) \\ \wedge\ \pi_{B(\sigma)}(X) = (D,\Pi,s) \text{ for some } s\big\},$$

with $m_\sigma(D,\Pi) = +\infty$ if no such X exists. By construction of cleaning,

$$(D,\Pi,s) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\sigma) \quad\Longleftrightarrow\quad s = m_\sigma(D,\Pi).$$

We show that the transition rules compute these minima correctly in each constructor.

Leaf. $\mathcal{G}(\texttt{leaf})$ has no vertices. The unique feasible signature is $(D,\Pi) = (\emptyset,\emptyset)$ with $m_{\texttt{leaf}}(\emptyset,\emptyset) = 0$, matching $\mathcal{C}_t^{\mathrm{FVS}}[k].\texttt{LeafSet}$.

Introduce u. Let $\tau = \texttt{intro\_}u(\sigma)$ and write $x := \theta[\tau](u)$ for the introduced isolated vertex. Every $X \in X_{\mathrm{FVS}}(\mathcal{G}(\tau))$ is either a feedback vertex set of $\mathcal{G}(\sigma)$ (with $x \notin X$) or is of the form $X' \cup \{x\}$ where $X' \in X_{\mathrm{FVS}}(\mathcal{G}(\sigma))$. In the first case the signature gains a singleton block $\{u\}$ in the partition; in the second case the deletion set gains u. In both cases, the outside-count is unchanged because x lies in the bag. This is exactly what $\widehat{\texttt{intro\_}u}$ generates, and cleaning keeps the minimum per signature.

Forget u. Let $\tau = \texttt{forget\_}u(\sigma)$. We have $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$ and $B(\tau) = B(\sigma) \setminus \{u\}$. If $u \in D$ then u moves from the bag into the outside deletion counter, so $m_\tau(D \setminus \{u\},\Pi) = m_\sigma(D,\Pi) + 1$. If $u \notin D$ then u remains in the forest and forgetting it simply removes it from the boundary partition, so $m_\tau(D,\Pi \setminus u) = m_\sigma(D,\Pi)$. These are exactly the two cases implemented by $\widehat{\texttt{forget\_}u}$, followed by cleaning.

Introduce edge $\{u,v\}$. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ and write $e := \{\theta[\tau](u),\theta[\tau](v)\}$ for the introduced edge. If $u \in D$ or $v \in D$, then e is incident to a deleted vertex and disappears in $\mathcal{G}(\tau) - X$, so the signature and outside-count are unchanged. If $u,v \notin D$ and $\mathrm{blk}_\Pi(u) = \mathrm{blk}_\Pi(v)$, then u and v are already connected in the forest $\mathcal{G}(\sigma) - X$ and adding e creates a cycle, so no feedback vertex set can realize this signature in τ. Otherwise u and v lie in distinct tree components and adding e merges them into a single tree component, so the partition merges the two blocks and the outside-count is unchanged. This matches $\widehat{\texttt{intro\_}u\texttt{\_}}v$.

Join. Let $\tau = \texttt{join}(\sigma_1,\sigma_2)$ with common bag B. Write

$$M(D,\Pi) := \min\{\, m_{\sigma_1}(D,\Pi_1) + m_{\sigma_2}(D,\Pi_2) :\ \Pi = \Pi_1 \sqcup \Pi_2 \text{ and } H(\Pi_1,\Pi_2) \text{ is acyclic}\,\},$$

with $M(D,\Pi) = +\infty$ if no such pair exists.

If $M(D,\Pi) < +\infty$, fix any pair (Π_1,Π_2) occurring in the minimum and choose witnesses $X_i \subseteq V(\mathcal{G}(\sigma_i))$ with $\pi_B(X_i) = (D,\Pi_i,m_{\sigma_i}(D,\Pi_i))$. Form the pushout union $X :=$

$\iota_1(X_1) \cup \iota_2(X_2) \subseteq V(\mathcal{G}(\tau))$. The outside vertices of τ are the disjoint union of outside vertices from the children, so

$$|X \setminus \theta[\tau](B)| = |X_1 \setminus \theta[\sigma_1](B)| + |X_2 \setminus \theta[\sigma_2](B)|.$$

By Lemma 48, the residual graph $\mathcal{G}(\tau) - X$ is a forest iff $H(\Pi_1, \Pi_2)$ is acyclic, and in that case the induced boundary partition is $\Pi_1 \sqcup \Pi_2$.

$$\pi_B(X) = (D, \Pi, m_{\sigma_1}(D, \Pi_1) + m_{\sigma_2}(D, \Pi_2)).$$

Hence $m_\tau(D, \Pi) \leq m_{\sigma_1}(D, \Pi_1) + m_{\sigma_2}(D, \Pi_2)$, and taking the minimum over all feasible pairs gives

$$m_\tau(D, \Pi) \leq M(D, \Pi).$$

If $M(D, \Pi) = +\infty$, this inequality is vacuous.

If $m_\tau(D, \Pi) < +\infty$, let $X \subseteq V(\mathcal{G}(\tau))$ witness $m_\tau(D, \Pi)$, so $\pi_B(X) = (D, \Pi, m_\tau(D, \Pi))$ and $\mathcal{G}(\tau) - X$ is a forest. Set $X_i := \iota_i^{-1}(X) \subseteq V(\mathcal{G}(\sigma_i))$ and let Π_i be the partition of $B \setminus D$ induced by $\mathcal{G}(\sigma_i) - X_i$. Then each X_i is a feedback vertex set of the child graph, the deleted bag set in each child is again D, and

$$|X \setminus \theta[\tau](B)| = |X_1 \setminus \theta[\sigma_1](B)| + |X_2 \setminus \theta[\sigma_2](B)|.$$

Applying Lemma 48 to X_1, X_2 shows that $H(\Pi_1, \Pi_2)$ is acyclic and $\Pi = \Pi_1 \sqcup \Pi_2$. Therefore (Π_1, Π_2) is feasible for $M(D, \Pi)$, and

$$M(D, \Pi) \leq m_{\sigma_1}(D, \Pi_1) + m_{\sigma_2}(D, \Pi_2) \leq |X \setminus \theta[\tau](B)| = m_\tau(D, \Pi).$$

If $m_\tau(D, \Pi) = +\infty$, the inequality $M(D, \Pi) \leq m_\tau(D, \Pi)$ is automatic. Combining the two inequalities yields $m_\tau(D, \Pi) = M(D, \Pi)$, which is exactly what $\widehat{\texttt{join}}$ followed by cleaning produces.

In every constructor, the cleaned transition F_a computes precisely the minima $m_{a(\vec{\sigma})}(\cdot)$, so (LSC) holds for $\mathcal{C}_t^{\mathrm{FVS}}$. □

D.10.6 Global correctness

We now state the global correctness result that links the denotation to the intended invariant.

Theorem 36 (Feedback vertex set correctness)**.** *For every t, k and term $\tau \in \mathsf{ITD}_k$,*

$$\Gamma[\mathcal{C}_t^{\mathrm{FVS}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\tau) = \{(D, \Pi, s) : \mathsf{Interp}_{\mathrm{FVS}}(\tau, (D, \Pi, s))\}.$$

Moreover,

$$\min\{s + |D| : (D, \Pi, s) \in \Gamma[\mathcal{C}_t^{\mathrm{FVS}}, k](\tau)\} = \min\{|X| : X \in X_{\mathrm{FVS}}(\mathcal{G}(\tau))\},$$

and τ is accepted by $\mathcal{C}_t^{\mathrm{FVS}}[k]$ iff $\mathcal{G}(\tau)$ has a feedback vertex set of size at most t.

Proof. Lemma 49 yields (LSC), so Theorem 17 gives

$$\Gamma[\mathcal{C}_t^{\mathrm{FVS}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\tau) = \{(D, \Pi, s) : \mathsf{Interp}_{\mathrm{FVS}}(\tau, (D, \Pi, s))\}$$

for every $\tau \in \mathsf{ITD}_k$.

Write

$$\mu_\tau := \min\{s + |D| : (D, \Pi, s) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\tau)\}$$

and

$$\nu_\tau := \min\{|X| : X \in X_{\mathrm{FVS}}(\mathcal{G}(\tau))\}.$$

If $(D, \Pi, s) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\tau)$, then $\mathsf{Interp}_{\mathrm{FVS}}$ gives $X \in X_{\mathrm{FVS}}(\mathcal{G}(\tau))$ with $\pi_{B(\tau)}(X) = (D, \Pi, s)$, and therefore

$$|X| = |X \setminus \theta[\tau](B(\tau))| + |\theta[\tau]^{-1}(X) \cap B(\tau)| = s + |D|.$$

Hence $\nu_\tau \le \mu_\tau$.

Conversely, let $X \in X_{\mathrm{FVS}}(\mathcal{G}(\tau))$ and write $\pi_{B(\tau)}(X) = (D, \Pi, s_0)$. By the definition of cleaning, $\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{FVS}}}(\tau)$ contains some witness (D, Π, s) with $s \le s_0$. Thus

$$\mu_\tau \le s + |D| \le s_0 + |D| = |X|.$$

Taking the minimum over all X shows $\mu_\tau \le \nu_\tau$. Therefore $\mu_\tau = \nu_\tau$, which is exactly the displayed invariant identity.

Finally, τ is accepted by $\mathcal{C}_t^{\mathrm{FVS}}[k]$ iff it has a final witness (D, Π, s) with $s + |D| \le t$, i.e. iff $\mu_\tau \le t$. By the equality $\mu_\tau = \nu_\tau$, this holds iff $\mathcal{G}(\tau)$ has a feedback vertex set of size at most t. □

D.10.7 Equivariance and coherence

Equivariance is immediate: relabelings act by renaming bag elements in D and in each block of Π and fix the counter s, and every transition commutes with this action (including the join guard via $H(\Pi_1, \Pi_2)$). Coherence follows by Corollary 14.

D.10.8 Bitlength, multiplicity, and running time

We record the bitlength and multiplicity bounds for FVS witnesses.

Theorem 37 (FVS bitlength and multiplicity)**.**

$$\beta_{\mathrm{FVS}}(k,n) = O(k\log k + \log n), \qquad \mu_{\mathrm{FVS}}(k,n) \le 2^{k+1}\mathrm{B}_{k+1},$$

and hence

$$T_{\mathrm{FVS}}(k,n) = n \cdot k^{O(1)} \cdot (k\log k + \log n)^{O(1)} \cdot \left(2^{k+1}\mathrm{B}_{k+1}\right)^{O(1)}$$

by Theorem 4.

Proof. At a bag B of size $m \le k+1$, the deleted set $D \subseteq B$ is encodable in $O(m)$ bits. The partition Π of $B \setminus D$ can be encoded by assigning canonical block IDs to elements of $B \setminus D$, which uses $O(m \log m)$ bits. Finally, the counter s is at most $|V(\mathcal{G}(\sigma))| \le n$ and is encodable in $O(\log n)$ bits. Thus $\beta_{\mathrm{FVS}}(k,n) = O(m\log m + \log n) = O(k\log k + \log n)$.

Cleaning keeps at most one witness per signature (D,Π). For a bag of size m, there are 2^m choices for D and, for each D, at most $\mathrm{B}_{m-|D|} \le \mathrm{B}_m$ partitions of $B \setminus D$. Hence $\mu_{\mathrm{FVS}}(k,n) \le 2^m\mathrm{B}_m \le 2^{k+1}\mathrm{B}_{k+1}$. By Lemma 35, Theorem 4 applies. The runtime bound follows. □

D.11 Supplementary Framework Notes

D.11.1 Abstract interpretation (optional)

Remark 20 (Optional abstract-interpretation reading). For each instructive constructor instance

$$\tau = a(\sigma_1, \ldots, \sigma_r),$$

write $G_i := \mathcal{G}(\sigma_i)$, $G := \mathcal{G}(\tau)$, $B_i := B(\sigma_i)$, and $B := B(\tau)$. Suppose one has chosen concrete transformers

$$F^{\mathrm{concr}}_{a,\tau} : \prod_{i=1}^{r} \mathcal{P}_{\mathrm{fin}}(\mathcal{X}(G_i)) \to \mathcal{P}_{\mathrm{fin}}(\mathcal{X}(G))$$

such that

$$\mathrm{Conc}(\tau) = F^{\mathrm{concr}}_{a,\tau}\big(\mathrm{Conc}(\sigma_1), \ldots, \mathrm{Conc}(\sigma_r)\big),$$

and such that for all X_i the following square commutes:

$$\alpha^{\mathrm{can}}_{B}\big(F^{\mathrm{concr}}_{a,\tau}(X_1, \ldots, X_r)\big) = \mathsf{F}_{a,B}\big(\alpha^{\mathrm{can}}_{B_1}(X_1), \ldots, \alpha^{\mathrm{can}}_{B_r}(X_r)\big),$$

where $\alpha_B^{\mathrm{can}} := \mathtt{Clean}_k \circ \alpha_B$. Then, by structural induction on τ,

$$\Gamma[\mathcal{D}, k](\tau) = \alpha^{\mathrm{can}}_{B(\tau)}(\mathrm{Conc}(\tau)).$$

When $\mathrm{Conc}(\tau) = \mathcal{X}(\mathcal{G}(\tau))$, this is exactly $\mathrm{Sem}_{\mathcal{D}}(\tau)$. We do not use this viewpoint later; it is only an optional reformulation of the same constructor-wise law captured by (LSC).

Lemma 50 (Quantitative cores: pointwise-optimal canonization)**.** *Suppose witnesses have the form (s, c) where s is a finite signature (e.g. $s \subseteq B$) and $c \in \mathbb{N} \cup \{\infty\}$ is a cost. Assume $\mathtt{Clean}_k$ retains, for each fixed signature s, only those pairs (s, c) with minimal c among the set, breaking ties in a fixed (but otherwise arbitrary) deterministic way.*

Define for each term σ the value function $m_\sigma(s) := \min\{c : (s, c) \in \mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\sigma)\}$ (with $\min \emptyset := \infty$). If for each constructor a the transition rules implement the corresponding Bellman-type recurrence for $m_{a(\vec{\sigma})}$ from the child functions m_{σ_i}, then $\mathcal{D}$ satisfies (LSC).

Proof. Fix a constructor a of arity r and terms $\sigma_1, \ldots, \sigma_r$, and abbreviate $S_i := \mathrm{Sem}_{\mathcal{D}}(\sigma_i) = \mathtt{Clean}_k(\mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\sigma_i))$. Let $B := B(a(\sigma_1, \ldots, \sigma_r))$. For each signature s, the assumption on $\mathtt{Clean}_k$ implies that S_i contains at most one pair with first component s, and if $m_{\sigma_i}(s) < \infty$ then $(s, m_{\sigma_i}(s)) \in S_i$.

Let

$$U := \bigcup_{(w_1,\ldots,w_r) \in S_1 \times \cdots \times S_r} \Delta_{k,a}(B, w_1, \ldots, w_r) \subseteq W_k$$

and define the induced candidate optimum

$$m'(s) := \min\{c : (s, c) \in U\} \qquad (\min \emptyset := \infty).$$

By construction of $\mathsf{F}_{a,B}$ and the cleaning policy, we have $\mathsf{F}_{a,B}(S_1, \ldots, S_r) = \{(s, m'(s)) : m'(s) < \infty\}$. By the Bellman-recurrence hypothesis, $m'(s) = m_{a(\vec{\sigma})}(s)$ for all s, hence

$$\mathsf{F}_{a,B}(\mathrm{Sem}_{\mathcal{D}}(\sigma_1), \ldots, \mathrm{Sem}_{\mathcal{D}}(\sigma_r)) = \mathrm{Sem}_{\mathcal{D}}\big(a(\sigma_1, \ldots, \sigma_r)\big),$$

which is exactly (LSC). □

D.11.2 Derivations and proof certificates

The interpretation law admits a proof-theoretic reading, yielding an explicit derivation system.

Definition 62 (DP-derivations)**.** *Fix k and a DP-core instance $\mathcal{D}$. Define a judgment $\tau \vdash w$ for instructive terms $\tau \in \mathsf{ITD}_k$ and witnesses $w \in W_k$ by the following rule scheme:*

whenever $\tau = a(\tau_1, \ldots, \tau_r)$*, one may infer*

$$\frac{\tau_1 \vdash w_1 \quad \cdots \quad \tau_r \vdash w_r \quad w \in \Delta_{k,a}(B(\tau), w_1, \ldots, w_r)}{\tau \vdash w}.$$

(For $r = 0$ *this means* $\tau \vdash w$ *whenever* $\tau = a$ *and* $w \in \Delta_{k,a}(B(\tau))$*.)*

Lemma 51 (Derivability yields certificates)**.** *Fix* k *and a DP-core instance* $\mathcal{D}$*. For every instructive term* $\tau \in \mathsf{ITD}_k$*,*

$$\Gamma[\mathcal{D}, k](\tau) \subseteq \{\, w \in W_k : \tau \vdash w \,\}.$$

Proof. We proceed by structural induction on τ. If $\tau = b$ is a nullary constructor, then

$$\Gamma[\mathcal{D}, k](\tau) = \mathtt{Clean}_k(\Delta_{k,b}(B(\tau))) \subseteq \Delta_{k,b}(B(\tau))$$

since cleaning is shrinking, and $\tau \vdash w$ holds for every $w \in \Delta_{k,b}(B(\tau))$ by definition of the derivation system.

For the inductive step, let $\tau = a(\tau_1, \ldots, \tau_r)$ and pick $w \in \Gamma[\mathcal{D}, k](\tau)$. By Definition 54, we have $w \in \mathtt{Clean}_k(U)$ where

$$U := \bigcup_{(w_1,\ldots,w_r) \in \prod_i \Gamma[\mathcal{D},k](\tau_i)} \Delta_{k,a}(B(\tau), w_1, \ldots, w_r).$$

Since cleaning is shrinking, $w \in U$, so there exist witnesses $w_1, \ldots, w_r$ with $w_i \in \Gamma[\mathcal{D}, k](\tau_i)$ and $w \in \Delta_{k,a}(B(\tau), w_1, \ldots, w_r)$. By the induction hypothesis, each $\tau_i \vdash w_i$ is derivable, hence by the inference rule for a we obtain $\tau \vdash w$. □

Corollary 15 (Witness sets admit linear-step derivations)**.** *Every* $w \in \Gamma[\mathcal{D}, k](\tau)$ *has a derivation tree with* $O(|\tau|)$ *inference steps witnessing* $\tau \vdash w$*. No claim is made here about the bitlength of the witness labels stored at those derivation nodes. In particular, acceptance admits certificates with a linear number of derivation steps relative to the input term.*

Proof. By Lemma 51, every computed witness w is derivable. The derivation follows the term structure: each node of τ contributes exactly one inference step, so the derivation tree has size $O(|\tau|)$. □

Remark on cleaning. By definition, denotation is canonized: $\mathtt{Clean}(\mathrm{Sem}_{\mathcal{D}}(\sigma)) = \mathrm{Sem}_{\mathcal{D}}(\sigma)$ for all σ. When cleaning is the identity on the denotational sets of a core (as in most decision cores), one may omit the explicit `Clean` in the definition of $\mathrm{Sem}_{\mathcal{D}}$. When cleaning is nontrivial (e.g. "keep the minimal counter per signature"), the predicate $\mathsf{Interp}_{\mathcal{D}}$ is chosen to characterize exactly the witnesses that survive cleaning (as in the vertex-cover core).

