ARITHMETIC INTEGRATION OF DECISION PROCEDURES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Ting Zhang
June 2006

© Copyright by Ting Zhang 2006
All Rights Reserved

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Zohar Manna Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Henny B. Sipma

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

David L. Dill

Approved for the University Committee on Graduate Studies.

To the Apple of My I

Acknowledgments

I thank Zohar Manna, for advising me for six years at Stanford. This dissertation would not be possible but for his constant support and encouragement. I learnt from him that the essence of a researcher is "to think" as "for every problem there is a solution". I would have been lost without his guidance.

Many thanks to Henny Sipma for being my co-advisor. Over the years she has patiently advised me to be a better researcher. I owe her a great debt for her meticulously reading through my writings and correcting errors.

Many thanks to Grigori Mints for his enlightening classes and instructions, and for his constant support, counsel, and enthusiasm. He showed me a wonderland of logic.

Thanks to David Dill for the collaboration over the last five years. I am grateful to him for his helpful guidance and also for being a part of my orals and reading committees. I would like to thank my orals committee member, John Mitchell for his valuable feedback.

Many thanks to Maggie McLoughlin, Lynda Harris, Kathleen DiTommaso and other helpful staff of the computer science department. Thanks to Miles Davis for keeping the servers running.

Thanks to the members of my research group, past and present: Nikolaj Bjørner, Aaron Bradley, Michael Colón, Bernd Finkbeiner, César Sánchez, Sriram Sankaranarayanan, Matteo Slanina, and Calogero Zarba. I am grateful to Nikolaj Bjørner whose Ph.D. thesis is a source of inspiring ideas.

Thanks to Gu, Lisha, Parker, Rui, Yan and Yong for being such good pals for years. Thanks to Bill, Carol, Haidong, Hongbin, Honglei, Hui, Jan, Jian, Jianying, Jing, Josh, Jun, Karen, Lei, Linda, Mengqiu, Qiong, Shelley, Steven, Tao, Weilai, Yirong, and Ying for making the years at Stanford very lively. I am so fortunate to have friends like you.

Lastly, and most importantly, I wish to thank my parents, Mr. Zhongsheng Zhang and Mrs. Yanan Yao, and my brother Jian Zhang for their unconditional love and support in the course of my study at Stanford. To them I dedicate this thesis.

This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, by ARO grant DAAD19-01-1-0723, and by NAVY/ONR contract N00014-03-1-0939.

Abstract

Decision procedures are algorithms that can determine validity or satisfiability of first-order formulas in a given decidable theory. They are at the heart of virtually every verification system, reasoning about the system behaviors, proving their safety or finding situations that could lead to bugs. They can efficiently discharge a large number of formulas in specific theories without any user guidance, thereby bringing practical program verification closer to reality. Besides their indispensable role in rigorous formal verification, decision procedures are also of great importance in model checking and program analysis where automatic proof engines can improve the overall efficiency and increase the analysis accuracy.

Decision procedures exist for many specialized logical domains including integers, reals as well as for many data structures frequently appearing in programs such as lists, queues, sets and multisets. Programs, however, often involve multiple data domains, resulting in verification conditions spanning multiple logical theories. To be able to verify high-level programs with rich data types, we need decision procedures that can reason on complex domains.

This thesis offers novel solutions to an important class of decision problems, the mixed constraints on data structures and their quantitative properties. It includes three major contributions.

1. Decision Procedures for Term Algebras with Integer Arithmetic. Term Algebras can model a variety of tree-like data types such as records, lists, stacks, etc., which are essential constructs in programming languages. We developed the basic reduction technique, namely, extraction of accurate integer constraints from data constraints. From the construction of accurate integer constraints that precisely characterize data constraints, we can derive decision procedures for the combined constraints by utilizing decision procedures for data structures and decision procedures for integer arithmetic. We also developed quantifier elimination procedures for the first-order theory of the combined structure. The procedure has optimal complexity; it removes in a single step a block of quantifiers of the same kind regardless of the size of the block.

2. Decision Procedures for Queues with Integer Arithmetic. A *Queue* is a typical data type of linear structure. It is widely used in programming languages and forms the basis for many concurrent algorithms and communication protocols. As a queue can grow at both ends, it does not fall in the category of recursive data structures, which can be modeled as term algebras. For this reason,

we have further improved the reduction technique and developed new normalization procedures to handle the distinguished properties of queues. With these new improvements, we designed decision procedures for the quantifier-free theories as well as quantifier elimination procedures for the first-order theories.

3. Decision Procedures for Knuth-Bendix Order. Using quantifier elimination and the reduction technique developed for solving the combined constraints of recursive data structures and integer arithmetic, we proved the decidability of the first-order theory of *Knuth-Bendix Order*, thereby solving a long-standing open problem in term rewriting (officially listed as RTA open problem 99 since 2000). Knuth-Bendix order is widely used in term rewriting and theorem proving, along with the *lexicographic path order*. Unfortunately, the first-order theory of lexicographic path order is undecidable. Therefore, our result on the decidability of Knuth-Bendix order may greatly benefit future algorithm design in term rewriting and theorem proving.

Contents

Acknowledgments **vi**

Abstract **vii**

1 Introduction **1**
1.1 Combination Problem . . . 2
1.2 Our Contribution . . . 3
1.3 Preliminaries . . . 5
1.4 Thesis Organization and Publications . . . 8

2 Term Algebras with Integers **9**
2.1 Introduction . . . 10
2.2 Term Algebras . . . 13
2.3 Decision Procedures for $\mathrm{Th}^{\forall}(\mathrm{TA})$ . . . 16
2.4 Decision Procedures for Quantifier-free Theories . . . 20
2.4.1 Term Algebras with Integers . . . 20
2.4.2 A Generic Decision Procedure for $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ . . . 22
2.4.3 A Decision Procedure for $\mathrm{Th}^{\forall}(\mathrm{TA}^{\omega}_{\mathbb{Z}})$ . . . 25
2.4.4 A Decision Procedure for $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ . . . 27
2.4.5 Richer Theories on Constant Domain . . . 30
2.4.6 Complexity . . . 30
2.5 Case Study: Verification of Red-Black Tree Algorithm . . . 31
2.5.1 The Theory of Red-Black Trees . . . 32
2.5.2 Algorithm and Example . . . 35
2.5.3 Verification Conditions . . . 36
2.6 A New Quantifier Elimination Procedure for Th(TA) . . . 48
2.7 Decision Procedures for Quantified Theories . . . 53
2.7.1 Term Algebras with Integers and Parameters . . . 54
2.7.2 A Quantifier Elimination Procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$ . . . 57

2.7.3 Richer Theories on Constant Domain . . . 63
2.7.4 Adaptation for Infinite Languages . . . 64
2.8 Conclusion . . . 66
2.9 Proofs . . . 67

3 Queues with Integers **82**
3.1 Introduction. . . . 83
3.2 The Theory of Queues . . . 85
3.3 Decision Procedure for $\mathrm{Th}^{\forall}(\mathfrak{Q})$ . . . 88
3.4 Decision Procedures for $\mathrm{Th}^{\forall}(\mathfrak{Q}_{\mathbb{Z}})$ . . . 90
3.4.1 The Theory of Queues with Integers . . . 90
3.4.2 Decision Procedure for $\mathrm{Th}^{\forall}(\mathfrak{Q}_{\mathbb{Z}})$ . . . 92
3.5 Decision Procedure for $\mathrm{Th}^{\forall}(\mathfrak{Q}^{+}_{\mathbb{Z}})$ . . . 96
3.6 Decision Procedure for $\mathrm{Th}(\mathfrak{Q}_{\mathbb{Z}})$ . . . 110
3.6.1 Normalization in $\mathfrak{Q}_{\mathbb{Z}}$ with Parameters $\bar{\mathbf{Y}}$ . . . 111
3.7 Conclusion . . . 113
3.8 Proofs . . . 114