D.11.3 Dominance cleaning for optimization cores

Many optimization cores use witnesses of the form (s, m) where s is a finite *structural signature* (e.g. a subset of bag labels) and m is a numeric measure combined monotonically by transitions (e.g. costs added at join). In such cases it is sound to canonize by retaining only Pareto-optimal witnesses per signature.

This argument relies on three simple properties:

- witnesses split into a finite signature part s and a numeric measure m,
- cleaning retains an extremal value of m for each fixed signature s (discarding strictly dominated witnesses),
- transitions implement the corresponding Bellman recurrence for the extremal value function $m_\sigma(s)$.

Definition 63 (Dominance order and Pareto canonization)**.** *Fix a bag B. Let $W(B) = S(B) \times \mathbb{N}$. Define a dominance preorder by*

$$(s, m) \preceq (s', m') \iff s = s' \text{ and } \begin{cases} m \le m' & \text{for minimization cores,} \\ m \ge m' & \text{for maximization cores.} \end{cases}$$

We say that (s, m) is strictly dominated in a set $X \subseteq W(B)$ if there exists $(s, m') \in X$ with $(s, m') \preceq (s, m)$ and $(s, m) \not\preceq (s, m')$ (equivalently: $m' < m$ in the minimization case, or $m' > m$ in the maximization case). Let $\texttt{Clean}^{\text{Pareto}}(X)$ delete all strictly dominated elements, i.e. keep exactly the $\preceq$-minimal representatives for each s (the cost-optimal witness per signature).

Theorem 38 (Soundness of dominance cleaning)**.** *Let $\mathcal{D}$ be a quantitative DP-core whose witnesses are pairs (s, m) with a finite signature s and a numeric measure $m \in \mathbb{N}$, and whose cleaning policy is Pareto canonization as above (keep, for each fixed s, only the cost-optimal m).*

For each term σ and signature s, write

$$m_\sigma(s) := \begin{cases} \min\{m : (s, m) \in \mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\sigma)\}, & \text{for minimization cores,} \\ \max\{m : (s, m) \in \mathrm{Sem}^{\mathrm{pre}}_{\mathcal{D}}(\sigma)\}, & \text{for maximization cores,} \end{cases}$$

where the minimum/maximum is taken over a finite set, with the conventions $\min \emptyset := \infty$ and $\max \emptyset := -\infty$. Assume that for every constructor a the transition rules implement the corresponding Bellman-type recurrence for $m_{a(\vec{\sigma})}$ from the child functions m_{σ_i} (in the minimization case, this is the hypothesis of Lemma 50. The maximization case is analogous). Then $\mathcal{D}$ satisfies (LSC).

Proof. For minimization cores, this is exactly Lemma 50: Pareto canonization coincides with keeping, for each signature s, the unique pair $(s, m_\sigma(s))$ with minimal cost, and the Bellman-recurrence hypothesis implies (LSC).

For maximization cores, the same argument applies with every min replaced by max: for each signature s, Pareto canonization keeps exactly the pair $(s, m_\sigma(s))$ with maximal m, and the assumed Bellman recurrence for these maxima yields (LSC) by the same set-level calculation. □

D.12 Vertex Cover ($\leq t$): Worked LSC Proof

We revisit the vertex-cover DP-core $\mathcal{C}_t^{\mathrm{VC}}$ from Section 4.8.2 and give a detailed instantiation of the proof framework from Section 9.1, proving local soundness and completeness. Because this core also carries an optimization invariant, the section records both the threshold-decision theorem and the exact root invariant value.

D.12.1 Witness space, cleaning, and acceptance

Fix $t, k \in \mathbb{N}$ and a bag $B \subseteq [k{+}1]$.

- A witness at bag $B(\sigma)$ is (S, s) with $S \subseteq B(\sigma)$ and $s \in \mathbb{N}$, where S records which bag vertices are chosen and s counts how many chosen vertices have already been forgotten. Thus the encoded cover size is $s + |S|$.
- $\texttt{Clean}_k$ keeps, for each S, only the pair (S, s) with minimal s. The invariant is $\texttt{Inv}_k(W) = \min\{s + |S| : (S, s) \in W\}$. Finality accepts exactly those witnesses with $s + |S| \leq t$.

This witness representation matches the TreeWidzard implementation: selection decisions are made at introduce-vertex steps, while the counter s is incremented only when a selected vertex is forgotten.

D.12.2 Transitions as set-valued functions

With current bag B (and constructors applied only when the side-conditions of Section 3.4 hold, e.g. endpoints of an `intro_u_v` are in B and both children of a join share

the same bag B):

$$\begin{aligned}
\mathcal{C}_t^{\mathrm{VC}}[k].\texttt{LeafSet} &= \{(\emptyset, 0)\},\\
\mathcal{C}_t^{\mathrm{VC}}[k].\texttt{intro_}u(S,s) &= \{(S,s),\ (S\cup\{u\},s)\},\\
\mathcal{C}_t^{\mathrm{VC}}[k].\texttt{forget_}u(S,s) &= \{(S\setminus\{u\}, s+\mathbf{1}_{u\in S})\},\\
\mathcal{C}_t^{\mathrm{VC}}[k].\texttt{intro_}u\texttt{_}v(S,s) &= \begin{cases}\{(S,s)\}, & u\in S \text{ or } v\in S,\\ \emptyset, & \text{otherwise,}\end{cases}\\
\mathcal{C}_t^{\mathrm{VC}}[k].\texttt{join}\big((S,s),(S',s')\big) &= \begin{cases}\{(S,\ s+s')\}, & S=S',\\ \emptyset, & \text{otherwise.}\end{cases}
\end{aligned}$$

D.12.3 Denotation and the local interpretation predicate

We now spell out the global objects, bag projection, and interpretation predicate used for vertex cover.

Global objects. For a graph G let

$$X_{\mathrm{VC}}(G) := \{\, X \subseteq V(G) : X \text{ is a vertex cover} \,\}.$$

Projection. For a term σ with bag $B(\sigma)$ and labeling $\theta[\sigma] : B(\sigma) \to V(\mathcal{G}(\sigma))$ define the bag projection

$$\pi_{B(\sigma)}(X) := \Big(\ \theta[\sigma]^{-1}(X)\cap B(\sigma),\ \ |X\setminus\theta[\sigma](B(\sigma))|\ \Big) \in W_k^{\mathrm{VC}}.$$

Denotation and interpretation. Set

$$\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\sigma) := \texttt{Clean}_k\big(\{\pi_{B(\sigma)}(X) : X \in X_{\mathrm{VC}}(\mathcal{G}(\sigma))\}\big),$$

and

$$\begin{aligned}
\mathsf{Interp}_{\mathrm{VC}}(\sigma,(S,s)) \iff \exists X \in X_{\mathrm{VC}}(\mathcal{G}(\sigma)) \text{ such that} &\text{(i) } \pi_{B(\sigma)}(X) = (S,s),\\
&\text{(ii) } \forall X' \in X_{\mathrm{VC}}(\mathcal{G}(\sigma)) :\\
&\quad \theta[\sigma]^{-1}(X')\cap B(\sigma) = S\\
&\quad \Rightarrow\ |X'\setminus\theta[\sigma](B(\sigma))| \geq s.
\end{aligned}$$

Then $\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\sigma) = \{(S,s) : \mathsf{Interp}_{\mathrm{VC}}(\sigma,(S,s))\}$ because cleaning removes, for each fixed S, all pairs (S,s') with non-minimal s', and $\mathsf{Interp}_{\mathrm{VC}}$ selects exactly those pairs (S,s) whose s is minimal among covers with bag intersection S. Unpacking the definitions shows that this predicate is precisely the vertex-cover specification fixed at the start of

this section (see Section D.12).

D.12.4 Local law (INT / LSC)

We verify the local soundness and completeness law for vertex cover constructor-wise.

Lemma 52 (VC (LSC))**.** *For each constructor a of the instructive alphabet and all child terms $(\sigma_i)_i$ we have*

$$\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}\big(a(\vec{\sigma})\big) = \mathsf{F}_a\Big(\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\sigma_1), \ldots, \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\sigma_r)\Big).$$

Proof. For a term σ with bag $B(\sigma)$ and a set $S \subseteq B(\sigma)$, define

$$m_\sigma(S) := \min\big\{|X \setminus \theta[\sigma](B(\sigma))| : \ X \in X_{\mathrm{VC}}(\mathcal{G}(\sigma)) \ \wedge \ \theta[\sigma]^{-1}(X) \cap B(\sigma) = S\big\},$$

with $m_\sigma(S) = +\infty$ if no such cover exists. By construction of cleaning,

$$(S, s) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\sigma) \quad \Longleftrightarrow \quad s = m_\sigma(S).$$

We show that the transition rules compute these minima correctly in each constructor.

Leaf. $\mathcal{G}(\texttt{leaf})$ has no edges. The unique feasible signature is $S = \emptyset$ with $m_{\texttt{leaf}}(\emptyset) = 0$, matching $\mathcal{C}_t^{\mathrm{VC}}[k].\texttt{LeafSet}$.

Introduce u**.** Let $\tau = \texttt{intro\_}u(\sigma)$. The new vertex $v := \theta[\tau](u)$ is isolated in $\mathcal{G}(\tau)$. For any cover X of $\mathcal{G}(\sigma)$, both X and $X \cup \{v\}$ are covers of $\mathcal{G}(\tau)$, and every cover of $\mathcal{G}(\tau)$ is of one of these two forms. Hence for $S \subseteq B(\sigma)$,

$$m_\tau(S) = m_\sigma(S) \qquad \text{and} \qquad m_\tau(S \cup \{u\}) = m_\sigma(S),$$

and this is exactly what $\widehat{\texttt{intro\_}u}$ generates from child witnesses (selecting u does not change the outside-count), and cleaning keeps the minimum per signature.

Forget u**.** Let $\tau = \texttt{forget\_}u(\sigma)$. We have $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$ and $B(\tau) = B(\sigma) \setminus \{u\}$. For a fixed $S \subseteq B(\tau)$, any cover of $\mathcal{G}(\sigma)$ with bag intersection S or $S \cup \{u\}$ projects to the same parent signature S. Therefore

$$m_\tau(S) = \min\{\, m_\sigma(S), \ 1 + m_\sigma(S \cup \{u\}) \,\},$$

and this minimum is computed by $\widehat{\texttt{forget\_}u}$ followed by cleaning.

Introduce edge $\{u, v\}$. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$. The new edge must be covered, so a parent witness (S, s) exists exactly when $u \in S$ or $v \in S$. Hence

$$m_\tau(S) = \begin{cases} m_\sigma(S), & u \in S \text{ or } v \in S, \\ \infty, & \text{otherwise,} \end{cases}$$

and this is exactly the filtering performed by $\widehat{\texttt{intro\_}u\texttt{\_}v}$ (the decision to select an endpoint is made at the corresponding introduce-vertex step).

Join. Let $\tau = \texttt{join}(\sigma_1, \sigma_2)$ with common bag $B := B(\tau) = B(\sigma_1) = B(\sigma_2)$ and fix $S \subseteq B$. We claim

$$m_\tau(S) = m_{\sigma_1}(S) + m_{\sigma_2}(S).$$

If one child minimum is $+\infty$, no child cover with signature S exists there, so no parent cover with signature S exists either and the equality is immediate. Assume both child minima are finite. For the upper bound, pick child covers X_i with signature S and outside-count $m_{\sigma_i}(S)$, and set $X := \iota_1(X_1) \cup \iota_2(X_2) \subseteq V(\mathcal{G}(\tau))$. Every parent edge comes from a child, so X is a cover; outside-bag vertices of the two children are disjoint, hence

$$|X \setminus \theta[\tau](B)| = |X_1 \setminus \theta[\sigma_1](B)| + |X_2 \setminus \theta[\sigma_2](B)| = m_{\sigma_1}(S) + m_{\sigma_2}(S),$$

so $m_\tau(S) \leq m_{\sigma_1}(S) + m_{\sigma_2}(S)$. For the lower bound, let X be any parent cover with signature S, and put $X_i := \iota_i^{-1}(X \cap \iota_i(V(\mathcal{G}(\sigma_i))))$. Then each X_i is a child cover with signature S, and by disjointness of child outside-bag vertex sets,

$$|X \setminus \theta[\tau](B)| = |X_1 \setminus \theta[\sigma_1](B)| + |X_2 \setminus \theta[\sigma_2](B)| \geq m_{\sigma_1}(S) + m_{\sigma_2}(S).$$

Taking the minimum over all such X proves the equality. The join transition outputs exactly $(S, s_1 + s_2)$ from compatible children $(S, s_1), (S, s_2)$, and cleaning keeps the minimum per signature.

In every constructor, the cleaned transition F_a computes precisely the minima $m_{a(\vec{\sigma})}(\cdot)$, so (LSC) holds for $\mathcal{C}_t^{\mathrm{VC}}$. □

D.12.5 From local law to global correctness

We now state the resulting global correctness theorems for the vertex-cover core.

Theorem 39 (VC correctness)**.** *For every t, k and term $\tau \in \mathsf{ITD}_k$,*

$$\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau) = \{(S, s) : \mathsf{Interp}_{\mathrm{VC}}(\tau, (S, s))\}.$$

Moreover, $\tau \in \mathsf{Acc}(\mathcal{C}_t^{\mathrm{VC}}[k])$ if and only if $\mathcal{G}(\tau)$ has a vertex cover of size at most t.

Proof. Lemma 52 yields (LSC), so Theorem 17 gives

$$\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau).$$

The displayed equality with $\mathsf{Interp}_{\mathrm{VC}}$ is exactly the definition of $\mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}$ above.

For acceptance, recall $\mathtt{Final}_k(S, s) = 1$ iff $s + |S| \leq t$. Hence

$$\tau \in \mathsf{Acc}(\mathcal{C}_t^{\mathrm{VC}}[k]) \iff \exists (S, s) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau) : \ s + |S| \leq t.$$

If such (S, s) exists, $\mathsf{Interp}_{\mathrm{VC}}$ gives a vertex cover X of $\mathcal{G}(\tau)$ with $\theta[\tau]^{-1}(X) \cap B(\tau) = S$ and $|X \setminus \theta[\tau](B(\tau))| = s$, so

$$|X| = |X \setminus \theta[\tau](B(\tau))| + |\theta[\tau]^{-1}(X) \cap B(\tau)| = s + |S| \leq t.$$

Conversely, let X be a cover of size at most t, set $S := \theta[\tau]^{-1}(X) \cap B(\tau)$ and $s_0 := |X \setminus \theta[\tau](B(\tau))|$. Then $m_\tau(S) \leq s_0$, so $(S, m_\tau(S)) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau)$ and

$$m_\tau(S) + |S| \leq s_0 + |S| = |X| \leq t.$$

Thus τ is accepted. □

Theorem 40 (VC invariant exactness)**.** *For every t, k and term $\tau \in \mathsf{ITD}_k$,*

$$\mathtt{Inv}_k\big(\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau)\big) = \min\{|X| : X \in X_{\mathrm{VC}}(\mathcal{G}(\tau))\}.$$

Proof. By Theorem 39,

$$\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau).$$

Unfolding the invariant and the denotational characterization therefore gives

$$\mathtt{Inv}_k\big(\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau)\big) = \min\{s + |S| : (S, s) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau)\}.$$

For every $(S, s) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau)$, the predicate $\mathsf{Interp}_{\mathrm{VC}}(\tau, (S, s))$ provides a vertex cover X of $\mathcal{G}(\tau)$ with

$$|X| = |X \setminus \theta[\tau](B(\tau))| + |\theta[\tau]^{-1}(X) \cap B(\tau)| = s + |S|.$$

Hence

$$\min\{|X| : X \in X_{\mathrm{VC}}(\mathcal{G}(\tau))\} \leq \mathtt{Inv}_k\big(\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau)\big).$$

Conversely, let X be any vertex cover of $\mathcal{G}(\tau)$, put

$$S := \theta[\tau]^{-1}(X) \cap B(\tau), \qquad s_0 := |X \setminus \theta[\tau](B(\tau))|.$$

Then, by the definition of $m_\tau(S)$ in the proof of Lemma 52, we have $m_\tau(S) \le s_0$, so $(S, m_\tau(S)) \in \mathrm{Sem}_{\mathcal{C}_t^{\mathrm{VC}}}(\tau)$ and therefore

$$\mathtt{Inv}_k\big(\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau)\big) \le m_\tau(S) + |S| \le s_0 + |S| = |X|.$$

Taking the minimum over all covers X yields the reverse inequality. □

Corollary 16 (VC invariant coherence)**.** *Let $\tau \in \mathsf{ITD}_k$ and $\tau' \in \mathsf{ITD}_{k'}$. If*

$$\mathcal{G}(\tau) \simeq \mathcal{G}(\tau'),$$

then

$$\mathtt{Inv}_k\big(\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k](\tau)\big) = \mathtt{Inv}_{k'}\big(\Gamma[\mathcal{C}_t^{\mathrm{VC}}, k'](\tau')\big).$$

Proof. By Theorem 40, both sides are equal to the minimum size of a vertex cover of the realized graph, and that minimum is preserved under graph isomorphism. □

D.12.6 Equivariance and acceptance invariance

Equivariance is standard: relabelings act by

$$\rho \cdot (S, s) := (\rho(S), s),$$

and transitions and cleaning commute with this action.

Proposition 10 (Acceptance invariance for $\mathcal{C}_t^{\mathrm{VC}}$)**.** *Let $\tau \in \mathrm{ITD}_k$ and $\tau' \in \mathrm{ITD}_{k'}$. If $\mathcal{G}(\tau) \cong \mathcal{G}(\tau')$, then*

$$\tau \in \mathsf{Acc}(\mathcal{C}_t^{\mathrm{VC}}[k]) \iff \tau' \in \mathsf{Acc}(\mathcal{C}_t^{\mathrm{VC}}[k']).$$

Proof. By Theorem 39, acceptance is equivalent to existence of a vertex cover of size at most t, which is isomorphism-invariant. □

Remark 21. As in the chromatic case, the decomposition-independent statement for this decision core is acceptance invariance as in Remark 17, not raw witness-set equality with $\mathtt{Inv} = \mathrm{id}$ on root-open terms.

D.12.7 Bitlength, multiplicity, and running time

We now record the witness-complexity bounds that yield the runtime guarantee for vertex cover.

Lemma 53 (Internal polynomiality of $\mathcal{C}_t^{\mathrm{VC}}$)**.** *With the witness encoding (S, s) described above, the core $\mathcal{C}_t^{\mathrm{VC}}$ is internally polynomial.*

Proof. Membership in the witness space is decided by checking that $S \subseteq [k+1]$ and that $s \in \mathbb{N}$. The nullary transition outputs the singleton $\{(\emptyset, 0)\}$. Each constructor rule listed above is computed by inspecting the bitmask of S and the binary encoding of s, so transition time is polynomial in $k + |S| + \log(1+s)$. Cleaning scans a finite witness set, groups witnesses by their subset S, and retains the minimum s in each group, which is polynomial in the total input size. Finality checks the inequality $s + |S| \leq t$, and the invariant computes the minimum value of $s + |S|$ over the witness set. Finally, the invariant output has bitlength $O(\log n)$, which is polynomial in the witness bitlength $O(k + \log n)$. Hence Definition 33 is satisfied. □

Theorem 41 (VC bitlength and multiplicity)**.** *Let B be a bag of size $m := |B| \leq k+1$. Then $\beta_{\mathsf{VC}}(k, n) = O(k + \log n)$ and $\mu_{\mathsf{VC}}(k, n) \leq 2^{k+1}$. Hence*

$$T_{\mathsf{VC}}(k, n) = n \cdot k^{O(1)} \cdot \beta_{\mathsf{VC}}(k, n)^{O(1)} \cdot \mu_{\mathsf{VC}}(k, n)^{O(1)}.$$

Proof. A witness is a pair (S, s) with $S \subseteq B$ and $s \in \mathbb{N}$. Since s counts selected vertices outside the bag, we always have $s \leq |V(\mathcal{G}(\tau))| \leq n$ on an input of size n. Encoding S as a bitmask of length $m \leq k+1$ and s in $O(\log n)$ bits yields $\beta_{\mathsf{VC}}(k, n) = O(k + \log n)$. Cleaning keeps, for each S, only the unique pair with minimal s, so every cleaned witness set contains at most one witness per subset $S \subseteq B$. Hence $\mu_{\mathsf{VC}}(k, n) \leq 2^m \leq 2^{k+1}$. By Lemma 53, Theorem 4 applies to $\mathcal{C}_t^{\mathrm{VC}}$. Specializing to instructive decompositions (arity 2) and substituting the displayed β and μ bounds yields the stated running time expression. □

D.13 Maximum Independent Set

We now instantiate the framework of Section 9.1 for the maximum independent-set core $\mathcal{C}^{\alpha}$ from Section 4.8.2, using independent sets as global objects and the usual bag projection.

D.13.1 Witness space, cleaning, and acceptance

At bag B a witness is a pair (A, a) with $A \subseteq B$ and $a \in \mathbb{N}$. Informally, A is the current intersection with a chosen independent set and a counts selected vertices that have already been forgotten. For each fixed bag B, cleaning keeps, for each $A \subseteq B$, only witnesses (A, a) with maximal a among the set (dominance canonization). The finality predicate accepts every witness and the invariant map returns $\max\{a + |A| : (A, a) \in W\}$.

D.13.2 Transitions as set-valued functions

For self-containment we restate the transition rules here (the catalog anchor is Section 4.8.2). Writing B for the current bag, the transitions are:

$$\begin{aligned}
\mathcal{C}^\alpha[k].\texttt{LeafSet} &= \{(\emptyset, 0)\},\\
\mathcal{C}^\alpha[k].\texttt{intro_}u(A,a) &= \{(A,a)\}\\
&\quad \cup\ \big\{(A\cup\{u\},a) : u \text{ is non-adjacent to every } x\in A\big\},\\
\mathcal{C}^\alpha[k].\texttt{forget_}u(A,a) &= \{(A\setminus\{u\}, a+\mathbf{1}_{u\in A})\},\\
\mathcal{C}^\alpha[k].\texttt{intro_}u\texttt{_}v(A,a) &= \begin{cases}\{(A,a)\}, & |\{u,v\}\cap A|\le 1,\\ \emptyset, & \text{otherwise,}\end{cases}\\
\mathcal{C}^\alpha[k].\texttt{join}\big((A,a),(A',a')\big) &= \begin{cases}\{(A,a+a')\}, & A=A',\\ \emptyset, & \text{otherwise.}\end{cases}
\end{aligned}$$

The set-lifted operators F_a apply these rules pointwise and then clean (max per signature A).

D.13.3 Denotation and the local interpretation predicate

For denotation we use the catalog predicate P^α_k: for $\tau \in \mathsf{ITD}_k$ and (A,a),

$$\mathsf{Interp}_{\mathcal{C}^\alpha}(\tau,(A,a)) \iff \mathsf{P}^\alpha_k(\tau,(A,a)), \qquad \mathrm{Sem}_{\mathcal{C}^\alpha}(\tau) := \{(A,a) : \mathsf{Interp}_{\mathcal{C}^\alpha}(\tau,(A,a))\}.$$

Since cleaning keeps only a maximum counter per signature A, the intended meaning of a witness is canonized: (A,a) represents the best possible value of a among all independent sets with the same boundary intersection A. Equivalently, writing (with the convention $\max\emptyset := -\infty$)

$$m_\tau(A) := \max\{|I\setminus\theta[\tau](B)| : I\subseteq V(\mathcal{G}(\tau)) \text{ independent and } I\cap\theta[\tau](B)=\theta[\tau](A)\},$$

the predicate $\mathsf{Interp}_{\mathcal{C}^\alpha}(\tau,(A,a))$ asserts $a = m_\tau(A)$. This matches the cleaning policy and coincides with P^α_k.

D.13.4 Local law (INT / LSC)

We verify the local law for the independent-set core by checking that transitions realize the intended Bellman recurrences.

Lemma 54 (Independent set (LSC))**.** *For each constructor a of the instructive alphabet*

and all child terms $(\sigma_i)_i$ we have

$$\mathrm{Sem}_{\mathcal{C}^\alpha}\big(a(\vec{\sigma})\big) = \mathsf{F}_a(\mathrm{Sem}_{\mathcal{C}^\alpha}(\sigma_1), \ldots, \mathrm{Sem}_{\mathcal{C}^\alpha}(\sigma_r)) .$$

Proof. For a term σ with bag $B(\sigma)$ and a set $A \subseteq B(\sigma)$, recall $m_\sigma(A)$ as defined above. By construction of the cleaning policy,

$$(A, a) \in \mathrm{Sem}_{\mathcal{C}^\alpha}(\sigma) \quad \Longleftrightarrow \quad a = m_\sigma(A) \neq -\infty.$$

We show that the transitions compute these maxima correctly in each constructor.

Leaf. $\mathcal{G}(\texttt{leaf})$ has no vertices, so the only independent set is $\emptyset$ and the only signature is $A = \emptyset$ with $m_{\texttt{leaf}}(\emptyset) = 0$. This matches the leaf transition.

Introduce vertex. Let $\tau = \texttt{intro\_}v(\sigma)$, so $\mathcal{G}(\tau)$ is obtained from $\mathcal{G}(\sigma)$ by adding an isolated vertex $x := \theta[\tau](v)$ and $B(\tau) = B(\sigma) \cup \{v\}$. For any independent set I of $\mathcal{G}(\sigma)$, both I and $I \cup \{x\}$ are independent in $\mathcal{G}(\tau)$, and every independent set of $\mathcal{G}(\tau)$ is of one of these two forms. Therefore, for each $A \subseteq B(\sigma)$ we have

$$m_\tau(A) = m_\sigma(A) \qquad \text{and} \qquad m_\tau(A \cup \{v\}) = m_\sigma(A).$$

This is exactly the branching implemented by $\widehat{\texttt{intro\_}v}$ followed by keeping, for each signature, the maximal counter.

Introduce edge. Let $\tau = \texttt{intro\_}u\texttt{\_}v(\sigma)$ with $B(\tau) = B(\sigma)$. Introducing the edge $\{u, v\}$ does not change the set of vertices, so the outside count is unchanged. The only new constraint is that u and v cannot both be selected. Hence

$$m_\tau(A) = \begin{cases} m_\sigma(A), & \{u, v\} \not\subseteq A, \\ -\infty, & \{u, v\} \subseteq A. \end{cases}$$

This is precisely what the introduce-edge transition enforces by discarding witnesses with $\{u, v\} \subseteq A$.

Forget vertex. Let $\tau = \texttt{forget\_}v(\sigma)$, so $\mathcal{G}(\tau) = \mathcal{G}(\sigma)$ and $B(\tau) = B(\sigma) \setminus \{v\}$. Fix $A \subseteq B(\tau)$. Any independent set I of $\mathcal{G}(\sigma)$ with $I \cap \theta[\sigma](B(\sigma)) = \theta[\sigma](A)$ remains an independent set for $\mathcal{G}(\tau)$ with the same signature A, and contributes outside count $|I \setminus \theta[\sigma](B(\sigma))|$. If instead I satisfies $I \cap \theta[\sigma](B(\sigma)) = \theta[\sigma](A \cup \{v\})$, then after forgetting v the vertex $\theta[\sigma](v)$ moves outside the bag and contributes one additional outside vertex, so the resulting outside count is $|I \setminus \theta[\sigma](B(\sigma))| + 1$. Therefore

$$m_\tau(A) = \max\{\, m_\sigma(A),\ m_\sigma(A \cup \{v\}) + 1 \,\}.$$

This is exactly computed by $\widehat{\texttt{forget\_}v}$ followed by cleaning (max per signature).

Join. Let $\tau = \texttt{join}(\sigma_1, \sigma_2)$ be a join node with common bag B. Fix $A \subseteq B$. If $I_i \subseteq V(\mathcal{G}(\sigma_i))$ are independent sets with $I_i \cap \theta[\sigma_i](B) = \theta[\sigma_i](A)$, then $I := I_1 \cup I_2$ is independent in the pushout graph because the children share only the bag vertices and there are no edges between non-bag vertices of different children. Moreover, the outside parts of I_1 and I_2 are disjoint, so the outside count adds. Conversely, restricting any independent set $I \subseteq V(\mathcal{G}(\tau))$ to the two children yields independent sets with the same bag intersection. Hence

$$m_\tau(A) = m_{\sigma_1}(A) + m_{\sigma_2}(A).$$

The join transition computes exactly this addition, and cleaning keeps the maximum counter per A.

In every constructor, the cleaned transition F_a computes precisely the maxima $m_{a(\vec{\sigma})}(\cdot)$, so (LSC) holds for $\mathcal{C}^\alpha$. □

D.13.5 From local law to global correctness

Thus (LSC) holds for $\mathcal{C}^\alpha$. Theorem 17 then yields equality.

Theorem 42. *For every term $\tau \in \mathsf{ITD}_k$, we have*

$$\Gamma[\mathcal{C}^\alpha, k](\tau) = \mathrm{Sem}_{\mathcal{C}^\alpha}(\tau),$$

and the invariant value at the root equals $\alpha(\mathcal{G}(\tau))$. Moreover, $\beta = O(k + \log n)$ and $\mu \le 2^{k+1}$.

Proof. Lemma 54 establishes (LSC) for $\mathcal{C}^\alpha$, so Theorem 17 yields $\Gamma[\mathcal{C}^\alpha, k] = \mathrm{Sem}_{\mathcal{C}^\alpha}$.

Fix a term τ with realized graph $G := \mathcal{G}(\tau)$ and root bag $B := B(\tau)$. As in Lemma 54, for each $A \subseteq B$ let $m_\tau(A)$ be the maximum number of chosen vertices outside the bag among independent sets with bag intersection A. Cleaning keeps, for each fixed A, the unique witness $(A, m_\tau(A))$, and the invariant map evaluates

$$\max\{\, m_\tau(A) + |A| : A \subseteq B \,\} = \alpha(G),$$

since every independent set has some bag intersection A and size $|I| = m_\tau(A) + |A|$ for its signature, and conversely each $m_\tau(A) + |A|$ is realized by an independent set.

For encoding, $A \subseteq B$ can be represented by a bitmask of length $m \le k + 1$, and $a \le |V(G)| \le n$ can be encoded in $O(\log n)$ bits, giving $\beta = O(k + \log n)$. As above, cleaning keeps at most one witness per subset $A \subseteq B$, so $\mu \le 2^{|B|} \le 2^{k+1}$. □

D.13.6 Equivariance and coherence

Equivariance is as usual: relabelings act by renaming the subset A and leave the counter a unchanged, and the transitions and cleaning commute with this action. Coherence of the induced invariant follows by Corollary 14.

D.13.7 Bitlength, multiplicity, and running time

We now isolate the multiplicity bound and recall the bitlength bound needed for the generic runtime estimate.

Theorem 43 (Multiplicity of Max Independent Set)**.** *The Independent Set core with dominance cleaning has multiplicity at most* 2^{k+1}*.*

Proof. At a bag B of size $m \leq k+1$, witnesses are of the form (A, a) with $A \subseteq B$. Cleaning keeps, for each fixed A, only one representative with maximal a, so the cleaned witness set contains at most one element per subset $A \subseteq B$. Hence every node stores at most $2^m \leq 2^{k+1}$ witnesses. □

The bitlength bound is recorded in Theorem 42: $\beta = O(k + \log n)$. By Lemma 35, Theorem 4 applies to fixed-term evaluation of $\mathcal{C}^\alpha$. Since instructive decompositions have arity 2, substituting

$$\beta_{\mathrm{IS}}(k, n) = O(k + \log n), \qquad \mu_{\mathrm{IS}}(k, n) \leq 2^{k+1}$$

gives

$$T_{\mathrm{IS}}(k, n) = n \cdot k^{O(1)} \cdot (k + \log n)^{O(1)} \cdot 2^{O(k)}$$

for computing the root witness set and invariant value.

Corollary 17 (Non-finiteness)**.** *The maximum independent-set core* $\mathcal{C}^\alpha$ *is non-finite in the sense of Definition 32.*

Proof. Fix width $k = 0$. For each $n \geq 1$, let τ_n be a root-closed width-0 term whose realized graph is the edgeless n-vertex graph (e.g. by repeating n times the pattern "introduce one vertex and then forget it", reusing the same bag label). Since τ_n is root-closed, $B(\tau_n) = \emptyset$, so every root witness has the form $(\emptyset, a)$. Dominance cleaning keeps the unique maximal counter at this signature, hence

$$\Gamma[\mathcal{C}^\alpha, 0](\tau_n) = \{(\emptyset, a_n)\}.$$

At the root, the invariant value is a_n, and by Theorem 42 it equals $\alpha(\mathcal{G}(\tau_n)) = n$. Thus $a_n = n$.

For each $m \in \{0, 1, \ldots, n\}$, let τ_m be the analogous root-closed width-0 term whose realized graph is the edgeless m-vertex graph. In particular, $|\tau_m| \leq |\tau_n|$. As above, $\Gamma[\mathcal{C}^\alpha, 0](\tau_m) = \{(\emptyset, a_m)\}$ and the root invariant value equals a_m. By Theorem 42 this value equals $\alpha(\mathcal{G}(\tau_m)) = m$, so $a_m = m$. Hence the root witness sets produced from $\tau_0, \tau_1, \ldots, \tau_n$ contain $n + 1$ pairwise distinct witnesses.

Let $t := \lfloor \log_2(n + 1) \rfloor$. There are at most $2^t - 1$ binary strings of length at most $t - 1$. Therefore, among these $n + 1$ distinct root witnesses, at least one has bitlength at least t. Since each of $\tau_0, \ldots, \tau_n$ has size at most $|\tau_n|$, it follows that

$$\beta_{\mathcal{C}^\alpha}(0, |\tau_n|) \geq t.$$

In particular, witness bitlength cannot be bounded by a function of k alone, and $\mathcal{C}^\alpha$ is non-finite. □

D.14 Vertex Counting

We now verify the vertex-counting core $\mathcal{C}^{\text{VCNT}}$ from Section 4.8.2 within the framework of Section 9.1. This core provides a minimal example of an invariant computation with purely numeric witnesses: transitions implement the expected counter recurrence, and the join rule accounts for overlap by subtracting the bag size. The subsections below establish the local law and conclude that the root invariant equals $|V(\mathcal{G}(\tau))|$, together with the trivial witness-complexity bounds.

D.14.1 Witness space, cleaning, and acceptance

At bag B a witness is a single integer $n \in \mathbb{N}$. Cleaning is the identity, the finality predicate accepts every witness, and the invariant returns the root counter value.

D.14.2 Transitions as set-valued functions

Transitions implement the obvious recurrence: at an introduce-vertex node n is incremented by 1, forget-vertex and introduce-edge nodes leave n unchanged, and at a join we subtract the shared bag size $|B|$ to compensate for double-counting.

D.14.3 Denotation and the local interpretation predicate

For denotation we reuse the catalog predicate $\mathsf{P}_k^{\text{VCNT}}$: for $\tau \in \mathsf{ITD}_k$ and $n \in \mathbb{N}$,

$$\mathsf{Interp}_{\mathcal{C}^{\text{VCNT}}}(\tau, n) \iff \mathsf{P}_k^{\text{VCNT}}(\tau, n), \qquad \text{Sem}_{\mathcal{C}^{\text{VCNT}}}(\tau) := \{n : \mathsf{Interp}_{\mathcal{C}^{\text{VCNT}}}(\tau, n)\}.$$

By definition of $\mathsf{P}_k^{\text{VCNT}}$ this says that n is the number of vertices of $\mathcal{G}(\tau)$.

D.14.4 Local law (INT / LSC)

We establish the constructor-wise interpretation law for the vertex-counting core.

Lemma 55 (Vertex counting INT)**.** *For each constructor a and children $(\sigma_i)_i$,*

$$\mathsf{Interp}_{\mathcal{C}^{\mathrm{VCnt}}}\big(a(\vec{\sigma}), n\big) \iff \exists \vec{n} \, \Big(\bigwedge_i \mathsf{Interp}_{\mathcal{C}^{\mathrm{VCnt}}}(\sigma_i, n_i) \ \wedge \ n \in \widehat{a}(\vec{n}) \Big).$$

Proof. We verify the equivalence by cases on the constructor a.

Leaf. $\mathcal{G}(\texttt{leaf})$ has no vertices, so $\mathsf{Interp}(\texttt{leaf}, 0)$ holds and no other n is admissible. This matches the leaf transition returning $\{0\}$.

Introduce vertex. Let $\tau = \texttt{intro\_}v(\sigma)$. Then $\mathcal{G}(\tau)$ is obtained from $\mathcal{G}(\sigma)$ by adding one isolated vertex, hence $|V(\mathcal{G}(\tau))| = |V(\mathcal{G}(\sigma))| + 1$. The introduce transition implements exactly $n \mapsto n + 1$, so (INT) holds.

Forget-vertex / introduce-edge. Both constructors do not change the realized vertex set, so the unique correct count is preserved. The corresponding transitions leave n unchanged, establishing (INT).