4 Decidability of Knuth-Bendix Order **123**
4.1 Introduction . . . 124
4.2 Term Algebras with Knuth-Bendix Order . . . 126
4.2.1 Proof Plan . . . 127
4.2.2 Decomposition of Knuth-Bendix Order . . . 128
4.2.3 Gap Orders . . . 128
4.2.4 Boundary Functions . . . 129
4.2.5 Extension of Knuth-Bendix Order . . . 129
4.2.6 Tuples of Terms . . . 130
4.2.7 Delineated Gap Order Completion . . . 132
4.2.8 Lemmas . . . 133
4.3 Quantifier Elimination for $\mathrm{Th}(\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^{+}})$ . . . 136
4.3.1 The Elimination Procedure . . . 136
4.3.2 An Example . . . 139
4.4 Presence of a 0-weight Unary Function . . . 141
4.4.1 Reduction of $u \prec^{\mathsf{w}}_{n} v$ and $u \preceq^{\mathsf{w}}_{n} v$ . . . 141
4.4.2 Reduction of $u \prec^{\mathsf{p}}_{n} v$ and $u \preceq^{\mathsf{p}}_{n} v$ . . . 142
4.4.3 Reduction of $u \prec^{\mathsf{l}}_{n} v$ and $u \preceq^{\mathsf{l}}_{n} v$ . . . 144
4.5 Conclusion . . . 146
4.6 Proofs . . . 147

4.7 Definitions . . . 174
4.8 Reductions . . . 177
4.8.1 Reduction of Closed Equalities . . . 177
4.8.2 Reduction of Closed Stretchable Gap Orders . . . 178
4.8.3 Reduction of Closed Rigid Gap Orders . . . 181
4.8.4 Reduction of Half Open Stretchable Gap Orders . . . 185
4.8.5 Reduction of Half Open Rigid Gap Orders . . . 186
4.8.6 Reductions of Open Stretchable Gap Orders . . . 187
4.8.7 Reductions of Open Rigid Gap Orders . . . 188
4.8.8 Reduction of Closed Tuple Equalities . . . 188
4.8.9 Reduction of Half Open Tuple Equalities. . . . 189
4.8.10 Reductions of Closed Stretchable Tuple Gap Orders . . . 189
4.8.11 Reductions of Closed Rigid Tuple Gap Orders . . . 193
4.8.12 Reductions of Half Open Stretchable Tuple Gap Orders . . . 197
4.8.13 Reductions of Half Open Rigid Tuple Gap Orders . . . 198
4.8.14 Reductions of Open Stretchable Tuple Gap Orders . . . 199
4.8.15 Reductions of Open Rigid Tuple Gap Orders . . . 200

5 Conclusion **202**

Bibliography **205**

List of Definitions **211**

List of Algorithms **213**

List of Propositions **215**

List of Lemmas **216**

List of Theorems **217**

List of Examples **218**

Index **221**

Chapter 1

Introduction

00:28:41:100

> Morpheus : Unfortunately, no one can be told what the matrix is. You have to see it for yourself. This is your last chance. After this, there is no turning back. You take the blue pill, the story ends. You wake up in your bed and believe whatever you want to believe. You take the red pill, you stay in Wonderland, and I show you how deep the rabbit hole goes. Remember, all I'm offering is the truth. Nothing more.
>
> —*The Matrix*.

This thesis presents a general technique for integrating integer arithmetic with decision procedures of recursively defined data structures.

1.1 Combination Problem

Decision procedures are algorithms that can determine whether a formula in a given logical theory is valid or satisfiable. An important distinction between these algorithms and general-purpose theorem provers is the level of automation. Decision procedures are fully automatic; they always terminate with either positive or negative answers. In formal program verification, hundreds of thousands of verification conditions are generated. Program verification will never be practical if human efforts are need to prove these formulas, most of which are repetitive, tediously long and mathematically uninteresting. Decision procedures can automatically discharge verification conditions that fall in the scope of certain decidable theories, and hence relieve users of verification systems from tedious interaction with general-purpose theorem provers. Besides their indispensable role in rigorous formal verification, decision procedures are also of great importance in model checking and program analysis where automatic proof engines can improve the overall efficiency and increase the analysis accuracy.

Decision procedures exist for many specialized logical domains including integers, reals as well as for many data structures frequently appearing in programs such as lists, queues, sets and multi-sets. These specialized decision procedures can only handle a particular class of formulas in a particular theory. Programs, even very simple kind, however, often involve multiple data domains, resulting in verification conditions spanning multiple logical theories. To be able to verify high-level programs with rich data types, we have to find decision procedures that can reason on complex domains.

Designing a decision procedure not only requires a good understanding of the algebraic properties of the specific domains, but also requires a lot of effort and often ingenuity to exploit these algebraic properties. Decision procedures for simple domains are well-studied in this way; by exploiting specific structures of the domains, efficient algorithms are obtained whose complexity in most cases matches the optimal theoretical bounds. Designing a decision procedure for a combined theory from scratch, however, is not practical because there are potentially many different combinations. Combined structures, however, usually are constructed in a modular way, by imposing additional constructs on component substructures. Therefore, it is natural to investigate whether decision procedures for such combined structures can be obtained modularly, namely, by utilizing decision procedures for the component theories as black boxes.

This line of investigation goes back to Nelson and Oppen who discovered a combination procedure for stably infinite quantifier-free theories with disjoint signatures. The procedure combines satisfiability procedures for component theories using equality propagation. The applicability of Nelson-Oppen method, however, is limited mainly by two conditions: (i) it works only with quantifier-free theories and, (ii) it requires that the signatures of component theories are disjoint. It turns out that these two restrictions are very hard to remove in spite of much research in this

direction. There are several recent advances for combining quantified theories or non-disjoint theories. The practical applicability of these general combination methods, however, is limited in that these methods require additional nontrivial conditions on component theories or structures. In fact accumulated evidence shows that the existence of modular combinations like the Nelson-Oppen method should be viewed as an exception. In combining theories with quantifiers or non-disjoint signatures, we should not expect the same level of modularity as in Nelson-Oppen schema, but rather rely on close examination of the characteristics of the combining constructs as well as the properties of the individual component theories.

1.2 Our Contribution

This thesis presents another approach to the combination problem; instead of seeking solutions to general purpose combinations, we study specific combination problems by exploiting algebraic properties of the combined domain. As a result, we offer novel solutions to an important class of decision problems which commonly appear in program verification, namely, the mixed constraints on data structures with constraints on "integral measures" of those data structures. Such constraints can express a wide range of program properties, in particular memory safety properties such as absence of memory overflow and out-of-bound array access, which are crucial for program correctness.

The combination consists of recursively defined data structures and integer arithmetic integrated by the size function that maps a data object to its size. This kind of combination tightly links data domains with integer domains, rendering a combination that does not fall under the Nelson-Oppen framework nor any other current combination techniques.

Our approach is to reduce constraints on data structures to constraints on integers, and in the presence of quantifiers, to reduce quantifiers on data objects to quantifiers on integers. The technical development divides into three parts.

Decision Procedures for Term Algebras with Integer Arithmetic.

Term Algebras can model a variety of tree-like data types such as records, lists, stacks, etc., which are essential constructs in programming languages. We developed the basic reduction technique, namely, extraction of accurate integer constraints from term constraints. From the construction of accurate integer constraints that precisely characterize term constraints, we can derive decision procedures for the combined constraints by utilizing decision procedures for term algebras and decision procedures for integer arithmetic.

We showed that for structures with infinite constant domain such an accurate integer constraint, which is satisfiable if and only if the corresponding term constraint is satisfiable, can be effectively computed and is expressible by a quantifier-free Presburger formula linear in the size of the term

constraint. For structures with finite constant domain we introduce an additional counting constraint to account for the fact that with finitely many constants the number of distinct terms of a particular length is bounded. We showed that also this counting constraint is expressible by a quantifier-free Presburger formula. The latter decision procedure directly extends Oppen's decision procedure for infinite data domains to finite domains.

For the first-order theory, we first developed a new quantifier elimination procedure for the theory of *pure* term algebras which can eliminate blocks of quantifiers of the same kind in one step. Then we extended it to a decision procedure for the theory of term algebras with integer constraints by, again, extracting integer constraints from term constraints combined with a reduction of quantifiers on term variables to quantifiers on integer variables. The complexity of the new decision procedure is k-fold exponential for formulas with k quantifier alternations. This is optimal in the sense that the theory of pure term algebras itself has non-elementary complexity.