Join. Let $\tau = \texttt{join}(\tau_1, \tau_2)$ with common bag B. By pushout semantics, $V(\mathcal{G}(\tau)) = V(\mathcal{G}(\tau_1)) \cup V(\mathcal{G}(\tau_2))$ and $V(\mathcal{G}(\tau_1)) \cap V(\mathcal{G}(\tau_2)) = \theta[\tau](B)$, hence $|V(\mathcal{G}(\tau))| = |V(\mathcal{G}(\tau_1))| + |V(\mathcal{G}(\tau_2))| - |B|$. This is exactly the join transition $n_1 \otimes_B n_2 := \{n_1 + n_2 - |B|\}$, so (INT) holds. □

Thus (LSC) holds for $\mathcal{C}^{\mathrm{VCnt}}$, so Theorem 17 yields $\Gamma[\mathcal{C}^{\mathrm{VCnt}}, k] = \mathrm{Sem}_{\mathcal{C}^{\mathrm{VCnt}}}$.

D.14.5 From local law to global correctness

We now state the resulting correctness theorem and the corresponding witness-complexity bounds.

Theorem 44. *For every term $\tau \in \mathsf{ITD}_k$, we have*

$$\Gamma[\mathcal{C}^{\mathrm{VCnt}}, k](\tau) = \mathrm{Sem}_{\mathcal{C}^{\mathrm{VCnt}}}(\tau),$$

and the root value equals $|V(\mathcal{G}(\tau))|$. Moreover, $\beta = O(\log n)$ and $\mu = 1$.

Proof. Lemma 55 establishes (INT) for every constructor of $\mathcal{C}^{\mathrm{VCnt}}$. Since cleaning is the identity, (LSC) follows and Theorem 17 yields $\Gamma[\mathcal{C}^{\mathrm{VCnt}}, k] = \mathrm{Sem}_{\mathcal{C}^{\mathrm{VCnt}}}$. Fix a term τ with realized graph $G := \mathcal{G}(\tau)$. By definition of $\mathsf{P}_k^{\mathrm{VCnt}}$, the unique root witness value equals $|V(G)|$, hence the root value is $|V(G)|$.

At a term of size n, the vertex count is at most n, so encoding the integer witness takes $O(\log n)$ bits. All transitions are functional (the counter updates deterministically), hence every node carries a singleton witness set and $\mu = 1$. □

D.14.6 Equivariance and coherence

Equivariance is immediate: relabelings do not affect the numeric count and all transitions are bag-symmetric. Coherence follows by Corollary 14.

D.14.7 Bitlength, multiplicity, and running time

The bitlength and multiplicity bounds are recorded in Theorem 44: $\beta = O(\log n)$ and $\mu = 1$. By Lemma 35, Theorem 4 applies. Hence

$$T_{\mathrm{VCnt}}(k, n) = n \cdot k^{O(1)} \cdot (\log n)^{O(1)}$$

by Theorem 4.

Corollary 18 (Non-finiteness)**.** *The vertex-counting core $\mathcal{C}^{\mathrm{VCnt}}$ is non-finite in the sense of Definition 32.*

Proof. Fix width $k = 0$. For each $n \geq 1$, let τ_n be a root-closed width-0 term whose realized graph has n vertices. For each $m \in \{0, 1, \ldots, n\}$, let τ_m be the analogous root-closed width-0 term whose realized graph has m vertices. In particular, $|\tau_m| \leq |\tau_n|$.

By Theorem 44, for every $m \in \{0, 1, \ldots, n\}$ we have

$$\Gamma[\mathcal{C}^{\mathrm{VCnt}}, 0](\tau_m) = \{m\}.$$

Hence the root witness sets produced from $\tau_0, \tau_1, \ldots, \tau_n$ contain $n + 1$ pairwise distinct witnesses.

Let $t := \lfloor \log_2(n + 1) \rfloor$. There are at most $2^t - 1$ binary strings of length at most $t - 1$. Therefore, among these $n + 1$ distinct root witnesses, at least one has bitlength at least t. Since each of $\tau_0, \ldots, \tau_n$ has size at most $|\tau_n|$, it follows that

$$\beta_{\mathcal{C}^{\mathrm{VCnt}}}(0, |\tau_n|) \geq t.$$

Thus witness bitlength cannot be bounded by a function of k alone, and $\mathcal{C}^{\mathrm{VCnt}}$ is non-finite. □

D.15 Boundary Myhill and Nerode equivalence and lower bounds

To establish lower bounds on the number of deterministic states induced by DP-cores, we appeal to the Myhill and Nerode theory for graphs of bounded width.

Definition 64 (Boundary equivalence)**.** *Let $\mathcal{P}$ be a graph property. Two B-boundaried graphs G_1, G_2 are boundary equivalent with respect to $\mathcal{P}$, written $G_1 \equiv_B^{\mathcal{P}} G_2$, if for every context graph H (a B-boundaried graph),*

$$G_1 \oplus_B H \in \mathcal{P} \iff G_2 \oplus_B H \in \mathcal{P}.$$

The unrestricted relation $\equiv_B^{\mathcal{P}}$ is useful for intuition, but in our setting we only ever encounter contexts that are realizable within the same width bound. We therefore also use the following restricted variant.

Definition 65 (Restricted boundary equivalence (width k))**.** *Fix $k \in \mathbb{N}$ and a bag $B \subseteq [k{+}1]$. Let $\mathcal{C}_{k,B}$ be the class of B-boundaried graphs realizable by some width-k term whose top bag is B. Equivalently, $\mathcal{C}_{k,B}$ is the class of B-contexts realizable within the width-k term model.*

For a graph property $\mathcal{P}$ and $G_1, G_2 \in \mathcal{C}_{k,B}$ we write

$$G_1 \equiv_{B,k}^{\mathcal{P}} G_2$$

if for every context $H \in \mathcal{C}_{k,B}$ we have

$$G_1 \oplus_B H \in \mathcal{P} \iff G_2 \oplus_B H \in \mathcal{P}.$$

Proposition 11 (Myhill and Nerode lower bound for DP-cores (deterministic states))**.** *Fix k and a bag $B \subseteq [k{+}1]$, and let $\mathcal{C}_{k,B}$ and $\equiv_{B,k}^{\mathcal{P}}$ be as in Definition 65.*

Let $\mathcal{D}[k]$ be a DP-core correct for the graph property $\mathcal{P}$. Write $W_k(B) \subseteq W_k$ for the set of witnesses whose support/bag is B, and set

$$Q(B) \;:=\; \mathrm{Cl}(W_k(B)) \quad \textit{(canonical witness-set states over bag } B\textit{)}.$$

Fix a choice function $G \mapsto \tau_G$ selecting, for each $G \in \mathcal{C}_{k,B}$, a width-k term τ_G with top bag B that realizes G. Define the induced (deterministic) state map

$$s_{\mathcal{D}[k]}(G) \;:=\; \Gamma[\mathcal{D}, k](\tau_G) \;\in\; Q(B).$$

Then $s_{\mathcal{D}[k]}$ refines restricted boundary equivalence: if $s_{\mathcal{D}[k]}(G_1) = s_{\mathcal{D}[k]}(G_2)$, then $G_1 \equiv_{B,k}^{\mathcal{P}}$

G_2. *Consequently,*

$$\left|\mathrm{Im}(s_{\mathcal{D}[k]})\right| \;\geq\; \#\big(\mathcal{C}_{k,B}/\equiv^{\mathcal{P}}_{B,k}\big),$$

i.e. the number of distinct reachable canonical witness-set states over bag B is at least the number of $\equiv^{\mathcal{P}}_{B,k}$-classes.

Remark 22. If $\mathcal{D}[k]$ is coherent on $\mathcal{C}_{k,B}$-terms, then $s_{\mathcal{D}[k]}$ is independent of the choice function $G \mapsto \tau_G$.

Proof. Assume $s_{\mathcal{D}[k]}(G_1) = s_{\mathcal{D}[k]}(G_2) =: S$ and let $H \in \mathcal{C}_{k,B}$ be any context. Pick a width-k term η_H with top bag B that realizes H.

For $i \in \{1,2\}$ consider the join term $\mathtt{join}(\tau_{G_i}, \eta_H)$. By the semantics of join nodes for instructive decompositions, its realized boundaried graph is (canonically isomorphic to) the pushout $G_i \oplus_B H$. Now wrap this join term at the root with a chain of forget constructors for all labels in B, yielding a root-closed term (empty top bag) that still realizes the (unboundaried) graph $G_i \oplus_B H$.

Now evaluate $\mathcal{D}[k]$ on these two root-closed terms. By Proposition 7, evaluation is deterministic on canonical witness-set states: the state at the join node depends only on the two child states, and the subsequent forget sequence applies a fixed deterministic chain of forget transitions at the root. Since the left child states coincide (both equal S by assumption) and the right child state is the same ($\Gamma[\mathcal{D}, k](\eta_H)$), the two root-closed join terms yield the same root state and therefore the same acceptance.

Finally, $\mathcal{D}[k]$ is correct for $\mathcal{P}$, so acceptance coincides with membership in $\mathcal{P}$ on the realized graph. Hence $G_1 \oplus_B H \in \mathcal{P} \iff G_2 \oplus_B H \in \mathcal{P}$ for all $H \in \mathcal{C}_{k,B}$, i.e. $G_1 \equiv^{\mathcal{P}}_{B,k} G_2$. The lower bound on $|\mathrm{Im}(s_{\mathcal{D}[k]})|$ follows immediately. □

Lemma 56 (HP necessitates path matchings)**.** *Fix $k \in \mathbb{N}$ and a bag $B \subseteq [k{+}1]$. Write $m := |B|$ and let $\mathcal{M}(B)$ be the set of partial matchings on B (so $|\mathcal{M}(B)| = I(m)$). For $M \in \mathcal{M}(B)$ define a B-boundaried graph G_M as follows:*

- *if $m \leq 2$, take G_M to be the graph on vertex set B with edge set M.*
- *if $m \geq 3$ (so $k \geq 2$), take G_M to be the graph obtained from M by subdividing every matching edge $\{u, v\} \in M$ with a fresh vertex $x_{u,v}$, i.e. replace $\{u, v\}$ by the length-2 path $u - x_{u,v} - v$.*

Then $G_M \in \mathcal{C}_{k,B}$ for all $M \in \mathcal{M}(B)$, and for any two distinct matchings $M_1 \neq M_2$ we have

$$G_{M_1} \not\equiv^{\mathrm{HP}}_{B,k} G_{M_2}.$$

Proof. The membership $G_M \in \mathcal{C}_{k,B}$ is immediate: for $m \leq 2$ we have G_M on at most two boundary vertices, and for $m \geq 3$ both G_M and the witness contexts constructed below

have treewidth at most $m-1 \le k$ with root bag B (hence admit width-k instructive decompositions).

Fix two matchings $M_1 \neq M_2$. If $m \le 2$, then $|\mathcal{M}(B)| \le 2$ and the claim can be checked directly: for $M_1 = \emptyset$ and $M_2 = \{\{u, v\}\}$, taking the empty context H gives $G_{M_2} \oplus_B H = G_{M_2}$ (which has a Hamiltonian path) while $G_{M_1} \oplus_B H$ is disconnected (hence has none). We therefore assume $m \ge 3$ (so every G_M uses subdividing vertices).

We distinguish two cases.

Case 1: a vertex is unmatched in exactly one matching. Assume there exists $w \in B$ that is unmatched in M_1 but matched in M_2. We build a context $H \in \mathcal{C}_{k,B}$ such that $G_{M_1} \oplus_B H$ has a Hamiltonian path but $G_{M_2} \oplus_B H$ does not.

Let $\mathcal{C}_1, \ldots, \mathcal{C}_t$ be the connected components of the matching graph (B, M_1), so each $\mathcal{C}_i$ is either an isolated vertex or a matching edge. Fix any linear order of these components. For each $\mathcal{C}_i$ choose two (not necessarily distinct) boundary vertices $p_i, q_i \in V(\mathcal{C}_i)$ as follows:

- if $\mathcal{C}_i = \{v\}$ is isolated, set $p_i = q_i := v$.
- if $\mathcal{C}_i$ is an edge component $\{a, b\} \in M_1$, set $(p_i, q_i) := (a, b)$ or $(p_i, q_i) := (b, a)$.

Introduce fresh vertices $y_0, \ldots, y_t$ and fresh endpoints ℓ, r, and define H to have the edges

$$\ell - y_0 - p_1, \qquad q_i - y_i - p_{i+1} \ \ (1 \le i < t), \qquad q_t - y_t - r.$$

We view H as a B-boundaried graph by taking the boundary map to be the identity on B.

By construction, in $G_{M_1} \oplus_B H$ every vertex $x_{u,v}$ subdividing an M_1-edge has degree 2, every connector vertex y_i has degree 2, and every boundary vertex has degree 2 except the endpoints ℓ, r which have degree 1. Hence $G_{M_1} \oplus_B H$ is a single path (a subdivision of the connector chain), and therefore has a Hamiltonian path.

In $G_{M_2} \oplus_B H$, the vertex w is incident to three distinct degree-2 vertices: it is adjacent to two connector vertices (because w has degree 2 in H) and to one subdivider $x_{w,z}$ (since w is matched in M_2). Moreover, ℓ and r are the only degree-1 vertices of $G_{M_2} \oplus_B H$, so any Hamiltonian path would have ℓ and r as its two endpoints. Consequently none of the three degree-2 neighbors of w can be an endpoint of a Hamiltonian path. Every internal degree-2 vertex of a Hamiltonian path uses both of its incident edges, so the two connector vertices and the subdivider $x_{w,z}$ would each force their edge to w into the path. This would make w incident to three Hamiltonian-path edges, impossible. Thus $G_{M_2} \oplus_B H$ has no Hamiltonian path. This shows $G_{M_1} \not\equiv^{\mathrm{HP}}_{B,k} G_{M_2}$ in this case.

Case 2: matched vertex sets agree (different pairings). Otherwise, every boundary vertex is matched in M_1 iff it is matched in M_2. Since $M_1 \neq M_2$, pick an edge $\{u, v\} \in M_1 \setminus M_2$. Then in M_2 the vertices u and v are matched to *other* vertices (in particular, u and v are distinct and not paired together).

We now build a context $H \in \mathcal{C}_{k,B}$ that turns $G_{M_2} \oplus_B H$ into a single Hamiltonian path with endpoints u and v, while $G_{M_1} \oplus_B H$ is disconnected. Let $\mathcal{D}_1, \ldots, \mathcal{D}_t$ be the connected components of (B, M_2). Choose an order and orientations (p_i, q_i) as above such that $p_1 = u$ and $q_t = v$, and introduce fresh vertices $y_1, \ldots, y_{t-1}$. Let H have the edges $q_i - y_i - p_{i+1}$ for $1 \leq i < t$. Then $G_{M_2} \oplus_B H$ is again a subdivision of this connector path, hence has a Hamiltonian path.

In $G_{M_1} \oplus_B H$, the vertices u and v have no incident edges from H by construction. Since $\{u, v\} \in M_1$, the subdivider vertex $x_{u,v}$ exists and the subgraph induced by $\{u, v, x_{u,v}\}$ is a connected component disjoint from the rest of the glued graph. Therefore $G_{M_1} \oplus_B H$ is disconnected, so it has no Hamiltonian path.

In both cases we find a context in $\mathcal{C}_{k,B}$ separating G_{M_1} and G_{M_2}, proving $G_{M_1} \not\equiv^{\mathrm{HP}}_{B,k} G_{M_2}$. □

Corollary 19 (HP lower bound (deterministic states))**.** *Fix k and a bag $B \subseteq [k{+}1]$ and write $m := |B|$. Let $\mathcal{D}[k]$ be any DP-core correct for Hamiltonian Path, and let $s_{\mathcal{D}[k]} : \mathcal{C}_{k,B} \to \mathrm{Cl}(W_k(B))$ be the induced state map from Proposition 11. Then*

$$\left|\mathrm{Im}(s_{\mathcal{D}[k]})\right| \;\geq\; I(m),$$

where $I(m)$ is the number of partial matchings on an m-element set.

Proof. By Lemma 56, there are $I(m)$ pairwise non-$\equiv^{\mathrm{HP}}_{B,k}$-equivalent graphs in $\mathcal{C}_{k,B}$, so the index $\#(\mathcal{C}_{k,B}/\equiv^{\mathrm{HP}}_{B,k})$ is at least $I(m)$. Proposition 11 then yields $|\mathrm{Im}(s_{\mathcal{D}[k]})| \geq \#(\mathcal{C}_{k,B}/\equiv^{\mathrm{HP}}_{B,k}) \geq I(m)$. □

Appendix E

SAT Encodings

This appendix records the SAT encodings for the implemented catalog, together with the reusable cardinality gadget on which several of them depend. Vertex counting is not included here, because it is handled by direct node counting rather than by a CNF encoding. Unless stated otherwise, the encodings are for finite, undirected, loopless simple graphs. Multiple Edge Detection is the only multigraph case. Correctness is understood as satisfiability equivalence, or equisatisfiability when auxiliary variables are introduced.

E.1 Boolean Logic and SAT Formulations

We use standard SAT notation. A literal is a variable or its negation, a clause is a disjunction of literals, and a CNF formula is a conjunction of clauses. All formulas below are written directly in CNF. When auxiliary variables are introduced, correctness is understood equisatisfiably: a graph certificate can be extended to satisfy the auxiliary clauses, and every satisfying assignment decodes to a valid certificate for the original graph instance.

E.2 Encoding Non-CNF Constraints in SAT

While many graph problems can be expressed directly in propositional logic using basic adjacency and selection constraints, others involve higher-level arithmetic constraints, such as bounding the number of true variables in a subset. Such cardinality constraints are not naturally in CNF, and thus require transformation before a SAT solver can process them. In this section, we focus on techniques to encode these constraints, with particular attention to the sequential counter encoding [Sinz, 2005], which is our primary method.

Sequential Counter Encoding. Let $x_1, \ldots, x_n$ be Boolean variables and $t \in \mathbb{N}$. A common constraint in graph problems is to enforce that at most t of these variables are assigned the value true, i.e.,

$$\sum_{i=1}^{n} x_i \leq t.$$

A naive way to encode this is to add, for every $(t+1)$-subset $\{i_1, \ldots, i_{t+1}\} \subseteq \{1, \ldots, n\}$, the clause $\neg x_{i_1} \vee \cdots \vee \neg x_{i_{t+1}}$, forbidding all $t+1$ variables from being simultaneously true. However, this approach results in a combinatorial explosion of clauses, with $\binom{n}{t+1}$ clauses of length $t+1$ in the worst case. To avoid this, we use a more scalable encoding due to Sinz [Sinz, 2005], based on a unary sequential counter.

The idea is to introduce auxiliary variables $s_{i,j}$ that represent a unary counter over prefixes of the variable sequence. We use the standard sequential-counter encoding due to Sinz [2005]. Intuitively, $s_{i,j}$ means that among the variables $x_1, \ldots, x_i$, at least j are true (i.e., the running count has reached level j).

Construction. Formally, to encode the constraint $\sum_{i=1}^{n} x_i \leq t$, we introduce auxiliary variables $s_{i,j}$ for $1 \leq i \leq n-1$ and $1 \leq j \leq t$, and add the following clause families: *(For $t = 0$, the constraint is equivalent to $\bigwedge_{i=1}^{n} \neg x_i$. We therefore assume $t \geq 1$ below.)*

- Base: $\neg x_i \vee s_{i,1}$ for $1 \leq i \leq n-1$.
- Monotonicity: $\neg s_{i-1,j} \vee s_{i,j}$ for $2 \leq i \leq n-1$, $1 \leq j \leq t$.
- Increment: $\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}$ for $2 \leq i \leq n-1$, $2 \leq j \leq t$.
- Overflow prevention: $\neg x_i \vee \neg s_{i-1,t}$ for $2 \leq i \leq n$.

The overflow-prevention clauses ensure that no assignment can set more than t of the x_i to true: once the prefix counter has reached t, every subsequent x_i is forced to false.

Because the encoding introduces auxiliary variables, correctness is understood as an equisatisfiability statement: an assignment to the original variables satisfies $\sum_{i=1}^{n} x_i \leq t$ if and only if it can be extended to satisfy the CNF gadget $\mathrm{SC}(x_1, \ldots, x_n, t)$.

Lemma 57 (Sequential counter correctness)**.** *Assume $t \geq 1$ and let $\mathrm{SC}(x_1, \ldots, x_n, t)$ be the CNF formula consisting of the clause families above. Then:*

1. *$\mathrm{SC}(x_1, \ldots, x_n, t)$ contains $\mathcal{O}(nt)$ auxiliary variables and $\mathcal{O}(nt)$ clauses.*
2. *For every assignment α to $x_1, \ldots, x_n$, $\sum_{i=1}^{n} \alpha(x_i) \leq t$ if and only if α has an extension $\widehat{\alpha}$ to the auxiliary variables such that*

$$\widehat{\alpha} \models \mathrm{SC}(x_1, \ldots, x_n, t).$$

Proof. **(1)** There are $(n-1)\cdot t = \mathcal{O}(nt)$ auxiliary variables $s_{i,j}$. Each clause family contributes $\mathcal{O}(nt)$ clauses of constant length, hence the overall size is $\mathcal{O}(nt)$ clauses.

(2) Completeness. Let α be an assignment to $x_1, \dots, x_n$ such that $\sum_{i=1}^{n} \alpha(x_i) \le t$. Extend α to the $s_{i,j}$ by setting, for all $1 \le i \le n-1$ and $1 \le j \le t$,

$$\alpha(s_{i,j}) = 1 \quad \text{if and only if} \quad \sum_{k=1}^{i} \alpha(x_k) \ge j.$$

We verify that every clause is satisfied:

- *Base.* If $\alpha(x_i) = 0$, then $\neg x_i \vee s_{i,1}$ is true. If $\alpha(x_i) = 1$, then $\sum_{k=1}^{i} \alpha(x_k) \ge 1$, so $\alpha(s_{i,1}) = 1$.
- *Monotonicity.* If $\alpha(s_{i-1,j}) = 1$, then $\sum_{k=1}^{i-1} \alpha(x_k) \ge j$, hence also $\sum_{k=1}^{i} \alpha(x_k) \ge j$, so $\alpha(s_{i,j}) = 1$.
- *Increment.* If $\alpha(x_i) = 1$ and $\alpha(s_{i-1,j-1}) = 1$, then $\sum_{k=1}^{i-1} \alpha(x_k) \ge j-1$, so $\sum_{k=1}^{i} \alpha(x_k) \ge j$ and therefore $\alpha(s_{i,j}) = 1$.
- *Overflow prevention.* If $\alpha(x_i) = 1$, then

$$\sum_{k=1}^{i-1} \alpha(x_k) = \sum_{k=1}^{i} \alpha(x_k) - 1 \le \sum_{k=1}^{n} \alpha(x_k) - 1 \le t - 1,$$

  so by the definition of $\alpha(s_{i-1,t})$ we get $\alpha(s_{i-1,t}) = 0$. Thus $\neg x_i \vee \neg s_{i-1,t}$ is satisfied.

So $\alpha \models \mathrm{SC}(x_1, \dots, x_n, t)$.

(2) Soundness. Let α be a satisfying assignment of $\mathrm{SC}(x_1, \dots, x_n, t)$, and define prefix sums

$$p(i) := \sum_{k=1}^{i} \alpha(x_k).$$

Claim. For all $i \in \{1, \dots, n-1\}$ and $j \in \{1, \dots, t\}$, if $p(i) \ge j$ then $\alpha(s_{i,j}) = 1$. *Proof of claim.* We induct on i. For $i = 1$, $p(1) \ge 1$ implies $\alpha(x_1) = 1$, and the base clause $\neg x_1 \vee s_{1,1}$ forces $\alpha(s_{1,1}) = 1$. For the inductive step, assume the claim holds for $i-1$. If $p(i-1) \ge j$ then by induction $\alpha(s_{i-1,j}) = 1$, and monotonicity $\neg s_{i-1,j} \vee s_{i,j}$ yields $\alpha(s_{i,j}) = 1$. Otherwise $p(i) \ge j$ but $p(i-1) < j$, so necessarily $\alpha(x_i) = 1$ and $p(i-1) \ge j-1$. If $j = 1$ then the base clause gives $\alpha(s_{i,1}) = 1$. If $j \ge 2$, by induction $\alpha(s_{i-1,j-1}) = 1$ and the increment clause $\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}$ yields $\alpha(s_{i,j}) = 1$. This proves the claim.

Now suppose for contradiction that $\sum_{i=1}^{n} \alpha(x_i) \ge t+1$. Let $i^\star$ be the smallest index with $p(i^\star) \ge t+1$. Then $\alpha(x_{i^\star}) = 1$ and $p(i^\star - 1) \ge t$, so by the claim $\alpha(s_{i^\star-1,t}) = 1$. But

then the overflow clause $\neg x_{i^\star} \vee \neg s_{i^\star-1,t}$ is violated, contradicting satisfiability. Therefore $\sum_{i=1}^{n} \alpha(x_i) \leq t$. □

Corollary 20 (Lower bounds via negation)**.** *For Boolean variables $x_1, \ldots, x_n$ and an integer t,*

$$\sum_{i=1}^{n} x_i \geq t \quad \Longleftrightarrow \quad \sum_{i=1}^{n} \neg x_i \leq n - t.$$

Proof. Let $k := \sum_{i=1}^{n} x_i$. Then $\sum_{i=1}^{n} \neg x_i = n-k$, so $k \geq t$ if and only if $n-k \leq n-t$. □

The sequential counter encoding is widely used in SAT-based encodings of optimization and graph problems due to its compactness and strong unit propagation behavior, which makes it particularly effective in CDCL solvers [Sinz, 2005].

Handling Lower Bounds. While the sequential counter directly encodes upper bounds like $\sum x_i \leq t$, it can also be used for lower bounds of the form $\sum x_i \geq t$ by rewriting them as upper bounds on negated variables:

$$\sum_{i=1}^{n} x_i \geq t \quad \Longleftrightarrow \quad \sum_{i=1}^{n} \neg x_i \leq n - t.$$

In this form, the lower bound becomes an upper bound on the number of variables not selected. This duality allows us to reuse the sequential counter encoding by applying it to the negated variables $\neg x_i$. When $n - t \geq 1$, the resulting CNF formula enforces the bound $\sum_{i=1}^{n} \neg x_i \leq n - t$ *under extension to the auxiliary variables* (Lemma 57), i.e., at least t of the x_i are assigned true. When $n - t = 0$, the lower bound is simply $\bigwedge_{i=1}^{n} x_i$.

Formally, for $0 \leq t \leq n$, we encode $\sum_{i=1}^{n} x_i \geq t$ by:

- using the constant CNF $\top$ when $t = 0$;
- using the unit clauses $\bigwedge_{i=1}^{n} x_i$ when $t = n$;
- otherwise applying Lemma 57 to the literals $\neg x_1, \ldots, \neg x_n$ with bound $n - t$ (equivalently, by taking $\mathrm{SC}(y_1, \ldots, y_n, n-t)$ and substituting $y_i := \neg x_i$, simplifying double negations).

This method of translating lower bounds into upper bounds ensures that the encoding remains uniform and compatible with the same auxiliary structure and solver heuristics. Throughout the thesis, lower-bound cardinality constraints are typically handled in this way. Trivial cases such as $\sum x_i \geq 1$ are encoded directly as a single clause.

Tseitin Transformation. Another foundational method for CNF transformation is the Tseitin transformation [Tseitin, 1983], which is used to translate arbitrary Boolean formulas into equisatisfiable CNF formulas. This transformation introduces auxiliary variables for each internal gate in the formula and adds clauses that encode the gate's truth table. While the Tseitin transformation is critical in compiler-based SAT tools and formal logic pipelines, it is not employed in our main encodings. Nonetheless, it provides essential theoretical background and guarantees that any propositional formula can be translated into an equisatisfiable CNF formula in time linear in its size.

E.3 Encoding Graph Problems into SAT

This section records the CNF encodings used in this appendix. Each subsection states the decision problem, specifies the variable set and clause families, proves correctness, and records the asymptotic size of the encoding. When a cardinality constraint is needed, we use the sequential-counter gadget from Section E.2 unless stated otherwise.

Unless stated otherwise, inputs are finite, undirected, loopless simple graphs. Parameter values outside their natural range are handled by trivial constant CNFs and are omitted from the presentation.

E.3.1 Vertex Cover

We record a CNF encoding for the decision problem whether G has a vertex cover of size at most t.

Definition 66 (Vertex Cover)**.** *Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices and $m = |E|$ edges, and let $t \in \mathbb{N}$. A vertex cover is a subset $C \subseteq V$ such that every edge $\{u, v\} \in E$ has at least one endpoint in C. The decision problem asks whether G has a vertex cover of size $|C| \leq t$.*

SAT Formula Construction

We construct a CNF formula $\phi_{\mathrm{VC}}(G, t)$ whose satisfying assignments encode vertex covers of size at most t.

Variables. Introduce one Boolean variable x_v for each vertex $v \in V$, where $x_v = 1$ means that v is selected into the cover. For an enumeration $V = \{v_1, \ldots, v_n\}$ we also write $x_i := x_{v_i}$.

Edge coverage. For every edge $\{u, v\} \in E$, at least one of its endpoints must be in the vertex cover. This requirement is encoded as a disjunctive clause:

$$(x_u \vee x_v).$$

Budget. To ensure the size of the vertex cover does not exceed t, we enforce the cardinality constraint

$$\sum_{i=1}^{n} x_i \leq t.$$

If $t = 0$, this is equivalent to $\bigwedge_{v \in V} \neg x_v$. Otherwise, we encode the upper bound using the sequential counter gadget from Section E.2 (Lemma 57), introducing $\mathcal{O}(nt)$ auxiliary variables and $\mathcal{O}(nt)$ clauses.

Lemma 58 (Correctness of the Vertex Cover encoding)**.** *The formula $\phi_{\mathrm{VC}}(G, t)$ is satisfiable if and only if G has a vertex cover of size at most t.*

Proof. **Completeness.** Let $C \subseteq V$ be a vertex cover with $|C| \leq t$. Define an assignment α by $\alpha(x_v) = 1$ iff $v \in C$. Every edge $\{u, v\} \in E$ has an endpoint in C, hence the clause $(x_u \vee x_v)$ is satisfied. Moreover, $\sum_{v \in V} \alpha(x_v) = |C| \leq t$, so the budget constraint holds. For $t \geq 1$, Lemma 57 yields an extension of α to the auxiliary variables satisfying the sequential-counter clauses. Thus $\alpha \models \phi_{\mathrm{VC}}(G, t)$.

Soundness. Let $\alpha \models \phi_{\mathrm{VC}}(G, t)$ and define $C := \{v \in V \mid \alpha(x_v) = 1\}$. For every edge $\{u, v\} \in E$, the clause $(x_u \vee x_v)$ is satisfied, so $u \in C$ or $v \in C$. Hence C is a vertex cover. Finally, the budget part enforces $\sum_{v \in V} \alpha(x_v) \leq t$ (trivially if $t = 0$, and by Lemma 57 otherwise), so $|C| \leq t$. □

Complexity. The encoding uses n primary variables and $\mathcal{O}(nt)$ auxiliary variables for the sequential counter. The edge coverage constraint contributes m binary clauses, and the sequential counter contributes $\mathcal{O}(nt)$ clauses. Thus the CNF has $n + \mathcal{O}(nt)$ variables and $m + \mathcal{O}(nt)$ clauses, and it can be generated in time polynomial in n, m, and t.

Input: Undirected graph $G = (V, E)$, integer t
Output: CNF formula ϕ satisfiable iff G has a vertex cover of size at most t
Introduce Boolean variables x_i for each vertex $v_i \in V$;
foreach *edge* $\{u, v\} \in E$ **do**
  Add clause $(x_u \vee x_v)$;
end
if $t = 0$ **then**
  foreach *vertex* $v_i \in V$ **do**
    Add clause $\neg x_i$;
  end
end
else
  Apply the sequential counter encoding to enforce $\sum x_i \leq t$ (see Section E.2);
end
return ϕ

Algorithm 5: Reduction from Vertex Cover to SAT

Example 6. *Consider the triangle graph* $G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\})$. *To determine whether there exists a vertex cover of size* $t = 2$, *we introduce variables* x_1, x_2, x_3 *for the three vertices.*

We add clauses for edge coverage:

$$(x_1 \vee x_2), \quad (x_2 \vee x_3), \quad (x_1 \vee x_3).$$

Then, we apply the sequential counter encoding to enforce $x_1 + x_2 + x_3 \leq 2$ *as described in Section E.2. A satisfying assignment is* $x_1 = \mathit{true}, x_2 = \mathit{true}, x_3 = \mathit{false}$, *which corresponds to the vertex cover* $\{v_1, v_2\}$.

This example illustrates how a classical graph constraint is naturally and precisely expressed using SAT.

E.3.2 Graph Coloring

We record a CNF encoding for the decision problem whether G is properly c-colorable.

Definition 67 (Graph Coloring)**.** *Given an undirected graph* $G = (V, E)$ *and an integer* c, *a proper* c*-coloring is a function* $\gamma : V \to \{1, \ldots, c\}$ *such that for every edge* $\{u, v\} \in E$ *we have* $\gamma(u) \neq \gamma(v)$. *The decision problem asks whether such a coloring exists.*

SAT Formula Construction

We construct a CNF formula $\phi_{\mathrm{COL}}(G, c)$ whose satisfying assignments decode into proper c-colorings of G.

Variables. Introduce Boolean variables $x_{v,i} \in \{0,1\}$ for every $v \in V$ and every $i \in \{1,\dots,c\}$, where $x_{v,i} = 1$ means that vertex v is assigned color i.

Color assignment constraint. Each vertex must be assigned at least one color. For every vertex $v \in V$, we add the clause:

$$\bigvee_{i=1}^{c} x_{v,i}$$

Color uniqueness constraint. No vertex may be assigned more than one color. For every vertex $v \in V$, we enforce the at-most-one constraint

$$\sum_{i=1}^{c} x_{v,i} \leq 1,$$

encoded here using the standard pairwise clauses: for all distinct colors $i \neq j$,

$$(\neg x_{v,i} \vee \neg x_{v,j}).$$

Adjacency constraint. Adjacent vertices must receive different colors. For every edge $\{u,v\} \in E$ and for every color $i \in \{1,\dots,c\}$, we add the clause:

$$(\neg x_{u,i} \vee \neg x_{v,i})$$

Lemma 59 (Correctness of the Graph Coloring encoding)**.** *The formula $\phi_{\mathrm{COL}}(G,c)$ is satisfiable if and only if G is c-colorable.*

Proof. **Completeness.** Let $\gamma : V \to \{1,\dots,c\}$ be a proper c-coloring. Define an assignment α by $\alpha(x_{v,i}) = 1$ iff $\gamma(v) = i$. Then every vertex satisfies the at-least-one clause, and the at-most-one constraint holds for each vertex, so all pairwise clauses $(\neg x_{v,i} \vee \neg x_{v,j})$ are satisfied. For every edge $\{u,v\} \in E$ and every i, we have $\gamma(u) \neq \gamma(v)$, so not both $x_{u,i}$ and $x_{v,i}$ are true. Thus all adjacency clauses are satisfied. Hence $\alpha \models \phi_{\mathrm{COL}}(G,c)$.

Soundness. Let $\alpha \models \phi_{\mathrm{COL}}(G,c)$. For each vertex v, the at-least-one clause ensures there is some i with $\alpha(x_{v,i}) = 1$, and the pairwise at-most-one clauses ensure this i is unique. Define $\gamma(v)$ to be that unique color. If $\{u,v\} \in E$ and $\gamma(u) = \gamma(v) = i$, then $\alpha(x_{u,i}) = \alpha(x_{v,i}) = 1$ violates $(\neg x_{u,i} \vee \neg x_{v,i})$, a contradiction. Thus γ is a proper c-coloring. □

Complexity. The encoding introduces $n \cdot c$ Boolean variables, where $n = |V|$. Each variable $x_{v,i}$ represents the assignment of color i to vertex v.

The color assignment constraint adds one clause of length c for each vertex. The at-most-one constraint contributes $\binom{c}{2}$ binary clauses per vertex, i.e., $\mathcal{O}(nc^2)$ in total. The adjacency constraint adds c binary clauses for each edge. Thus, the total number of clauses is:

$$\mathcal{O}(n \cdot c + n \cdot c^2 + |E| \cdot c) = \mathcal{O}(nc^2 + |E|c).$$

All clauses are either of length 2 or c, and the construction runs in time polynomial in n, m, and c.

Symmetry breaking and certificate decoding. The encoding has a color-permutation symmetry. One may break it by fixing the color of one seed vertex (e.g., by adding the unit clause $x_{v_1,1}$ for the smallest-labeled vertex v_1), which preserves satisfiability. Our implementation does not rely on this. A satisfying assignment decodes into the function $\gamma : V \to [c]$, which we compare against the acceptance of $\mathcal{C}_c^\chi$ by scanning c (or via binary search) in the TreeWidzard pipeline.