Decision Procedures for Queues with Integer Arithmetic.

A queue is a typical data type of linear structure. It is widely used in programming languages and forms the basis for many concurrent algorithms and communication protocols. As a queue can grow at both ends, it does not fall in the category of recursive data structures, which can be modeled as term algebras. For this reason, we have further improved the reduction technique and developed new normalization procedures to handle the distinguished properties of queues. With these new improvements, we designed decision procedures for the quantifier-free theories as well as quantifier elimination procedures for the first-order theories.

Decidability of the First-order Theory of Knuth-Bendix Order.

Using quantifier elimination and the reduction technique developed for solving the decision problem of term algebras augmented with integer arithmetic, we proved the decidability of the first-order theory of Knuth-Bendix Order, thereby solving a long-standing open problem in term rewriting (officially listed as RTA open problem 99 since 2000).

This decidability result is obtained by quantifier elimination on a complex structure containing term algebras and integer arithmetic. In this structure, we have a weight function mapping terms to integers as well as various boundary functions mapping integers to terms. In addition, the Knuth-Bendix order is expanded in two directions. First, the order is decomposed into three disjoint suborders depending on which of three conditions is used in the definition. Secondly, all orders (including the suborders) are extended to gap orders, which assert the least number of distinct objects between two terms. Moreover, as Knuth-Bendix order is recursively defined on a lexicographic extension of itself, gap orders are extended to tuples of terms. Thus we actually established the decidability of a richer theory. In constructing the quantifier elimination procedure, we

overcame several technical challenges, including simplification of complex literals, elimination of integer quantifiers, elimination of equality, elimination of negative literals, and proof of termination.

Knuth-Bendix order has numerous applications in term rewriting and theorem proving, along with *lexicographic path order*. In ordered term rewriting, a strategy built on ordering constraints can dynamically orient an equation, at the time of instantiation, even if the equation is not uniformly orientable. This provides a powerful tool to prove the termination of rewriting systems. In ordered resolution and paramodulation, ordering constraints are used to select maximal literals to perform resolution. It also serves as enabling conditions for inference rules and such conditions can be inherited from previous inferences at each deduction step. This helps to prune redundancy of the search space without compromising refutational completeness. Unfortunately, the first-order theory of lexicographic path order is undecidable. Therefore, our result on the decidability of Knuth-Bendix order may greatly benefit future algorithm design in term rewriting and theorem proving.

It is interesting that the combination of term algebras with integer arithmetic can help solve an open problem in another quite different field. We believe that this demonstrates the effectiveness of our approach to the combination problem; studying concrete combination types where richer algebraic properties can be exploited.

1.3 Preliminaries

We assume the first-order syntactic notions of variables, parameters and quantifiers, and semantic notions of structures, satisfiability and validity as in [16].

Syntax

A *signatureSignature* Σ is a set of *function symbols* and *predicate symbols* each of which is associated with an arity. The function symbols with *arity* 0 are also called *constants*. The set of Σ-*terms* $\mathcal{T}(\Sigma, \mathcal{X})$ is recursively defined by: (i) every constant $c \in \Sigma$ or variable $x \in \mathcal{X}$ is a term, and (ii) if $f \in \Sigma$ is an n-place function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. We write $\mathcal{T}(\Sigma)$ for $\mathcal{T}(\Sigma, \emptyset)$. Equality = is always included as a binary predicate symbol. If φ is a formula, we use $\mathcal{T}(\varphi)$ to denote the set of terms occurring in φ, and $\mathcal{V}(\varphi)$ to denote the set of variables in φ.

An *atomic formula* is a formula of the form $P(t_1, \ldots, t_n)$ where P is an n-place predicate symbol and $t_1, \ldots, t_n$ are terms. A *literal* is an atomic formula or its negation. A variable occurs *free* in a formula if it is not in the scope of a quantifier. A formula without quantifiers is called *quantifier-free*. A *ground formula* is a formula with no variables. A *sentence* is a formula in which no variable occurs free. Every quantifier-free formula can be put into *disjunctive normal form*, that is, a disjunction of conjunctions of literals.

We use $\bar{\mathbf{x}}$ to denote a sequence of variables, say, $x_1, \ldots, x_n$, and $\exists\bar{\mathbf{x}}$ (resp. $\forall\bar{\mathbf{x}}$) as an abbreviation of $\exists x_1, \ldots, \exists x_n$ (resp. $\forall x_1, \ldots, \forall x_n$). We write $\varphi(\bar{\mathbf{x}})$ to indicate $\bar{\mathbf{x}}$ occur freely in φ, without exclusion of other free variables which are of no interest in the context. A formula $\psi(\bar{\mathbf{x}})$ can be put into *prenex form* $Q_1 y_1, \ldots, Q_n y_n\, \varphi(\bar{\mathbf{x}}, y_1, \ldots, y_n)$, where Q_i's are either $\exists$ or $\forall$ and $\varphi(\bar{\mathbf{x}}, y_1, \ldots, y_n)$ is quantifier-free, called the *matrix* of ψ. In case $\bar{\mathbf{x}}$ contain all free variables in $\bar{\mathbf{x}}$, $(\exists\bar{\mathbf{x}})\varphi(\bar{\mathbf{x}})$ and $(\forall\bar{\mathbf{x}})\varphi(\bar{\mathbf{x}})$ are called *existential closure* and *universal closure* of $\varphi(\bar{\mathbf{x}})$, respectively. If, in addition, $\varphi(\bar{\mathbf{x}})$ is quantifier-free, then $(\exists\bar{\mathbf{x}})\varphi(\bar{\mathbf{x}})$ is called $\exists_1$ and $(\forall\bar{\mathbf{x}})\varphi(\bar{\mathbf{x}})$ is called $\forall_1$. The quantifier-free (resp. $\exists_1$, $\forall_1$) fragment of a language is the subclass of quantifier-free (resp. $\exists_1$, $\forall_1$) sentences in the language.

An *expression* E is either a formula or a term. We use $E[e_1, \ldots, e_n]$ to denote that $e_1, \ldots, e_n$ are subexpressions of E.

We use $\equiv$ for syntactic equality. We use interval notation for integer sets. For example, the closed interval $[m, n]$ means $\{i \mid m \leq i \leq n\}$. Similarly for open intervals (m, n) and half-open intervals $[m, n)$ and $(m, n]$. We use $(s_i)_{i\in I}$ to denote a sequence indexed by I. For example, by $(e_i)_{i<n}$ we mean $e_0, \ldots, e_n$. We use the standard notations for partial orders. Suppose $\lhd$ is a partial order, then by $x \trianglelefteq y$ we mean $x = y \vee x \lhd y$. If $\lhd$ is transitive, by $x_1 \lhd x_2 \lhd \cdots \lhd x_n$ we mean $\bigwedge_{0<i<n} x_i \lhd x_{i+1}$.

Semantics

A Σ-*structure* $\mathfrak{A}$ is a tuple $\langle A, I\rangle$ where A is a non-empty *domain* and I is a function that associates each n-place function symbol f (resp. predicate symbol P) with an n-place function $f^{\mathfrak{A}}$ (resp. relation $P^{\mathfrak{A}}$) on A. We use Gothic letters (like $\mathfrak{A}$) for structures and Roman letters (like A) for the underlying domain. We usually denote $\mathfrak{A}$ by $\langle A; \Sigma\rangle$ which is called the *signature* of $\mathfrak{A}$. A *variable assignment* σ (in $\mathfrak{A}$) is a function that assigns each variable an element of A. We use $[\![x]\!]\sigma$ to denote the assigned value of x under σ and $[\![\varphi]\!]\sigma$ for the truth value of φ under σ. $\mathfrak{A} \models [\![\varphi]\!]\sigma$ means φ is true under σ. A formula φ is *satisfiable* (in $\mathfrak{A}$), denoted by $\mathfrak{A} \models_\exists \varphi$, if $\mathfrak{A} \models [\![\varphi]\!]\sigma$ for some σ; is *unsatisfiable* (in $\mathfrak{A}$), denoted by $\mathfrak{A} \not\models_\exists \varphi$, if $\mathfrak{A} \models [\![\varphi]\!]\sigma$ for no σ; is *valid* (in $\mathfrak{A}$), denoted by $\mathfrak{A} \models \varphi$, if $\mathfrak{A} \models [\![\varphi]\!]\sigma$ for any σ. A formula φ is valid if and only if $\neg\, \varphi$ is unsatisfiable.