Input: Undirected graph $G = (V, E)$, integer c
Output: CNF formula ϕ, satisfiable if and only if G is c-colorable
Introduce Boolean variables $x_{v,i} \in \{0, 1\}$ for each vertex $v \in V$ and color $i \in \{1, \ldots, c\}$;
foreach *vertex* $v \in V$ **do**
  Add clause $\bigvee_{i=1}^{c} x_{v,i}$;
  foreach *pair of colors* $1 \le i < j \le c$ **do**
    Add clause $(\neg x_{v,i} \vee \neg x_{v,j})$;
  end
end
foreach *edge* $\{u, v\} \in E$ **do**
  foreach *color* $i \in \{1, \ldots, c\}$ **do**
    Add clause $(\neg x_{u,i} \vee \neg x_{v,i})$;
  end
end
return ϕ

Algorithm 6: Reduction from c-Coloring to SAT

Example 7. *Let $G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\})$ be a triangle graph. We ask whether G is 3-colorable. Introduce Boolean variables $x_{v_i,i} \in \{0, 1\}$ for all $v_i \in \{v_1, v_2, v_3\}$ and $i \in \{1, 2, 3\}$.*

For each vertex, we add one clause requiring at least one color to be assigned and enforce that at most one color is chosen via pairwise at-most-one clauses. For each edge and each color, we add a clause ensuring adjacent vertices do not receive the same color.

A satisfying assignment such as:

$$x_{v_1,1} = \mathit{true}, \quad x_{v_2,2} = \mathit{true}, \quad x_{v_3,3} = \mathit{true}$$

satisfies all constraints and represents a proper 3-coloring of the graph. Therefore, the formula is satisfiable.

E.3.3 Independent Set

We record a CNF encoding for the decision problem whether G contains an independent set of size at least t.

Definition 68 (Independent Set)**.** *Given an undirected graph $G = (V, E)$ and an integer t, an independent set is a subset $S \subseteq V$ such that no two vertices in S are adjacent. The decision problem asks whether such a subset of size at least t exists.*

SAT Formula Construction

We construct a CNF formula $\phi_{\mathrm{IS}}(G, t)$ whose satisfying assignments encode independent sets of size at least t.

Variables. Introduce one Boolean variable x_v for each vertex $v \in V$, where $x_v = 1$ means that v is selected into the independent set. For an enumeration $V = \{v_1, \ldots, v_n\}$ we also write $x_i := x_{v_i}$.

Non-adjacency constraint. For every edge $\{u, v\} \in E$, the corresponding vertices must not both be included in the independent set. This is enforced using binary clauses:

$$(\neg x_u \vee \neg x_v)$$

Cardinality constraint. To ensure that the number of selected vertices is at least t, we require:

$$\sum_{i=1}^{n} x_i \geq t$$

which is equivalently rewritten as:

$$\sum_{i=1}^{n} \neg x_i \leq n - t$$

If $t > n$, we instead define $\phi_{\mathrm{IS}}(G, t) := \bot$, since no n-vertex graph can contain an independent set of size at least t. If $t = n$, this is simply the conjunction $\bigwedge_{i=1}^{n} x_i$. Otherwise $n - t \geq 1$, so we apply the sequential counter encoding described in Section E.2 to the negated variables $\neg x_i$, enforcing the upper bound $\sum \neg x_i \leq n - t$. This introduces

$\mathcal{O}(n(n-t))$ auxiliary variables and $\mathcal{O}(n(n-t))$ clauses. In the implementation used for the experimental campaign, we enforce the lower bound using PySAT's built-in cardinality encoder `CardEnc.atleast` with `EncType.seqcounter`, which yields an equisatisfiable sequential-counter-based CNF instance.

Lemma 60 (Correctness of the Independent Set encoding)**.** *The formula $\phi_{\mathrm{IS}}(G,t)$ is satisfiable if and only if G has an independent set of size at least t.*

Proof. **Out-of-range case.** If $t > n$, then G has no independent set of size at least t, and by definition $\phi_{\mathrm{IS}}(G,t) = \bot$ is unsatisfiable. Thus both sides are false. Assume from now on that $t \leq n$.

Completeness. Let $S \subseteq V$ be an independent set with $|S| \geq t$. Define an assignment α by $\alpha(x_v) = 1$ iff $v \in S$. For every edge $\{u,v\} \in E$, at most one endpoint lies in S, hence the clause $(\neg x_u \vee \neg x_v)$ is satisfied. If $t = n$, then $|S| \geq n$ forces $S = V$, so every unit clause x_v is satisfied. Otherwise $n - t \geq 1$, and $\sum_{v \in V} \alpha(x_v) = |S| \geq t$, equivalently $\sum_{v \in V} \alpha(\neg x_v) \leq n - t$, so by Lemma 57 (applied to the literals $\neg x_v$) the cardinality gadget can be satisfied by extending α to its auxiliary variables. Thus $\alpha \models \phi_{\mathrm{IS}}(G,t)$.

Soundness. Let $\alpha \models \phi_{\mathrm{IS}}(G,t)$ and define $S := \{v \in V \mid \alpha(x_v) = 1\}$. For every edge $\{u,v\} \in E$, the clause $(\neg x_u \vee \neg x_v)$ is satisfied, so not both endpoints lie in S. Hence S is independent. If $t = n$, then all unit clauses x_v are present, so $S = V$ and hence $|S| = n \geq t$. Otherwise, the sequential-counter part enforces $\sum_{v \in V} \neg x_v \leq n - t$, i.e., $\sum_{v \in V} x_v \geq t$, so $|S| \geq t$. □

Complexity. If $t > n$, the formula is a fixed unsatisfiable CNF of constant size. Otherwise, the encoding uses n primary variables and $\mathcal{O}(n(n-t))$ auxiliary variables for the sequential counter on $\neg x_1, \ldots, \neg x_n$ with bound $n - t$. The non-adjacency constraint contributes m binary clauses, and the sequential counter contributes $\mathcal{O}(n(n-t))$ clauses, so the CNF has $n + \mathcal{O}(n(n-t))$ variables and $m + \mathcal{O}(n(n-t))$ clauses. It can be generated in time polynomial in n, m, and t.

Example 8. *Consider the triangle graph $G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\})$. We wish to determine whether there exists an independent set of size at least 1. We introduce Boolean variables x_1, x_2, x_3 for the vertices.*

We add the clauses:

$$(\neg x_1 \vee \neg x_2), \quad (\neg x_2 \vee \neg x_3), \quad (\neg x_1 \vee \neg x_3)$$

to ensure that no adjacent pair is selected.

The size constraint $x_1 + x_2 + x_3 \geq 1$ is encoded using the sequential counter described in Section E.2, applied to the negated variables. A satisfying assignment such as $x_1 = \mathit{true}, x_2 = \mathit{false}, x_3 = \mathit{false}$ verifies that $\{v_1\}$ is a valid independent set.

Input: Undirected graph $G = (V, E)$, integer t
Output: CNF formula ϕ satisfiable if and only if G has an independent set of size at least t
if $t > |V|$ **then**
  Add the empty clause (unsatisfiable);
  return ϕ;
end
Introduce Boolean variables x_i for each vertex $v_i \in V$;
foreach *edge* $\{u, v\} \in E$ **do**
  Add clause $(\neg x_u \vee \neg x_v)$ to ϕ;
end
Rewrite $\sum x_i \geq t$ as $\sum \neg x_i \leq n - t$;
if $t = n$ **then**
  Add the unit clauses $x_1, \ldots, x_n$ to ϕ;
end
else
  Apply the sequential counter encoding to $\neg x_1, \ldots, \neg x_n$ (see Section E.2);
  Add all resulting clauses to ϕ;
end
return ϕ

Algorithm 7: Reduction from Independent Set to SAT

E.3.4 Hamiltonian Path

We record a CNF encoding for the decision problem whether G has a Hamiltonian path. Following the simple-path convention in Appendix B, the empty graph is treated as a negative instance; for $n = |V(G)| = 0$ we set $\phi_{\mathrm{HP}}(G) := \bot$.

Definition 69 (Hamiltonian Path)**.** *Given an undirected graph $G = (V, E)$ with $n = |V| \geq 1$, a Hamiltonian path is a permutation $(p_1, p_2, \ldots, p_n)$ of V such that $\{p_i, p_{i+1}\} \in E$ for all $1 \leq i < n$.*

SAT Formula Construction

For $n = 0$, let $\phi_{\mathrm{HP}}(G) := \bot$. For $n \geq 1$, we construct a CNF formula $\phi_{\mathrm{HP}}(G)$ whose satisfying assignments decode into Hamiltonian paths of G.

Variables. For each vertex $v \in V$ and each position $i \in \{1, \ldots, n\}$, introduce a Boolean variable $x_{v,i}$. The intended meaning is that $x_{v,i} = 1$ iff v occupies position i in the path.

Vertex-position uniqueness. Each vertex must appear in exactly one position in the path. For each vertex $v \in V$, we require:

$$\sum_{i=1}^{n} x_{v,i} = 1$$

Since equality constraints are not directly in CNF form, we split this equality into two constraints:

$$\sum_{i=1}^{n} x_{v,i} \leq 1 \quad \text{and} \quad \sum_{i=1}^{n} x_{v,i} \geq 1$$

The upper-bound constraint $\sum_{i=1}^{n} x_{v,i} \leq 1$ is encoded using the sequential counter method described in Section E.2. This encoding introduces $\mathcal{O}(n)$ auxiliary variables and clauses. The lower-bound constraint $\sum_{i=1}^{n} x_{v,i} \geq 1$ is encoded using a single disjunctive clause:

$$x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,n}$$

Sequential counters are not used for lower-bound constraints, as a single clause is sufficient and more efficient.

Position-vertex uniqueness. Each position in the path must be assigned to exactly one vertex. For each position $i \in \{1, \ldots, n\}$, we enforce:

$$\sum_{v \in V} x_{v,i} = 1$$

This is again split into the two constraints:

$$\sum_{v \in V} x_{v,i} \leq 1 \quad \text{and} \quad \sum_{v \in V} x_{v,i} \geq 1$$

The upper-bound constraint is encoded using a sequential counter applied to the variables $x_{v,i}$ for fixed i, introducing $\mathcal{O}(n)$ auxiliary variables and clauses. The lower-bound constraint is encoded as a single clause:

$$x_{v_1,i} \vee x_{v_2,i} \vee \cdots \vee x_{v_n,i}$$

where $v_1, \ldots, v_n$ enumerate all vertices in V. Again, a sequential counter is not required for this part.

Adjacency preservation. To ensure that the path only moves between adjacent vertices, we forbid transitions between non-adjacent pairs. If two vertices $u, v \in V$ are not connected by an edge in the graph, then they must not appear in consecutive positions in the path. For each pair $\{u, v\} \notin E$ and each position $i \in \{1, \ldots, n-1\}$, we add the clause:

$$(\neg x_{u,i} \vee \neg x_{v,i+1})$$

and symmetrically

$$(\neg x_{v,i} \vee \neg x_{u,i+1}).$$

Together these clauses ensure that no non-edge can occur as a consecutive pair in either direction. The complete SAT formula consists of all clauses generated by the above constraints. Each equality constraint is expressed as the combination of a sequential counter for the upper bound and a single clause for the lower bound, ensuring both correctness and efficiency. The resulting formula is in CNF and has a size polynomial in $n = |V|$, making it suitable for modern SAT solvers.

Lemma 61 (Correctness of the Hamiltonian Path encoding)**.** *The formula $\phi_{\mathrm{HP}}(G)$ is satisfiable if and only if G has a Hamiltonian path.*

Proof. **Empty-graph case.** If $n = 0$, then G has no Hamiltonian path under the nonempty simple-path convention, and $\phi_{\mathrm{HP}}(G) = \bot$ is unsatisfiable. Assume from now on that $n \geq 1$.

Completeness. Let $(p_1, \ldots, p_n)$ be a Hamiltonian path of G. Define an assignment α by setting $\alpha(x_{v,i}) = 1$ iff $v = p_i$. Then each vertex appears in exactly one position and each position contains exactly one vertex, so all row/column constraints are satisfied (using Lemma 57 for the at-most-one parts). Moreover, for every non-edge $\{u, v\} \notin E$ and every $i \in \{1, \ldots, n-1\}$, the consecutive pair (p_i, p_{i+1}) is an edge, so we never have $\alpha(x_{u,i}) = \alpha(x_{v,i+1}) = 1$. Hence all non-edge clauses are satisfied. Thus $\alpha \models \phi_{\mathrm{HP}}(G)$.

Soundness. Let $\alpha \models \phi_{\mathrm{HP}}(G)$. By the per-position constraints, for every i there is a unique vertex p_i with $\alpha(x_{p_i,i}) = 1$, and by the per-vertex constraints each vertex occurs in exactly one position. Hence $(p_1, \ldots, p_n)$ is a permutation of V. If $\{p_i, p_{i+1}\} \notin E$ for some i, then the formula contains the clause $(\neg x_{p_i,i} \vee \neg x_{p_{i+1},i+1})$, which is falsified by $\alpha(x_{p_i,i}) = \alpha(x_{p_{i+1},i+1}) = 1$, a contradiction. Therefore $\{p_i, p_{i+1}\} \in E$ for all i, and $(p_1, \ldots, p_n)$ is a Hamiltonian path. □

Complexity. Let $G = (V, E)$ be the input graph with $n = |V|$ vertices. The SAT encoding introduces one Boolean variable $x_{v,i}$ for every pair $(v, i) \in V \times \{1, \ldots, n\}$, representing that vertex v appears in position i of the path. This gives a total of n^2 Boolean variables.

Each of the vertex-position and position-vertex uniqueness constraints consists of two parts: an at-most-one constraint and an at-least-one constraint. The at-most-one constraints are encoded using sequential counters, which introduce $\mathcal{O}(n)$ auxiliary variables and $\mathcal{O}(n)$ clauses per vertex (or per position). Since there are n such constraints in each group, the total number of auxiliary variables introduced is $\mathcal{O}(n^2)$, and the number of clauses is also $\mathcal{O}(n^2)$. The at-least-one constraints are each encoded using a single disjunctive clause of size n, contributing n clauses per group, i.e., $2n$ clauses in total (hence $\mathcal{O}(n)$).

The adjacency constraints are added for every pair of non-adjacent vertices $\{u, v\} \notin E$

and for each position $i \in \{1, \ldots, n-1\}$. Since we add two clauses per non-edge and position, and the number of non-edges is at most $\binom{n}{2} - |E|$, the number of adjacency clauses is at most $2(\binom{n}{2} - |E|)(n-1) = \mathcal{O}(n^3)$ in the worst case (i.e., when $E = \emptyset$).

Summing all components, the total number of variables (including auxiliary variables) is $\mathcal{O}(n^2)$, and the total number of clauses is $\mathcal{O}(n^3)$. The size of the formula, defined as the total number of literals across all clauses, is also bounded by $\mathcal{O}(n^3)$, since each clause is of constant size or at most linear in n.

Thus the CNF uses $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^3)$ clauses, and it can be generated in time polynomial in n and m.

Symmetry breaking and certificate decoding. The encoding has an inherent reversal symmetry: if $(p_1, \ldots, p_n)$ is a Hamiltonian path, then $(p_n, \ldots, p_1)$ is also a Hamiltonian path. When desired, we break only this symmetry by imposing an order on the endpoints (e.g., the label at position 1 is smaller than the label at position n). For a labeled vertex set $V = \{v_1, \ldots, v_n\}$, this can be encoded by adding, for all indices $a > b$, the binary clauses $(\neg x_{v_a,1} \vee \neg x_{v_b,n})$. This is equisatisfiable because one of a path and its reversal satisfies the endpoint order. A satisfying assignment decodes into the ordered sequence $(p_1, \ldots, p_n)$, which can be validated directly against G. We apply this only when $n \geq 2$. For $n = 1$ the reversal symmetry is vacuous.

Input: Undirected graph $G = (V, E)$, number of vertices n
Output: CNF formula ϕ, satisfiable if and only if G has a Hamiltonian path
if $n = 0$ **then**
  Add the empty clause (unsatisfiable);
  return ϕ;
end
Introduce Boolean variables $x_{v,i} \in \{0, 1\}$ for all $v \in V$, $i \in \{1, \ldots, n\}$;
Vertex-position constraints:;
foreach *vertex* $v \in V$ **do**
  Add clause $x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,n}$;
  Encode $\sum_{i=1}^{n} x_{v,i} \leq 1$ using the sequential counter;
end
Position-vertex constraints:;
foreach *position* $i = 1$ *to* n **do**
  Add clause $\bigvee_{v \in V} x_{v,i}$;
  Encode $\sum_{v \in V} x_{v,i} \leq 1$ using the sequential counter;
end
Adjacency constraints:;
foreach *pair* $\{u, v\} \notin E$ **do**
  foreach *position* $i = 1$ *to* $n - 1$ **do**
    Add clause $(\neg x_{u,i} \vee \neg x_{v,i+1})$;
    Add clause $(\neg x_{v,i} \vee \neg x_{u,i+1})$;
  end
end
return ϕ

Algorithm 8: Reduction from Hamiltonian Path to SAT

Example 9. *Consider the path graph* $G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}\})$*, which admits a Hamiltonian path:* $v_1 \to v_2 \to v_3$*.*

We introduce Boolean variables $x_{v,i}$ *for each vertex* $v \in \{v_1, v_2, v_3\}$ *and each position* $i \in \{1, 2, 3\}$*. According to the construction, each vertex must appear in exactly one position, and each position must be filled by exactly one vertex. These conditions are enforced through disjunctive clauses (for the "at least one" constraints) and sequential counters (for the "at most one" constraints).*

The only pair of vertices not connected by an edge is $\{v_1, v_3\}$*. The encoding still includes the corresponding non-edge clauses for every consecutive position, namely the clauses forbidding* v_1 *immediately before or after* v_3*. In the displayed satisfying assignment these vertices are placed at positions 1 and 3, respectively, so all of those non-edge clauses are satisfied.*

A satisfying assignment to the variables is:

$$x_{v_1,1} = \mathit{true}, \quad x_{v_2,2} = \mathit{true}, \quad x_{v_3,3} = \mathit{true}$$

and all other variables set to false. This assignment satisfies all constraints and corresponds to a valid Hamiltonian path in G.

E.3.5 Hamiltonian Cycle

We record a CNF encoding for the decision problem whether G has a Hamiltonian cycle. The construction below assumes $n = |V(G)| \geq 3$. For loopless simple graphs with $n < 3$, the instance is trivially negative, and the standalone encoder may return any fixed unsatisfiable CNF, for example a fresh unit variable together with its negation.

Definition 70 (Hamiltonian Cycle)**.** *Given an undirected graph $G = (V, E)$ with $n = |V| \geq 3$, a Hamiltonian cycle is a cyclic permutation $(p_1, p_2, \ldots, p_n)$ of V such that $\{p_i, p_{i+1}\} \in E$ for all $i \in \{1, \ldots, n\}$, where we identify p_{n+1} with p_1.*

SAT Formula Construction

We construct a CNF formula $\phi_{\mathrm{HC}}(G)$ whose satisfying assignments decode into Hamiltonian cycles of G.

Variables. For each vertex $v \in V$ and each position $i \in \{1, \ldots, n\}$, introduce a Boolean variable $x_{v,i}$. The intended meaning is that $x_{v,i} = 1$ iff v occupies position i in the cycle.

Vertex-position uniqueness. Each vertex must appear exactly once in the cycle. For each vertex $v \in V$, we enforce:

$$\sum_{i=1}^{n} x_{v,i} = 1$$

We express this as two constraints: the lower-bound constraint $\sum_{i=1}^{n} x_{v,i} \geq 1$, encoded as a single clause $x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,n}$, and the upper-bound constraint $\sum_{i=1}^{n} x_{v,i} \leq 1$, encoded using a sequential counter.

Position-vertex uniqueness. Each position in the cycle must be assigned to exactly one vertex. For each position $i \in \{1, \ldots, n\}$, we enforce:

$$\sum_{v \in V} x_{v,i} = 1$$

Again, this is split into a lower-bound constraint (encoded as a single disjunction over all $x_{v,i}$), and an upper-bound constraint (encoded using a sequential counter).

Adjacency preservation. To ensure that only valid consecutive steps are allowed in the cycle, we enforce that for every pair of non-adjacent vertices $\{u, v\} \notin E$, the cycle must not step directly from u to v. Since the cycle is circular, we add constraints for all positions $i \in \{1, \ldots, n\}$, using modular arithmetic:

$$(\neg x_{u,i} \vee \neg x_{v,(i \bmod n)+1})$$

and symmetrically

$$(\neg x_{v,i} \vee \neg x_{u,(i \bmod n)+1}).$$

These clauses are added for every non-edge $\{u, v\} \notin E$ and every position i, ensuring that the cycle never makes an invalid step between non-connected vertices in either direction.

Symmetry breaking (optional). The position encoding has a rotational symmetry: if $(p_1, \ldots, p_n)$ is a Hamiltonian cycle, then any cyclic shift yields another satisfying assignment. One may break this symmetry by fixing a designated vertex (e.g., the smallest-labeled vertex) at position 1, by adding the unit clause $x_{v_1,1}$ and the clauses $\neg x_{v,1}$ for $v \neq v_1$. This preserves satisfiability and is the form used in our implementation.

Lemma 62 (Correctness of the Hamiltonian Cycle encoding)**.** *Assume $n = |V(G)| \geq 3$. The formula $\phi_{\mathrm{HC}}(G)$ is satisfiable if and only if G has a Hamiltonian cycle.*

Proof. **Completeness.** Let $(p_1, \ldots, p_n)$ be a Hamiltonian cycle of G, with $p_{n+1} := p_1$. Define an assignment α by setting $\alpha(x_{v,i}) = 1$ iff $v = p_i$. Then each vertex appears in exactly one position and each position contains exactly one vertex, so all row/column constraints are satisfied (using Lemma 57 for the at-most-one parts). Moreover, for every non-edge $\{u, v\} \notin E$ and every i, the consecutive cyclic pair (p_i, p_{i+1}) is an edge, so we never have $\alpha(x_{u,i}) = \alpha(x_{v,(i \bmod n)+1}) = 1$. Hence all non-edge clauses are satisfied. Thus $\alpha \models \phi_{\mathrm{HC}}(G)$.

Soundness. Let $\alpha \models \phi_{\mathrm{HC}}(G)$. By the per-position constraints, for every i there is a unique vertex p_i with $\alpha(x_{p_i,i}) = 1$, and by the per-vertex constraints each vertex occurs in exactly one position. Hence $(p_1, \ldots, p_n)$ is a cyclic ordering of V. If $\{p_i, p_{(i \bmod n)+1}\} \notin E$ for some i, then the formula contains the clause $(\neg x_{p_i,i} \vee \neg x_{p_{(i \bmod n)+1},(i \bmod n)+1})$, which is falsified by $\alpha(x_{p_i,i}) = \alpha(x_{p_{(i \bmod n)+1},(i \bmod n)+1}) = 1$, a contradiction. Therefore every consecutive cyclic pair is an edge, so $(p_1, \ldots, p_n)$ is a Hamiltonian cycle. □

Complexity. The encoding introduces n^2 Boolean variables, one for each pair of a vertex $v \in V$ and a position $i \in \{1, \ldots, n\}$, where $n = |V|$. These variables represent that vertex v appears at position i in the Hamiltonian cycle.

The vertex-position constraints require that each vertex appears exactly once in the cycle. For each vertex v, the constraint $\sum_{i=1}^{n} x_{v,i} = 1$ is split into two parts. The lower-

bound constraint $\sum_{i=1}^{n} x_{v,i} \geq 1$ is encoded using a single disjunctive clause of size n, and the upper-bound constraint $\sum_{i=1}^{n} x_{v,i} \leq 1$ is encoded using a sequential counter. This encoding introduces $\mathcal{O}(n)$ auxiliary variables and $\mathcal{O}(n)$ clauses per vertex. Since there are n vertices, the total contribution of the vertex-position constraints is $\mathcal{O}(n^2)$ auxiliary variables and $\mathcal{O}(n^2)$ clauses.

Similarly, the position-vertex constraints ensure that each position in the cycle is filled by exactly one vertex. For each position i, the constraint $\sum_{v \in V} x_{v,i} = 1$ is split into a lower-bound clause and an upper-bound sequential counter. This again introduces $\mathcal{O}(n)$ auxiliary variables and $\mathcal{O}(n)$ clauses per position. As there are n positions, the total contribution is another $\mathcal{O}(n^2)$ auxiliary variables and $\mathcal{O}(n^2)$ clauses.

The adjacency constraints are added to prevent invalid transitions between non-adjacent vertices. For each non-edge $\{u, v\} \notin E$ and each position $i \in \{1, \ldots, n\}$, we add the two clauses $(\neg x_{u,i} \vee \neg x_{v,(i \bmod n)+1})$ and $(\neg x_{v,i} \vee \neg x_{u,(i \bmod n)+1})$. The number of non-edges is at most $\binom{n}{2} - |E|$, and since each such pair generates $2n$ clauses, the total number of adjacency clauses is still $\mathcal{O}(n^3)$ in the worst case.

Summing all components, the total number of Boolean variables, including auxiliary ones, is $\mathcal{O}(n^2)$, and the total number of clauses in the resulting CNF formula is $\mathcal{O}(n^3)$. All parts of the construction (variable generation, clause addition, and sequential counter encoding) can be performed in time polynomial in n and m.

Input: Undirected graph $G = (V, E)$, number of vertices n
Output: CNF formula ϕ, satisfiable if and only if G has a Hamiltonian cycle
if $n < 3$ **then**
    Add the empty clause (unsatisfiable);
    return ϕ;
end
Introduce Boolean variables $x_{v,i} \in \{0, 1\}$ for all $v \in V$, $i \in \{1, \ldots, n\}$;
Vertex-position constraints: each vertex appears exactly once;
foreach *vertex* $v \in V$ **do**
    Add clause $x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,n}$;
    Encode $\sum_{i=1}^{n} x_{v,i} \leq 1$ using a sequential counter;
end
Position-vertex constraints: each position is occupied by exactly one vertex;
foreach *position* $i = 1$ *to* n **do**
    Add clause $\bigvee_{v \in V} x_{v,i}$;
    Encode $\sum_{v \in V} x_{v,i} \leq 1$ using a sequential counter;
end
Adjacency constraints: disallow invalid transitions in the cycle;
foreach *non-edge* $\{u, v\} \notin E$ **do**
    foreach *position* $i = 1$ *to* n **do**
        Add clause $(\neg x_{u,i} \vee \neg x_{v,(i \bmod n)+1})$;
        Add clause $(\neg x_{v,i} \vee \neg x_{u,(i \bmod n)+1})$;
    end
end
return ϕ

Algorithm 9: Reduction from Hamiltonian Cycle to SAT

Example 10. *Consider the cycle graph* $G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_1\}\})$, *which clearly admits a Hamiltonian cycle:* $v_1 \to v_2 \to v_3 \to v_1$. *We introduce variables* $x_{v,i}$ *for each* $v \in \{v_1, v_2, v_3\}$ *and* $i \in \{1, 2, 3\}$. *Each vertex must appear in one position, and each position must be filled by one vertex. Since all pairs of vertices are connected by edges, no adjacency clauses are added.*

A satisfying assignment is:

$$x_{v_1,1} = \mathit{true}, \quad x_{v_2,2} = \mathit{true}, \quad x_{v_3,3} = \mathit{true}$$

and all other variables are false. This assignment satisfies the full formula and corresponds to a valid Hamiltonian cycle in G.

E.3.6 Clique

We record a CNF encoding for the decision problem whether G contains a clique of size at least ℓ.

Definition 71 (Clique)**.** *Let $G = (V, E)$ be an undirected graph and let $\ell \in \mathbb{N}$. A set $K \subseteq V$ is a clique of size at least ℓ if*

$$|K| \geq \ell \quad \textit{and} \quad \{u, v\} \in E \textit{ for all distinct } u, v \in K.$$

The associated decision problem asks whether such a set K exists in G.

SAT Formula Construction

Let $G = (V, E)$ with $|V| = n$. We introduce a Boolean variable $x_v \in \{0, 1\}$ for every vertex $v \in V$ and interpret $x_v = 1$ as "vertex v is selected for the clique". The CNF formula consists of:

- clauses ensuring that the chosen vertices form a clique.
- a cardinality constraint enforcing that at least ℓ vertices are selected.

Adjacency constraint. To guarantee that the selected vertices form a clique, we forbid selecting two vertices that are not adjacent. For each unordered pair $\{u, v\} \subseteq V$ with $\{u, v\} \notin E$, we add the binary clause

$$(\neg x_u \vee \neg x_v).$$

This clause forces at most one of u and v to be selected. If both variables were set to 1, the clause would be falsified.

Size constraint. We also require that the clique contains at least ℓ vertices:

$$\sum_{v \in V} x_v \;\geq\; \ell.$$

This is a lower-bound cardinality constraint and is not in CNF. Following the general pattern from Section E.2, we rewrite it as an upper bound on the negated variables:

$$\sum_{v \in V} \neg x_v \;\leq\; n - \ell.$$

If $\ell > n$, we instead define $\phi_{\mathrm{CLQ}}(G, \ell) := \bot$, since no n-vertex graph can contain a clique of size at least ℓ. If $\ell = n$, this is simply the conjunction $\bigwedge_{v \in V} x_v$. Otherwise $n - \ell \geq 1$, and we apply the sequential-counter encoding of Section E.2 to the variables

$\neg x_v$, $v \in V$, which introduces a suitable set of auxiliary counter variables and $O(n(n-\ell))$ clauses to enforce the inequality in CNF form. In the implementation used for the experimental campaign, we enforce the lower bound using PySAT's built-in cardinality encoder `CardEnc.atleast` with `EncType.seqcounter`, which yields an equisatisfiable sequential-counter-based CNF instance.

The final SAT instance $\phi_{\mathrm{CLQ}}(G, \ell)$ is obtained by conjoining all adjacency clauses with the clauses produced by the sequential counter.

Lemma 63 (Correctness of the Clique encoding)**.** *The formula $\phi_{\mathrm{CLQ}}(G, \ell)$ is satisfiable if and only if G has a clique of size at least ℓ.*

Proof. **Out-of-range case.** If $\ell > n$, then G has no clique of size at least ℓ, and by definition $\phi_{\mathrm{CLQ}}(G, \ell) = \bot$ is unsatisfiable. Thus both sides are false. Assume from now on that $\ell \leq n$.

Completeness. Assume that G contains a clique $K \subseteq V$ with $|K| \geq \ell$. Define a truth assignment α by $\alpha(x_v) = 1$ iff $v \in K$. If $\ell = n$, then $|K| \geq n$ forces $K = V$, so every unit clause x_v is satisfied. Otherwise, $|K| \geq \ell$ implies $\sum_{v \in V} \alpha(x_v) \geq \ell$, equivalently $\sum_{v \in V} \alpha(\neg x_v) \leq n - \ell$, so the sequential-counter clauses can be satisfied by extension (Lemma 57). Moreover, if $\{u, v\} \notin E$ then u and v cannot both lie in K, hence the clause $(\neg x_u \vee \neg x_v)$ is satisfied. Therefore $\alpha \models \phi_{\mathrm{CLQ}}(G, \ell)$.

Soundness. Conversely, suppose $\phi_{\mathrm{CLQ}}(G, \ell)$ is satisfiable and let α be a satisfying assignment. Define $S := \{v \in V \mid \alpha(x_v) = 1\}$. If $\ell = n$, then all unit clauses x_v are present, so $S = V$ and hence $|S| = n \geq \ell$. Otherwise, the sequential-counter part enforces $\sum_{v \in V} \neg x_v \leq n - \ell$, i.e. $|S| \geq \ell$. If two distinct vertices $u, v \in S$ were not adjacent, then $\{u, v\} \notin E$ and the CNF would contain $(\neg x_u \vee \neg x_v)$, which is falsified by $\alpha(x_u) = \alpha(x_v) = 1$, a contradiction. Hence every pair in S is adjacent, so S is a clique of size at least ℓ. □

Complexity. Let $n = |V|$ and $m = |E|$. If $\ell > n$, the formula is a fixed unsatisfiable CNF of constant size. Assume $\ell \leq n$ for the following bound.

- We introduce one selection variable x_v per vertex, giving n primary variables. The sequential-counter encoding for $\sum_{v \in V} \neg x_v \leq n - \ell$ on n inputs and bound $n - \ell$ adds $O(n(n - \ell))$ auxiliary variables.
- For every non-edge $\{u, v\} \notin E$ we add a single binary clause $(\neg x_u \vee \neg x_v)$. The number of such pairs is at most $\binom{n}{2} - m = O(n^2)$, so the adjacency part contributes $O(n^2)$ clauses.
- The sequential counter contributes $O(n(n-\ell))$ clauses, each of constant length (at most three literals).

Altogether, the formula uses $O(n + n(n - \ell))$ variables (including auxiliaries) and $O(n(n-\ell)+n^2)$ clauses. The total number of literal occurrences is bounded by the same asymptotic expression. All parts of the construction (variable introduction, adjacency clauses, and the sequential-counter gadget) can be generated in time polynomial in n and ℓ. Hence the encoding can be generated in polynomial time.

Input: Undirected graph $G = (V, E)$, target size ℓ
Output: CNF formula ϕ, satisfiable iff G has a clique of size at least ℓ

1. If $\ell > n$, add the empty clause to ϕ and return ϕ.
2. Introduce a Boolean variable x_v for each vertex $v \in V$.
3. For every unordered pair $\{u, v\} \subseteq V$ with $\{u, v\} \notin E$, add the clause $(\neg x_u \vee \neg x_v)$ to ϕ.
4. If $\ell = n$, add the unit clauses x_v for all $v \in V$. Otherwise encode $\sum_{v \in V} x_v \geq \ell$ as $\sum_{v \in V} \neg x_v \leq n - \ell$ and apply the sequential-counter encoding of Section E.2 to the variables $\neg x_v$, adding all resulting clauses to ϕ.
5. Return ϕ.

Algorithm 10: Reduction from Clique to SAT

Example 11. *Consider the path graph*

$$G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}\})$$

and let $\ell = 2$. We introduce Boolean variables $x_{v_1}, x_{v_2}, x_{v_3}$. The only non-edge is $\{v_1, v_3\}$, so the adjacency part of the encoding consists of the single clause

$$(\neg x_{v_1} \vee \neg x_{v_3}).$$

The size constraint demands that at least two vertices are selected:

$$x_{v_1} + x_{v_2} + x_{v_3} \;\geq\; 2,$$

which is rewritten as

$$\neg x_{v_1} + \neg x_{v_2} + \neg x_{v_3} \;\leq\; 1$$

and transformed to CNF via the sequential counter.

One satisfying assignment is

$$x_{v_1} = 1, \quad x_{v_2} = 1, \quad x_{v_3} = 0.$$

Exactly two vertices are selected, so the size constraint holds, and the adjacency clause is satisfied because at least one of x_{v_1}, x_{v_3} is 0. The selected set $\{v_1, v_2\}$ is a clique of size 2, so the SAT instance correctly confirms that G contains a clique of size at least 2.

Alternative Encoding. An alternative encoding defines Boolean variables $y_{v,i} \in \{0,1\}$, where $y_{v,i} = 1$ indicates that vertex $v \in V$ is assigned to position $i \in \{1, \ldots, \ell\}$ in the clique. This formulation includes constraints ensuring that each position is occupied by exactly one vertex, each vertex appears in at most one position, and no two vertices that are not connected by an edge are assigned to any pair of positions. While this encoding introduces more variables and clauses than the one based on selection variables x_v, it is equisatisfiable: it produces a satisfiable formula if and only if G contains a clique of size at least ℓ. The two approaches differ mainly in their structure and suitability for certain extensions or solver heuristics. For cardinalities, totalizers or cardinality networks are viable alternatives to sequential counters. We keep SC for consistency across encodings.

E.3.7 Perfect Matching

We record a CNF encoding for the decision problem whether G has a perfect matching.

Definition 72 (Perfect Matching)**.** *Given an undirected graph $G = (V, E)$, a perfect matching is a set of edges $M \subseteq E$ such that every vertex $v \in V$ is incident to exactly one edge of M.*

SAT Formula Construction

We construct a CNF formula $\phi_{\mathrm{PM}}(G)$ whose satisfying assignments decode into perfect matchings of G.

Variables. Introduce a Boolean variable $y_e \in \{0,1\}$ for every edge $e \in E$, where $y_e = 1$ means that e is selected into the matching.

Per-vertex exact-one constraints. For each vertex $v \in V$, let $\mathrm{Inc}(v)$ denote the set of incident edges. We enforce:

$$\sum_{e \in \mathrm{Inc}(v)} y_e = 1.$$

The lower bound $\sum_{e \in \mathrm{Inc}(v)} y_e \geq 1$ is encoded as a single clause $\bigvee_{e \in \mathrm{Inc}(v)} y_e$, and the upper bound $\sum_{e \in \mathrm{Inc}(v)} y_e \leq 1$ is encoded using the sequential counter of Section E.2. If $\mathrm{Inc}(v) = \emptyset$ for some vertex v, the lower-bound disjunction is empty (equivalently, the CNF contains an empty clause), so the formula becomes unsatisfiable, as required.

Lemma 64 (Correctness of the Perfect Matching encoding)**.** *The formula $\phi_{\mathrm{PM}}(G)$ is satisfiable if and only if G has a perfect matching.*

Proof. **Completeness.** Let $M \subseteq E$ be a perfect matching. Define an assignment α by $\alpha(y_e) = 1$ iff $e \in M$. Then for every vertex v, exactly one incident edge lies in M, so the at-least-one clause $\bigvee_{e \in \mathrm{Inc}(v)} y_e$ is satisfied and the at-most-one bound $\sum_{e \in \mathrm{Inc}(v)} y_e \leq 1$ holds. By Lemma 57 the sequential-counter clauses can be satisfied by extension. Hence $\alpha \models \phi_{\mathrm{PM}}(G)$.