$\mathfrak{A}$ is a *model* of a set T of sentences if every sentence in T is true in $\mathfrak{A}$. A sentence φ is *(logically) implied* by T (or T-*valid*), written $T \models \varphi$, if φ is true in every model of T. Similarly, we say that φ is T-*satisfiable* if φ is true in some model of T and it is T-*unsatisfiable* otherwise. The notions of (T-)validity and (T-)satisfiability naturally extend to a set of formulas. A *theory* T is a set of sentences that is closed under logical implication, that is, if $T \models \varphi$, then $\varphi \in T$. The theory of $\mathfrak{A}$, written $\mathrm{Th}(\mathfrak{A})$, is the set of all true sentences in $\mathfrak{A}$. By a *quantifier-free theory* of $\mathfrak{A}$, written $\mathrm{Th}^\forall(\mathfrak{A})$, we mean the quantifier-free subclass of $\mathrm{Th}(\mathfrak{A})$.

By satisfiability and validity of a quantifier-free formula, we actually mean the validity of the corresponding $\exists_1$ and $\forall_1$ formulas, respectively. Similarly, the satisfiability problem and the validity problem of a quantifier-free theory (or more precisely, the quantifier-free fragment), respectively, refer to the validity problem of the corresponding $\exists_1$ fragment and $\forall_1$ fragment.

Multisorted Logic

All above notions naturally generalize to multi-sorted logic. In a multi-sorted logic, we have a non-empty set S of *sorts*. Variables, constants, equality symbols and quantifiers are indexed by $s \in S$. A n-ary function symbol f is associated with a $n+1$-tuple $\langle s_1, \ldots, s_{n+1} \rangle$, called the type of f. Similarly, the type of a n-ary predicate symbol p is a n-tuple $\langle s_1, \ldots, s_n \rangle$. For $0 < i \leq n$, we say that the i^{th}-place of f (or p) has sort s_i. We require the type of the equality symbol of sort s be $\langle s, s \rangle$. A term t is of sort s if (i) t is a variable or a constant of sort s, or (ii) t is of the form $f(t_1, \ldots, t_n)$ such that the type of f is $\langle s_1, \ldots, s_n, s \rangle$, and $t_1, \ldots, t_n$ are of sorts $s_1, \ldots, s_n$, respectively. A formula is well-formed if in addition it is *well-typed* in the sense that (i) a term of sort s only occurs in a place of sort s in a function or predicate symbol, and (ii) variables of sort s are only quantified by $\forall_s$ and $\exists_s$.

A multi-sorted structure $\mathfrak{A}$ is a tuple $\langle \{A\}_S, S, I \rangle$ where S is the set of sorts, $\{A\}_S$ are mutually disjoint sets (domains) indexed by S, and I is an interpretation such that (i) each n-ary function symbol f with type $\langle s_1, \ldots, s_{n+1} \rangle$ is assigned a function $F : A_{s_1} \times \cdots \times A_{s_n} \rightarrow A_{s_{n+1}}$; (ii) each n-ary predicate symbol p with type $\langle s_1, \ldots, s_n \rangle$ is assigned a relation $P \subseteq A_{s_1} \times \cdots \times A_{s_n}$. A variable assignment assigns a variable of sort s an element in A_s. Satisfiability, unsatisfiability and validity are defined as above with $\forall_s$ (resp. $\exists_s$) being interpreted as "for all (resp. some) elements in the domain A_s".

Refutational Decision Procedure

All decision procedures for quantifier-free theories presented in this thesis are *refutation-based*; to determine the validity of a formula φ, they determine the unsatisfiability of $\neg \varphi$, which further reduces to determining the unsatisfiability of each disjunct in the *disjunctive normal form* of $\neg \varphi$. Henceforth, in discussions related to quantifier-free theories, a quantifier-free formula always refers to a conjunction of literals. We identify a conjunction of literals Φ with the set of all its conjuncts. By $A \in \Phi$ we mean A is a conjunct of Φ and by $\{A\} \cup \Phi$ we mean $\Phi \wedge A$.

Quantifier Elimination

All decision procedures for the first-order theories presented in this thesis are based on *quantifier elimination*. A theory T is said to *admit quantifier elimination* if any formula can be equivalently (modulo T) and effectively transformed into a quantifier-free formula. If a theory admits quantifier elimination, then the truth value of any sentence is reducible to the truth value of a ground formula.

It is well-known that eliminating arbitrary quantifiers reduces to eliminating existential quantifiers from formulas in the form

$$(\exists \bar{\mathbf{x}}) \left[A_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \cdots \wedge A_n(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \right], \tag{1.1}$$

where $A_i(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ $(0 < i \leq n)$ are literals [19]. In discussions related to quantifier elimination, a quantified formula always refers to a formula of the form (1.1). We also assume that the literals A_i are not of the form $x = t$ when x does not appear in t. For $\exists x(x = t \wedge \Phi(x, \bar{\mathbf{y}}))$ simplifies to $\Phi(t, \bar{\mathbf{y}})$.

Nondeterminism

We present algorithms in a nondeterministic manner; whenever we say "guess ϕ", we mean to add a valid (w.r.t. the context) disjunction $\bigvee_i \phi_i$ (where ϕ is one of the disjuncts) to the matrix of (1.1). When we replace ϕ by $\bigvee_i \phi_i$ or directly introduce $\bigvee_i \phi_i$, it should be understood that an implicit disjunctive splitting is carried out and we work on each resultant disjunct of the form (1.1) "simultaneously". We call the target formula *redex* and the resulting formula *reduct*.

Presburger Arithmetic

Presburger arithmetic is the first-order theory of addition in the arithmetic of integers. The corresponding structure is denoted by PA $= \langle \mathbb{Z}; 0, +, < \rangle$. We use $\mathscr{L}_{\mathbb{Z}}$ to denote the formal language of PA.

Presburger arithmetic was first shown to be decidable in 1929 by quantifier elimination [16]. A more efficient algorithm was later discovered by Cooper [12] and further improved by Reddy and Loveland [46].

1.4 Thesis Organization and Publications

Chapter 2 presents decision procedures for the theory of term algebras with integers. It introduces the technique to reduce term constraints to integer constraints, and in the presence of quantifiers, term quantifiers to integer quantifiers. These decision procedures were first published, without proofs, in IJCAR'04 [60] (*Best Paper Award*) and in TPHOLs'04 [61]; an expanded version, including proofs, has been accepted for publication in *Information and Computation* [64]. Chapter 3 adapts the reduction technique to construct decision procedures for the theory of queues with integer arithmetic. A part of this work was published in FSTTCS'05 [63]. Chapter 4 presents the proof of decidability of the first-order theory of Knuth-Bendix order using quantifier elimination and the reduction technique developed for the theory of term algebras with integer arithmetic. This result was published in CADE'05 [62]. Chapter 5 concludes the thesis with a discussion of future work.

Chapter 2

Term Algebras with Integers

01:14:08:650

> Oracle : You know what that means? It's Latin. Means, "Know Thyself." I'll let you in on a little secret. Being the One is just like being in love. No one can tell you you're in love. You just know it. Through and through . . .
>
> —*The Matrix*.

Terms "know themselves" in the sense that they can describe themselves formally in algebraic structures, called term algebras. Term algebras can model a class of recursively defined data structures such as lists, trees and records, which are widely used in programming languages. In this chapter we extend the theory of term algebras with integer functions which map terms to integers, resulting in a combined theory of term algebras and Presburger arithmetic. We present decision procedures for quantifier-free theories in structures with an infinite constant domain and with a finite constant domain. We also present a quantifier elimination procedure for the extended first-order theory that can remove a block of existential quantifiers in one step.

2.1 Introduction

Recursively defined data structures are essential constructs in programming languages. Intuitively, a data structure is *recursively defined* if it is partially composed of smaller or simpler instances of the same structure. Examples include lists, stacks, counters, trees, records and queues. To verify programs containing recursively defined data structures we must be able to reason about these data structures. Decision procedures for several data structures exist. However, in program verification decision procedures for a single theory are usually not applicable as programming languages often involve multiple data domains, resulting in verification conditions that span multiple theories. Common examples of such *mixed* constraints are combinations of data structures with integer constraints on the size of those structures. Such constraints can express memory safety properties such as absence of memory overflow and out-of-bound array access, which are crucial to program correctness.

In this chapter we consider the integration of Presburger arithmetic with term algebras which can represent an important class of recursively defined data structures known as *recursive data structures*. This class of structures satisfies the following two properties of term algebras: (i) the data domain is the set of data objects generated exclusively by applying constructors, and (ii) each data object is uniquely generated. Examples of such structures include lists, stacks, counters, trees and records; queues do not belong to this class as they are not uniquely generated: they can grow at both ends.

Our language of the integrated theory has two sorts; the integer sort $\mathbb{Z}$ and the term sort $\mathbb{T}$. The language is the set-theoretic union of the language of term algebras and the language of Presburger arithmetic plus the additional length function $|\cdot| : \mathbb{T} \to \mathbb{N}$. Formulas are formed from *term literals* and *integer literals* in the usual way. Term literals are exactly the literals in the theory of term algebras. Integer literals are those that can be built up from primitive integer terms (the length function applied to Σ-terms), addition and the other usual arithmetic functions and relations.

We present decision procedures for the quantifier-free and the first-order theory of term algebras with length function and integer constraints, for structures with both finite and infinite constant domain. In the rest of the chapter we will use the following notation for these theories. $\mathrm{Th}^{\forall}(\mathrm{TA})$ and $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ denote the quantifier-free theory of, respectively, pure term algebras and term algebras with a length function and Presburger arithmetic constraints. Similarly, $\mathrm{Th}(\mathrm{TA})$ and $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$ denote the full first-order theory of pure term algebras and term algebras with a length function and Presburger arithmetic constraints. When we separately consider decision procedures for structures with infinite constant domain, we add an ω superscript, for example, $\mathrm{Th}^{\forall}(\mathrm{TA}^{\omega}_{\mathbb{Z}})$.

The decision procedures for $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ are based on Oppen's algorithm for acyclic recursive data structures with infinite data domain (which, essentially, is $\mathrm{Th}^{\forall}(\mathrm{TA}^{\omega})$) [45]. To decide satisfiability of a term constraint φ, Oppen's procedure constructs a DAG for φ, extracts from this DAG all implied equalities between terms, and then checks for inconsistencies with disequalities in φ. We extend

this procedure to $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ by extracting an implied *length constraint* from the term constraint. We show that for structures with infinite constant domain such a length constraint, which is satisfiable if and only if the term constraint φ is satisfiable, can be effectively computed and is expressible by a quantifier-free Presburger formula linear in the size of φ. For structures with finite constant domain we introduce an additional *counting constraint* to account for the fact that with finitely many constants the number of distinct terms of a particular length is bounded. We show that also this counting constraint is expressible by a quantifier-free Presburger formula. The latter decision procedure directly extends Oppen's decision procedure for infinite data domains to finite domains. As a case study, we analyze the red-black tree algorithm using $\mathrm{Th}^{\forall}(\mathrm{RB}_{\mathbb{Z}})$, the theory of a term algebra with two integral functions. This serves as an example on how to apply our method to theories of term algebras with more than one integral function.

For the first-order theory, we first present a new quantifier elimination procedure for Th(TA) and then extend it to an elimination procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$. Our elimination procedure for Th(TA) is based on the elimination procedure in [19], but can eliminate blocks of quantifiers of the same kind in one step. We extend it to a decision procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$ by, again, extracting integer constraints from term constraints combined with a reduction of quantifiers on term variables to quantifiers on integer variables.

The decision procedures for $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ and $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$ were first published, without proofs, in [60]. The improved version that allows elimination of blocks of quantifiers was published in [61]. In that paper we showed that the complexity of our decision procedures was $2k$-fold exponential for k quantifier alternations for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$. This chapter provides an extended presentation of the results in both papers, improves the complexity of the decision procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$ to k-fold exponential for k quantifier alternations, and includes all the proofs.

Related Work and Comparison. Our component theories are both decidable. Presburger arithmetic was first shown to be decidable in 1929 by quantifier elimination [16]. A more efficient algorithm was later discovered by Cooper [12] and further improved by Reddy and Loveland [46].

It is well-known that recursive data structures can be modeled as term algebras which were shown to be decidable by Malcev using quantifier elimination [36]. This result was proved again several times in different settings [34, 9, 19, 8, 3, 49, 30, 29, 60].

Quantifier elimination has been used to obtain decidability results for various extensions of term algebras. Maher showed the decidability of the theory of infinite and rational trees [34]. Comon and Delor presented an elimination procedure for term algebras with membership predicate in the regular tree language [8]. Backofen presented an elimination procedure for structures of feature trees with arity constraints [3]. Rybina and Voronkov showed the decidability of term algebras with queues [49]. Kuncak and Rinard showed the decidability of term powers, which are term algebras augmented with coordinate-wise defined predicates [30].

Decision procedures for the quantifier-free theory of recursive data structures were discovered by Nelson, Oppen et al. [40, 45, 15]. Oppen gave a linear algorithm for acyclic structures [45] and (with Nelson) a quadratic algorithm for cyclic structures [40]. If the values of the selector functions on constants are specified, then the problem is NP-complete [45].

A general combination method for decision procedures for quantifier-free theories was developed by Nelson and Oppen in 1979 [39]. The method requires that component theories be loosely coupled, that is, have disjoint signatures, and are stably infinite[1] [53]. Tinelli and Ringeissen presented a general theoretical framework for combining satisfiability procedures of theories with non-disjoint signatures [54]. Tinelli and Zarba generalized Nelson-Oppen's method to theories in multisorted languages [55]. Armando, Ranise and Rusinowitch presented a uniform framework using superposition for deriving decision procedures for certain combined theories [1]. Ghilardi presented a set of model-theoretical conditions for the existence of Nelson-Oppen combination schema on theories having non-disjoint signatures [18]. But none of these general purpose combination methods are applicable to the combination of our component theories, which is a multisorted theory with a function mapping elements in one sort to another.

Zarba constructed decision procedures for a combined theory of sets and integers and a theory of multisets and integers, respectively [59, 58]. The integration of Presburger arithmetic with recursive data structures was discussed by Bjørner [5] and an incomplete procedure was implemented in STeP (Stanford Temporal Prover) [6].

Integer constraints not only arise in the combination of decision procedures, but they are also useful as an auxiliary extension to encode properties on data structures. This line of investigation goes back to Skolem who showed the decidability of the first-order theory of Boolean algebras by reducing constraints on sets to constraints on the cardinality of sets [51]. It readily follows from the reduction technique that the first order theory of sets with cardinality constraints in Presburger arithmetic is decidable [17]. Recently, Revesz [47], and Kuncak and Rinard [31] independently presented decision procedures for this theory. A combination of Presburger arithmetic and term algebras was used by Korovin and Voronkov to show that the quantifier-free theory of term algebras with Knuth-Bendix order is NP-complete [26, 27]. Along this line of investigation we proved the decidability of the first-order theory of Knuth-Bendix orders [62] using quantifier elimination (Chapter 4). The elimination procedure makes extensive use of Presburger arithmetic in the reduction of quantifiers on term variables to quantifiers on integer variables.