Soundness. Let $\alpha \models \phi_{\mathrm{PM}}(G)$ and define $M := \{e \in E \mid \alpha(y_e) = 1\}$. For every vertex v, the at-least-one clause forces some incident edge of v into M, and the sequential-counter part enforces that at most one incident edge is selected. Thus every vertex is incident to exactly one edge of M, so M is a perfect matching. □

Complexity. There are $m = |E|$ primary variables y_e. The n at-least-one clauses contribute n clauses. Encoding the n at-most-one constraints with sequential counters introduces $\mathcal{O}(\sum_{v \in V} |\mathrm{Inc}(v)|) = \mathcal{O}(m)$ auxiliary variables and $\mathcal{O}(m)$ clauses. Thus the overall size is $\mathcal{O}(m)$ variables and $\mathcal{O}(m + n)$ clauses.

Input: Undirected graph $G = (V, E)$
Output: CNF formula ϕ, satisfiable if and only if G has a perfect matching
Introduce Boolean variables y_e for every edge $e \in E$;
foreach *vertex* $v \in V$ **do**
  Add clause $\bigvee_{e \in \mathrm{Inc}(v)} y_e$;
  Encode $\sum_{e \in \mathrm{Inc}(v)} y_e \leq 1$ using the sequential counter;
end
return ϕ

Algorithm 11: Reduction from Perfect Matching to SAT

Example 12. *As a concrete instance, take the 4-cycle graph on vertices v_1, v_2, v_3, v_4 with edges*

$$\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_1\}\}.$$

One perfect matching is

$$M = \{\{v_1, v_2\}, \{v_3, v_4\}\}.$$

The encoding introduces one variable per edge, say $y_{12}, y_{23}, y_{34}, y_{41}$, and enforces for each vertex that exactly one incident edge-variable is true. Setting $y_{12} = 1$, $y_{34} = 1$, and $y_{23} = y_{41} = 0$ satisfies all per-vertex constraints, and the selected edges form a perfect matching.

E.3.8 Feedback Vertex Set

We record a CNF encoding for the decision problem whether G has a feedback vertex set of size at most t.

Definition 73 (Feedback Vertex Set)**.** *Given an undirected graph $G = (V, E)$ and an*

integer t, a feedback vertex set of size at most t is a set $S \subseteq V$ with $|S| \leq t$ such that the induced subgraph $G[V \setminus S]$ is acyclic (i.e., a forest).

SAT Formula Construction

We construct a CNF formula $\phi_{\mathrm{FVS}}(G, t)$ by explicitly selecting the deleted set S and enforcing that the remaining graph admits an acyclic orientation with outdegree at most one (equivalently, the remaining graph is a forest).

Lemma 65. *Let H be a finite undirected loopless simple graph. Then H is a forest if and only if it admits an acyclic orientation in which every vertex has outdegree at most* 1.

Proof. If H is a forest, orient each tree toward an arbitrary root. Then every vertex has outdegree at most one and the orientation is acyclic. Conversely, suppose H admits an acyclic orientation with outdegree at most one. If H contained an undirected cycle C, consider the restriction of the orientation to the edges of C. Each of the $|C|$ cycle edges contributes exactly one outgoing edge at its tail, so there are exactly $|C|$ outgoing edges along C. Since every vertex has outdegree at most 1, every vertex on C must have outdegree exactly 1 on C, yielding a directed cycle, contradicting acyclicity. Hence H is acyclic and therefore a forest. □

Variables.

- Deletion variables d_v for $v \in V$, where $d_v = 1$ means $v \in S$.
- Orientation variables $p_{u,v}$ for each ordered pair (u, v) with $\{u, v\} \in E$, where $p_{u,v} = 1$ means the edge $\{u, v\}$ is oriented from u to v in the remaining graph.
- Rank variables $r_{v,i}$ for $v \in V$ and $i \in \{1, \ldots, n-1\}$, where $r_{v,i} = 1$ means $\mathrm{rank}(v) \geq i$. These ranks witness acyclicity by strictly increasing along oriented edges.

Deletion budget. If $t = 0$, enforce $\bigwedge_{v \in V} \neg d_v$. Otherwise enforce $\sum_{v \in V} d_v \leq t$ using the sequential counter of Section E.2.

Orientation and outdegree. For each edge $\{u, v\} \in E$, if both endpoints remain (i.e., $\neg d_u$ and $\neg d_v$), exactly one of $p_{u,v}$ and $p_{v,u}$ must hold. This is encoded by:

$$(d_u \vee d_v \vee p_{u,v} \vee p_{v,u}) \quad \text{and} \quad (d_u \vee d_v \vee \neg p_{u,v} \vee \neg p_{v,u}).$$

We also forbid oriented edges incident to deleted vertices by adding $(\neg p_{u,v} \vee \neg d_u)$ and $(\neg p_{u,v} \vee \neg d_v)$ for every $p_{u,v}$.

Finally, for each vertex $u \in V$, we enforce the outdegree bound

$$\sum_{v \in N(u)} p_{u,v} \leq 1,$$

encoded using a sequential counter, where $N(u)$ is the neighborhood of u.

Acyclicity via ranks. We enforce that ranks strictly increase along every oriented edge. For each oriented edge variable $p_{u,v}$, add:

$$(\neg p_{u,v} \vee r_{v,1}),$$

and for every $i \in \{1, \ldots, n-2\}$,

$$(\neg p_{u,v} \vee \neg r_{u,i} \vee r_{v,i+1}),$$

and finally

$$(\neg p_{u,v} \vee \neg r_{u,n-1}).$$

Together with the monotonicity constraints $(\neg r_{v,i+1} \vee r_{v,i})$ for all $v \in V$ and $i \in \{1, \ldots, n-2\}$, these clauses ensure $\text{rank}(v) \geq \text{rank}(u) + 1$ whenever $p_{u,v} = 1$. Hence directed cycles are impossible.

Lemma 66 (Correctness of the Feedback Vertex Set encoding)**.** *The formula $\phi_{\text{FVS}}(G, t)$ is satisfiable if and only if G has a feedback vertex set of size at most t.*

Proof. **Completeness.** Let $S \subseteq V$ be a feedback vertex set with $|S| \leq t$, and let $H := G[V \setminus S]$. Then H is a forest. By Lemma 65, H admits an acyclic orientation with outdegree at most one. Fix such an orientation. Set $d_v = 1$ iff $v \in S$, and set $p_{u,v} = 1$ iff $\{u, v\} \in E(H)$ is oriented $u \to v$ (set all $p_{u,v} = 0$ if $u \in S$ or $v \in S$). Since the orientation is acyclic, it admits a topological order, hence ranks $\text{rank}(v) \in \{0, \ldots, n-1\}$ such that $p_{u,v} = 1 \Rightarrow \text{rank}(u) < \text{rank}(v)$. Set the unary rank variables accordingly: $r_{v,i} = 1$ iff $\text{rank}(v) \geq i$. If $t = 0$, then $S = \varnothing$, so every unit clause $\neg d_v$ is satisfied. Otherwise the budget constraint is enforced by Lemma 57. The outdegree constraints hold by construction, so their sequential-counter clauses can be satisfied by extension as well. The orientation and rank clauses are satisfied by the chosen orientation and ranking. Hence $\phi_{\text{FVS}}(G, t)$ is satisfiable.

Soundness. Let $\alpha \models \phi_{\text{FVS}}(G, t)$ and define $S := \{v \in V \mid \alpha(d_v) = 1\}$. If $t = 0$, then the unit clauses $\neg d_v$ force $S = \varnothing$. Otherwise the budget constraint enforces $|S| \leq t$ by Lemma 57. Let $H := G[V \setminus S]$. For every edge $\{u, v\} \in E(H)$, the orientation clauses force exactly one of $p_{u,v}$ and $p_{v,u}$, so the p-variables define an orientation of H. The outdegree constraints enforce outdegree at most one in this orientation, and the rank

clauses ensure it is acyclic. By Lemma 65, H is a forest, so S is a feedback vertex set of size at most t. □

Complexity. The encoding uses n deletion variables, $2m$ orientation variables, and $n(n-1) = \mathcal{O}(n^2)$ rank variables. The budget constraint contributes $\mathcal{O}(nt)$ auxiliary variables and clauses. The per-vertex outdegree constraints contribute $\mathcal{O}(m)$ auxiliary variables and clauses in total. The rank constraints contribute $\mathcal{O}(mn)$ clauses. Overall, the CNF has $\mathcal{O}(n^2 + m + nt)$ variables and $\mathcal{O}(mn + n^2 + m + nt)$ clauses.

Example 13. *Let G be the triangle C_3 on vertices $\{v_1, v_2, v_3\}$ and let $t = 1$. Deleting $S = \{v_1\}$ breaks all cycles, so S is a feedback vertex set of size at most 1. In the encoding, set $d_{v_1} = 1$ and $d_{v_2} = d_{v_3} = 0$. The remaining graph $G[V \setminus S]$ is a single edge $\{v_2, v_3\}$, which can be oriented (say) $v_2 \to v_3$ with outdegree at most one. Since this orientation is acyclic, one can assign ranks with $\mathrm{rank}(v_2) < \mathrm{rank}(v_3)$ and set the unary rank variables accordingly. This yields a satisfying assignment (extended to the sequential-counter auxiliaries), matching the intended certificate.*

E.3.9 Degree-at-least-d Vertices

We record a witness-style CNF encoding for the decision problem whether G has a vertex of degree at least d.

Definition 74 (Degree-at-least-d)**.** *Let $G = (V, E)$ be an undirected graph and $d \in \mathbb{N}$. The degree-at-least-d problem asks whether there exists a vertex $v \in V$ such that the number of neighbors of v, i.e., $\deg(v) \geq d$.*

SAT Formula Construction

We construct a CNF formula $\phi_{\mathrm{DEG}}(G, d)$ using a one-clause witness encoding: we precompute the high-degree vertices and emit a single disjunction that is satisfiable if and only if the witness set is non-empty.

Variables and clause. Let $V = \{v_1, \ldots, v_n\}$ and define

$$S_{\geq d} := \{\, i \in [n] \mid \deg(v_i) \geq d \,\}.$$

We introduce one witness variable y_i for each $i \in S_{\geq d}$ and add a single clause requiring that at least one of them is true:

$$\bigvee_{i \in S_{\geq d}} y_i.$$

If $S_{\geq d}$ is empty, this disjunction is empty (equivalently, the CNF contains an empty clause) and is unsatisfiable, reflecting that no vertex has degree at least d. No CNF

clauses relate y_i to degrees: the degree test is performed during CNF construction by computing $S_{\geq d}$ from G, and the CNF only witnesses that $S_{\geq d} \neq \emptyset$.

Lemma 67 (Correctness of the Degree-at-least-d encoding)**.** *The formula $\phi_{\mathrm{DEG}}(G, d)$ is satisfiable if and only if G contains a vertex of degree at least d.*

Proof. **Completeness.** If G contains a vertex v_i with $\deg(v_i) \geq d$, then $i \in S_{\geq d}$ and the clause $\bigvee_{i \in S_{\geq d}} y_i$ can be satisfied by setting $y_i = 1$.

Soundness. If $\phi_{\mathrm{DEG}}(G, d)$ is satisfiable, then the disjunction is non-empty and some y_i is set to 1. By construction this implies $i \in S_{\geq d}$ and therefore $\deg(v_i) \geq d$. □

Complexity. The encoding introduces $|S_{\geq d}| \leq n$ Boolean variables y_i and a single clause $\bigvee_{i \in S_{\geq d}} y_i$ of length $|S_{\geq d}|$. The set $S_{\geq d}$ can be computed in $\mathcal{O}(n + m)$ time by scanning adjacency lists, so the overall encoding size is $\mathcal{O}(n)$.

Input: Undirected graph $G = (V, E)$, integer d
Output: CNF formula ϕ satisfiable iff $\exists v_i \in V : \deg(v_i) \geq d$
Compute $S_{\geq d} = \{\, i \in [n] \mid \deg(v_i) \geq d \,\}$;
Introduce Boolean variables y_i for all $i \in S_{\geq d}$;
if $S_{\geq d} = \emptyset$ **then**
  Add the empty clause (unsatisfiable);
end
else
  Add clause $\bigvee_{i \in S_{\geq d}} y_i$;
end
return ϕ

Algorithm 12: One-clause witness encoding for Degree-at-least-d

Example 14. *Consider the star graph $G = (\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}\})$. Vertex v_1 has degree 3, and all others have degree 1. To determine if $\exists v_i : \deg(v_i) \geq 2$, we compute $S_{\geq 2} = \{1\}$ and add the clause:*

$$(y_1).$$

This clause is trivially satisfiable, confirming that the graph has a vertex of degree at least 2.

E.3.10 Multiple Edge Detection

We record a witness-style CNF encoding for whether the input multigraph contains a pair of vertices joined by at least two edges.

Definition 75 (Multiple Edge)**.** *Let $G = (V, E)$ be an undirected multigraph, where E is a multiset of unordered vertex pairs. A multiple edge exists if there are vertices $u \neq v$ such that the multiplicity of $\{u, v\}$ in E is at least two (equivalently, there exist two distinct edge occurrences with the same endpoints $\{u, v\}$). The decision problem asks whether G contains any such pair of vertices.*

Remark 23. The formal encoding below targets loopless multigraphs and witnesses non-simplicity only via parallel edges. A straightforward implementation variant could additionally flag self-loops, but that extension is not part of the present correctness statement.

SAT Formula Construction

As in the degree-at-least-d case, this property is polynomial-time decidable and we use the same one-clause witness encoding. We construct a CNF formula $\phi_{\text{MULTI}}(G)$ that is satisfiable if and only if a multiple edge exists.

Variables and clause. Let

$$S_{\text{multi}} := \{\, \{u, v\} \in \binom{V}{2} \mid \text{the multiplicity of } \{u, v\} \text{ in } E \text{ is at least } 2 \,\}.$$

We introduce one witness variable y_{uv} for each $\{u, v\} \in S_{\text{multi}}$ and add the single clause

$$\bigvee_{\{u,v\} \in S_{\text{multi}}} y_{uv}.$$

If S_{multi} is empty, this disjunction is empty and the CNF is unsatisfiable. As above, the multiplicity test is performed during CNF construction by computing S_{multi} from G, and the CNF only witnesses that $S_{\text{multi}} \neq \emptyset$.

Lemma 68 (Correctness of the Multiple Edge Detection encoding)**.** *The formula $\phi_{\text{MULTI}}(G)$ is satisfiable if and only if G contains a multiple edge.*

Proof. **Completeness.** If G contains a multiple edge, then $S_{\text{multi}} \neq \emptyset$ and we can satisfy the disjunction by setting $y_{uv} = 1$ for any $\{u, v\} \in S_{\text{multi}}$.

Soundness. If $\phi_{\text{MULTI}}(G)$ is satisfiable, then the disjunction must be non-empty, so $S_{\text{multi}} \neq \emptyset$. By definition this means that G contains a multiple edge. □

Complexity. The formula introduces $|S_{\text{multi}}|$ witness variables y_{uv} and a single clause of length $|S_{\text{multi}}|$, where $|S_{\text{multi}}| \leq \binom{n}{2}$. The set S_{multi} can be computed in $\mathcal{O}(n+m)$ time by scanning the edge multiset and counting multiplicities.

Input: Undirected multigraph $G = (V, E)$
Output: CNF formula ϕ satisfiable iff G contains a multiple edge
Compute $S_{\text{multi}} := \{\, \{u, v\} \in \binom{V}{2} \mid \text{multiplicity of } \{u, v\} \text{ in } E \geq 2 \,\}$;
Introduce Boolean variables y_{uv} for all $\{u, v\} \in S_{\text{multi}}$;
if $S_{\text{multi}} = \emptyset$ **then**
  Add the empty clause (unsatisfiable);
end
else
  Add clause $\bigvee_{\{u,v\} \in S_{\text{multi}}} y_{uv}$;
end
return ϕ

Algorithm 13: One-clause witness encoding for Multiple Edge Detection

Example 15. *Consider the multigraph $G = (V, E)$ with $V = \{v_1, v_2\}$ and $E = \{\{v_1, v_2\}, \{v_1, v_2\}\}$. There are two identical edges between v_1 and v_2, so we define one variable $y_{1,2}$ and add the clause:*

$$(y_{1,2}).$$

The formula is satisfiable by assigning $y_{1,2} = 1$, reflecting the presence of a multiple edge in the graph.

E.4 Summary

Table E.1 summarizes the encodings collected in this appendix. The bounds refer to a graph with n vertices and m edges, with parameters t, ℓ, c, and d when applicable. The cardinality-gadget column indicates whether the encoding uses the sequential counter from Section E.2, pairwise at-most-one clauses, or a one-clause witness, and the final column records the graph certificate decoded from a satisfying assignment.

Property	#vars	#clauses	Cardinality gadget	Witness interpretation
Vertex Cover	$\mathcal{O}(n + nt)$	$\mathcal{O}(m + nt)$	SC ($\sum x_i \leq t$)	Vertex set $C \subseteq V$
Independent Set	$\mathcal{O}(n + n(n-t))$	$\mathcal{O}(m + n(n-t))$	SC ($\sum \neg x_i \leq n - t$)	Vertex set $S \subseteq V$
Hamiltonian Path	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	SC per row/column	Ordering $p_1, \ldots, p_n$
Hamiltonian Cycle	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	SC per row/column	Cyclic ordering $(p_1, \ldots, p_n)$
Clique	$\mathcal{O}(n + n(n-\ell))$	$\mathcal{O}(n^2 + n(n-\ell))$	SC ($\sum_{v \in V} \neg x_v \leq n - \ell$)	Vertex set $K \subseteq V$
Graph Coloring	$\mathcal{O}(nc)$	$\mathcal{O}(nc^2 + mc)$	Pairwise AMO per vertex	Coloring $\gamma : V \to [c]$
Perfect Matching	$\mathcal{O}(m)$	$\mathcal{O}(n + m)$	SC per vertex	Edge set $M \subseteq E$
Feedback Vertex Set	$\mathcal{O}(n^2 + m + nt)$	$\mathcal{O}(mn{+}n^2{+}m{+}nt)$	SC (budget + outdegree)	Vertex set $S \subseteq V$
Maximum Degree $\geq d$	$\lvert S_{\geq d}\rvert \leq n$	1	None	Vertex $v \in V$ with $\deg(v) \geq d$
Multiple Edge Detection	$\lvert S_{\text{multi}}\rvert \leq \binom{n}{2}$	1	None	Pair $\{u, v\}$ with multiplicity ≥ 2

Table E.1: Asymptotic size of the SAT encodings and the certificates produced by the SAT solver. The cardinality gadget column records whether the encoding uses the sequential counter (SC), pairwise at-most-one clauses (AMO), or a one-clause witness. The last two rows are one-clause witness encodings based on precomputed sets $S_{\geq d}$ and S_{multi}.

Graphic design: Communication Division, UiB / Print: Skipnes AS

uib.no

ISBN: 9788230879009 (print)
9788230874370 (PDF)