Chapter Organization. Section 1.3 presents the notation and terminology. Section 2.2 introduces the language and structure of term algebras. Section 2.3 describes Oppen's algorithm for recursive data structures. Section 2.4 presents our decision procedures for the quantifier-free theory of term algebras augmented with a length function and Presburger arithmetic. In Section 2.4.1 we

[1] A theory is stably infinite if a quantifier-free formula in the theory is satisfiable if and only if it is satisfiable in an infinite model.

first describe the theory and then in Section 2.4.2 we outline our approach for constructing the decision procedures by introducing the concepts of implied length constraints and presenting a generic decision procedure. In Section 2.4.3 we specialize this procedure for structures with infinite constant domain, and in Section 2.4.4 we refine it further for structures with finite constant domain. Section 2.4.5 outlines the approach to obtain decision procedures for structures whose constant domain contains relations besides equality. Section 2.4.6 discusses the complexity of the decision procedures for the quantifier-free theories. In Section 2.5 we extend our method to a theory of a term algebra with two integral functions to analyze the red-black tree algorithm.

Section 2.6 presents a new decision procedure for the first-order theory of term algebras. Section 2.7 presents the decision procedures for the first-order theory of term algebras with integer constraints. We first introduce the technical machinery for the construction of a quantifier elimination procedure and then, in Section 2.7.2, we extend the elimination procedure for term algebras presented in Section 2.6 to term algebras with integers. Section 2.7.3 further generalizes the result to structures whose constant domain has an internal structure and admits quantifier elimination. In Section 2.7.4 we discuss how this procedure can be adapted for theories with infinite languages. Section 2.8 concludes with some ideas for future work. Most proofs are provided in Section 2.9.

2.2 Term Algebras

We present a general language and structure of term algebras. For simplicity, we do not distinguish syntactic terms in the language from semantic terms in the corresponding structure. The meaning should be clear from the context.

Definition 2.1 (Term Algebras). *A* term algebra TA : $\langle\, \mathbb{T}; \mathcal{C}, \mathcal{A}, \mathcal{S}, \mathcal{T} \,\rangle$ *consists of*

1. $\mathbb{T}$*: The* term domain*, which exclusively consists of terms recursively built up from constants by applying non-nullary constructors. Objects in* $\mathbb{T}$ *are called* TA-terms*. The type of a term* t*, denoted by* $\mathrm{type}(t)$*, is the outermost constructor symbol of* t*. We say that* t *is* α*-typed (or is an* α*-term) if* $\mathrm{type}(t) = \alpha$*.*

2. $\mathcal{C}$*: A set of constructors:* $\alpha, \beta, \gamma, \ldots$ *The arity of* α *is denoted by* $\mathrm{ar}(\alpha)$*.*

3. $\mathcal{A}$*: A set of constants:* $a, b, c, \ldots$ *We require* $\mathcal{A} \neq \emptyset$ *and* $\mathcal{A} \subseteq \mathcal{C}$*. For* $a \in \mathcal{A}$*,* $\mathrm{ar}(a) = 0$ *and* $\mathrm{type}(a) = a$*.*

4. $\mathcal{S}$*: A set of selectors. For a constructor* α *with arity* $k > 0$*, there are* k *selectors* $\mathsf{s}_1^\alpha, \ldots, \mathsf{s}_k^\alpha$ *in* $\mathcal{S}$*. We call* s_i^α $(0 < i \leq k)$ *the* i^{th} α*-selector. For a term* x*,* $\mathsf{s}_i^\alpha(x)$ *returns the* i^{th} *component of* x *if* x *is an* α*-term and* x *itself otherwise.*

5. $\mathcal{T}$*: A set of testers. For each constructor* α *there is a corresponding tester* Is_α*. For a term* x*,* $\mathsf{Is}_\alpha(x)$ *is true if and only if* x *is an* α*-term. For a constant* a*,* $\mathsf{Is}_a(x)$ *is just* $x = a$*. In addition there is a special tester* Is_A *such that* $\mathsf{Is}_\mathsf{A}(x)$ *is true if and only if* x *is a constant.*

We use $\mathscr{L}_{\mathbb{T}}$ to denote the language of term algebras. We use TA^{ω} *to denote a term algebra with an infinite constant domain.*

A term algebra has two essential properties: (i) the domain $\mathbb{T}$ is *exclusively* generated by recursively applying constructors; (ii) each object in $\mathbb{T}$ is *uniquely* constructed. Note that the term domain $\mathbb{T}$ is the *ground* Herbrand domain $\mathcal{T}(C)$ in the language consisting of only constructors (i. e., $\Sigma = C$). Selectors and testers are introduced into the *formal* language for our study. A more general definition of term algebras would include a variable base $\mathcal{X}$. For our purpose of modeling recursive data structures, however, it suffices to assume the term domain only consists of ground terms; i. e., the base $\mathcal{X} = \emptyset$. Nevertheless, our method can be modified to deal with a non-ground term domain by treating variables as special constants.

For simplicity, in the rest of this chapter we assume that $\mathscr{L}_{\mathbb{T}}$ is finite except in Section 2.4.3 where we present an algorithm for structures with an infinite constant domain as the basis for algorithms for structures with a finite constant domain. The main techniques used for finite languages, however, can be easily generalized to handle the case of infinite languages. Actually, the decision problems become considerably easier if we allow $\mathscr{L}_{\mathbb{T}}$ to have infinitely many constants. We defer the detailed discussion to Section 2.7.4.

Example 2.1 (LISP Lists). *Consider the LISP list structure*

$$\mathrm{List} = \langle\ \mathrm{list};\ \{\mathrm{cons}, \mathrm{nil}\},\ \{\mathrm{nil}\},\ \{\mathrm{car}, \mathrm{cdr}\},\ \{\mathrm{Is}_{\mathrm{cons}}, \mathrm{Is}_{\mathrm{nil}}, \mathrm{Is}_{\mathsf{A}}\}\ \rangle$$

where list *denotes the domain,* nil *denotes the empty list,* cons *is a binary constructor (pairing function) and* car *and* cdr *are the corresponding left and right selectors (projectors) respectively. It is not difficult to verify that* List *is an instance of term algebras.*

The theory of term algebras is axiomatizable. Let $\bar{z}_\alpha$ denote $z_1, \ldots, z_{\mathrm{ar}(\alpha)}$. The following formula schemes, in which variables are implicitly universally quantified over $\mathbb{T}$, axiomatize Th(TA).

A1. $t(x) \neq x$, if t is built solely by constructors and t properly contains x.

A2. $\alpha(x_1, \ldots, x_{\mathrm{ar}(\alpha)}) \neq \beta(y_1, \ldots, y_{\mathrm{ar}(\beta)})$, if $\alpha, \beta \in C$ and $\alpha \not\equiv \beta$.

A3. $\alpha(x_1, \ldots, x_{\mathrm{ar}(\alpha)}) = \alpha(y_1, \ldots, y_{\mathrm{ar}(\alpha)}) \rightarrow \bigwedge_{1 \leq i \leq \mathrm{ar}(\alpha)} x_i = y_i$.

A4. $\mathrm{Is}_\alpha(x) \leftrightarrow \exists \bar{z}_\alpha \alpha(\bar{z}_\alpha) = x$ for $\alpha \in C$. In particular, $\mathrm{Is}_a(x) \leftrightarrow x = a$ for $a \in \mathcal{A}$.

A5. $\mathrm{Is}_{\mathsf{A}}(x) \leftrightarrow \bigvee_{a \in \mathcal{A}} \mathrm{Is}_a(x)$.

A6. $\mathsf{s}_i^\alpha(x) = y \leftrightarrow \exists \bar{z}_\alpha \Big(\alpha(\bar{z}_\alpha) = x \wedge y = z_i)\Big) \vee \Big(\forall \bar{z}_\alpha (\alpha(\bar{z}_\alpha) \neq x) \wedge x = y\Big)$.

This set of axioms is a variant of the axiomatization given in [19]. In general, selectors and testers can be defined by constructors and vice versa. One direction has been shown by (A4), (A5)

and (A6), which are purely definitional axioms. The other direction follows from the equivalence of $\bigwedge_{i=1}^{k} \mathsf{s}_i^\alpha(x) = x_i \wedge \mathrm{Is}_\alpha(x)$ and $x = \alpha(x_1, \ldots, x_k)$.

We write $\alpha = (\mathsf{s}_1^\alpha, \ldots, \mathsf{s}_k^\alpha)$ to indicate that α is a non-nullary constructor with $\mathrm{ar}(\alpha) = k$ and $\mathsf{s}_1^\alpha, \ldots, \mathsf{s}_k^\alpha$ are the corresponding selectors of α. A term t is called a *constructor term* if t is a variable or the outermost function symbol of t is a constructor. A constructor term not containing selectors is called *pure*. For example, constants are pure constructor terms. A term t is called a *selector term* if either t is a variable or the outermost function symbol of t is a selector. Note that variables are both constructor terms and selector terms. We assume that no constructors appear immediately inside selectors as simplification can always be done. For example, $\mathsf{s}_i^\alpha(\alpha(x_1, \ldots, x_k))$ simplifies to x_i $(0 < i \leq k)$ and $\mathsf{s}_j^\beta(\alpha(x_1, \ldots, x_k))$ simplifies to $\alpha(x_1, \ldots, x_k)$ for $\alpha \not\equiv \beta$. As a consequence, a selector term has the form $\mathsf{s}_1(\cdots(\mathsf{s}_n(x)\cdots))$ for $n \geq 0$. We use $L, F, G, H, \ldots$ to denote (possibly empty) selector sequences. So $\mathsf{s}_1(\cdots(\mathsf{s}_n(x)\cdots))$ can be abbreviated as Lx for $L = \mathsf{s}_1, \ldots, \mathsf{s}_n$. The *depth* of x in Lx is $|L|$, the length of L. The depth of x in a formula φ is the maximum depth of x in the selector terms in φ, denoted by $\mathrm{depth}^\varphi(x)$. We use $\mathrm{Is}_\alpha(t_1, \ldots, t_n)$ as an abbreviation for $\bigwedge_{0<i\leq n} \mathrm{Is}_\alpha(t)$. We say a selector term $\mathsf{s}_i^\alpha(t)$ is *proper* in a formula φ if $\mathrm{Is}_\alpha(t)$ is a conjunct of φ. We can make selector terms proper with type information.

Definition 2.2 (Type Completeness). *A conjunction of literals Φ is* type complete *if for any selector term t occurring in Φ, exactly one type of tester predicate $\mathrm{Is}_\alpha(t)$ $(\alpha \in C \setminus \mathcal{A} \cup \{\mathsf{A}\})$ is a conjunct of Φ.*

For a type complete Φ containing a term t, we write $\mathrm{type}(t) = \alpha$ to indicate that $\mathrm{Is}_\alpha(t)$ is a conjunct of Φ. A type complete Φ can be simplified so that any selector term is proper.

Example 2.2 (Type Completeness). *Let us consider in* List *the type complete constraint*

$$y \neq \mathrm{cons}(x, \mathrm{car}(x)) \ \wedge \ \mathrm{Is}_{\mathsf{A}}(x, \mathrm{car}(x)) \ \wedge \ \mathrm{Is}_{\mathrm{cons}}(y) \ . \tag{2.1}$$

Thanks to the type information, it can be simplified to

$$y \neq \mathrm{cons}(x, x) \ \wedge \ \mathrm{Is}_{\mathsf{A}}(x) \ \wedge \ \mathrm{Is}_{\mathrm{cons}}(y) \ . \tag{2.2}$$

We could have defined the notion of type completeness only for terms that occur inside selectors. In this way, a type complete formula may contain terms of unspecified types; they are either variables or selector terms that are not embedded inside selectors. This will lead to more efficient algorithms in practice. We choose the above definition, however, because it simplifies descriptions of algorithms presented in the following sections, and in addition, it does not affect the worst-case complexity for those algorithms.

Given a quantifier-free formula φ, we can obtain a type complete φ' from φ by adding exactly one tester predicate $\mathrm{Is}_\alpha(t)$ $(\alpha \in C)$ for each term t occurring in φ. We call φ' so obtained a *type completion* of φ.

Example 2.3 (Type Completion). *Let us revisit Example 2.2. It is easily seen that* (2.1) *is a type completion of* $y \neq \mathrm{cons}(x, \mathrm{car}(x))$.

Example 2.4 (Type Completion and Simplification). *Let* $\alpha, \beta \in C$, $\alpha \not\equiv \beta$ *and* $\alpha = (\mathsf{s}_1^\alpha)$. *A possible type completion for* $y = \mathsf{s}_1^\alpha(x)$ *is*

$$y = \mathsf{s}_1^\alpha(x) \ \wedge \ \mathrm{Is}_\beta(x, \mathsf{s}_1^\alpha(x), y) \ , \tag{2.3}$$

which, by Axioms (A4) and (A6), simplifies to $y = x \wedge \mathrm{Is}_\beta(x, y)$. *Another type completion is*

$$y = \mathsf{s}_1^\alpha(x) \ \wedge \ \mathrm{Is}_\alpha(x) \ \wedge \ \mathrm{Is}_\beta(\mathsf{s}_1^\alpha(x), y) \ , \tag{2.4}$$

in which the selector term $\mathsf{s}_1^\alpha(x)$ *is proper and no simplification is possible. As a third possibility, we can have the type completion*

$$y = \mathsf{s}_1^\alpha(x) \ \wedge \ \mathrm{Is}_\alpha(x, \mathsf{s}_1^\alpha(x)) \ \wedge \ \mathrm{Is}_\beta(y) \ , \tag{2.5}$$

which simplifies to false *because of type conflicts.*

As shown by the above example, a type completion of a satisfiable formula may be contradictory due to type conflicts. A type completion φ' of φ is *compatible* with φ if the satisfiability of φ implies the satisfiability of φ'. Obviously, φ is satisfiable if and only if it has a satisfiable compatible type completion.

In the following sections, we present nondeterministic algorithms that rely on the successful guess of a satisfiable compatible type completion. Unless stated otherwise, we assume that any quantifier-free formula is type complete, and all occurring selector terms are simplified to proper ones. We assume that equalities (resp. disequalities) between terms with conflicting types are simplified to false (resp. true). For example, in List the appearance of $\mathrm{car}(x) \neq y$ should be read as $\mathrm{car}(x) \neq y \wedge \mathrm{Is}_{\mathrm{cons}}(x)$. We also assume that formulas do not have conflicting type literals. For example, we never encounter formulas containing subformulas of the form

$$x \neq \alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha)}) \ \wedge \ x \neq \beta(t'_1, \ldots, t'_{\mathrm{ar}(\beta)})$$

for $\alpha \not\equiv \beta$, because at least one conjunct would have been simplified to true. But to save notation, we omit test literals and treat them as implicit side conditions.

2.3 Decision Procedures for $\mathrm{Th}^\forall(\mathrm{TA})$

This section and the next section present decision procedures for quantifier-free theories. All our decision procedures for quantifier-free theories are *refutation-based*; to determine the validity of a

formula φ, a procedure determines the unsatisfiability of $\neg\,\varphi$, which further reduces to determining the unsatisfiability of each disjunct in the *disjunctive normal form* of $\neg\,\varphi$. Henceforth, in discussions related to quantifier-free theories, a quantifier-free formula (or an *input formula*) always refers to a conjunction of literals. To emphasize this, we use capital symbols $\Phi, \Psi, \ldots$ to denote conjunctions of literals. We identify a conjunction of literals Φ with the set of all its conjuncts. By $A \in \Phi$ we mean A is a conjunct of Φ and by $\Phi \cup A$ (or more formally $\Phi \cup \{A\}$) we mean $\Phi \wedge A$. We present algorithms in a nondeterministic manner; by saying "guess" φ, we mean to split on a disjunction that is valid in the context and contains φ as one of the disjuncts, and we work on each resultant constraint "simultaneously".

In [45] Oppen presented a decision procedure for the quantifier-free theory of acyclic recursive data structures which is essentially $\mathrm{Th}^{\forall}(\mathrm{TA})$. The basic idea of the decision procedure is to generate all equalities implied by the input formula and check for inconsistencies with disequalities given in the input. The decision procedure relies on the fact that $\mathrm{Th}^{\forall}(\mathrm{TA})$ is convex in a language without selectors. In fact, the convexity was shown implicitly in the correctness proof of Oppen's algorithm [45].

Definition 2.3 (Convexity Theory). *A theory is* convex *if whenever a conjunction of literals implies a disjunction of atomic formulas, it also implies one of the disjuncts.*

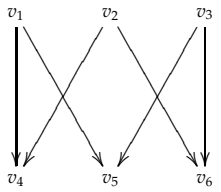
Let Φ be a conjunction of equalities and Ψ a disjunction of equalities. The convexity of $\mathrm{Th}^{\forall}(\mathrm{TA})$ can be rephrased as follows: $\Phi \wedge \neg\,\Psi$ is satisfiable if and only if for each of conjuncts $s \neq t \in \neg\,\Psi$, $\Phi \wedge s \neq t$ is satisfiable, i. e., $\Phi \not\models s = t$. The idea of Oppen's algorithm is to discover all logically implied equalities (between terms in Φ and Ψ) by constructing a directed acyclic graph (DAG) with terms as vertexes and computing an equivalence relation on the nodes based on equality of children and ancestor nodes (bidirectional closure), formally described below.

Definition 2.4 (DAG Representation). *A term t can be naturally represented by an ordered tree T_t such that (i) if t is a constant, then T_t is a leaf vertex labeled by t, and (ii) if t is in the form $\alpha(t_1, \ldots, t_k)$, then T_t is the tree with root labeled by α and $T_{t_1}, \ldots, T_{t_k}$ as its subtrees. A* directed acyclic graph *(DAG) G_t of t is obtained from T_t by merging all identical subgraphs of T_t.*

The *DAG of a formula* is the DAG representing all terms in the formula. For example, Figure 2-1 shows the DAG for

$$\mathrm{cons}(y, z) = \mathrm{cons}(x, z) \;\wedge\; \mathrm{cons}(x, y) \neq z \;. \tag{2.6}$$

We assume DAGs are *sibling complete* in the sense that a node and all of its siblings coexist. For example, $\mathrm{car}(x)$ appears if and only if $\mathrm{cdr}(x)$ does. A *sibling completion* can be easily obtained by adding trivial equality literals like $t = t$ to the original formula. For a vertex u, let $\delta(u)$ denote the outgoing degree and $u[i]$ $(1 \leq i \leq \delta(u))$ the i^{th} successor of u.



$v_1 : \text{cons}(x, y)$ $\quad$ $v_4 : x$
$v_2 : \text{cons}(x, z)$ $\quad$ $v_5 : y$
$v_3 : \text{cons}(y, z)$ $\quad$ $v_6 : z$

Figure 2-1: The DAG of $\text{cons}(y, z) = \text{cons}(x, z) \ \wedge \ \text{cons}(x, y) \neq z$.

Let R be a binary relation on the vertexes of a DAG and let u, v be any two vertexes.

Definition 2.5 (Unification Closure). *We say that R' is the* unification closure *of R (denoted by $R{\downarrow}$) if R' is the smallest equivalence relation extending R such that $(u, v) \in R'$ and* $\text{type}(u) = \text{type}(v)$ *implies $(u[i], v[i]) \in R'$, for every $i \in [1, \delta(u)]$.*

Definition 2.6 (Congruence Closure). *We say that R' is the* congruence closure *of R (denoted by $R{\uparrow}$) if R' is the smallest equivalence relation extending R such that $(u[i], v[i]) \in R'$ (for every $i \in [1, \delta(u)]$) and* $\text{type}(u) = \text{type}(v)$ *implies $(u, v) \in R'$.*

If R' is both unification and congruence closed (with respect to R), we call it the *bidirectional closure*, denoted by $R{\downarrow}{\uparrow}$.

Let R be the set of all pairs asserted equal in Φ. It was shown that $R{\downarrow}{\uparrow}$ represents all equalities logically implied by Φ [45]. Therefore, Φ is unsatisfiable if and only if there exist t and s such that $t \neq s \in \Phi$ and $(t, s) \in R{\downarrow}{\uparrow}$.

Algorithm 2.1 (Oppen's Decision Procedure for $\text{Th}^{\forall}(\text{TA})$ **[45]).** *Input:*

$$\Phi : \ q_1 = r_1 \ \wedge \ \cdots \ \wedge \ q_k = r_k \ \wedge \ s_1 \neq t_1 \ \wedge \ \cdots \ \wedge \ s_l \neq t_l \ ,$$

where q_i, r_i, s_i and t_i are pure constructor terms.

1. *Construct the DAG G of Φ.*
2. *Compute on G the bidirectional closure $R{\downarrow}{\uparrow}$ of $R = \{(q_i, r_i) \mid 0 < i \leq k\}$.*
3. *Return* FAIL *if $\exists i (s_i, t_i) \in R{\downarrow}{\uparrow}$, or $(\exists (u, v) \in R{\downarrow}{\uparrow})[\text{type}(u) \neq \text{type}(v)]$.*
   Return SUCCESS *otherwise.*

Oppen's original algorithm is given for the theory of LISP lists, which only has one non-nullary constructor (cons). It is straightforward, however, to generalize it to term algebras with an arbitrary number of non-nullary constructors; in Algorithm 2.1, we added type checking which is not present in Oppen's original algorithm.

In our setting, the language contains selectors and values of α-selectors on non α-terms are specified, e.g., $s_i^{\alpha}(x) = x$ if x is not an α-term. It was shown that for such structures the decision

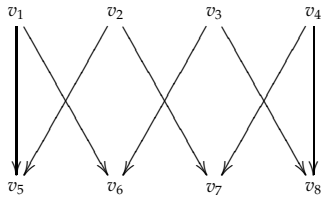


Figure 2-2: The DAG of $\mathrm{cons}(y,z) = \mathrm{cons}(\mathrm{cdr}(x),z) \ \wedge \ \mathrm{cons}(\mathrm{car}(x),y) \neq x \ \wedge \ \mathrm{Is}_{\mathrm{cons}}(x)$.

problem is NP-complete [40]. The complication is that it is not known *a priori* whether $\mathsf{s}(x)$ is a proper subterm of x and hence it is not possible to use the DAG representation directly. A solution to this problem is to guess the type information of terms occurring inside selectors before applying Algorithm 2.1.

Algorithm 2.2 (Decision Procedure for $\mathrm{Th}^{\forall}(\mathrm{TA})$ with Selectors).
Input: Φ*, a conjunction of equalities and disequalities.*

1. *Guess a type completion* Φ^{c} *of* Φ *and simplify selector terms accordingly.*

2. *Call Algorithm 2.1 on* Φ^{c}.

Example 2.5 (Oppen's Algorithm). *Figure 2-2 shows the DAG representation of the LISP list formula*

$$\mathrm{cons}(y,z) = \mathrm{cons}(\mathrm{cdr}(x),z) \ \wedge \ \mathrm{cons}(\mathrm{car}(x),y) \neq x \tag{2.7}$$

assuming that $\mathrm{Is}_{\mathrm{cons}}(x)$ *is present. Initially* $R = \{(v_3, v_4)\}$ *as* v_3 *and* v_4 *are asserted equal in* (2.7)*. (For simplicity reflexive pairs are not listed.) By the unification algorithm* (v_6, v_7) *are merged, which gives* $R\downarrow = \{(v_3, v_4), (v_6, v_7)\}$*. Then by the congruence algorithm* (v_1, v_2) *are merged, resulting in*

$$R\downarrow\uparrow = \{\, (v_1, v_2), (v_3, v_4), (v_6, v_7) \,\} \ .$$

(Here we used the property that $R\downarrow\uparrow = (R\downarrow)\uparrow$*.) Obviously this branch fails as* $v_1 \neq v_2$ *is asserted by* (2.7)*. The remaining branch (in the presence of* $\mathrm{Is}_{\mathsf{A}}(x)$*) simplifies to* $\mathrm{Is}_{\mathsf{A}}(x) \wedge x = y$ *which is clearly satisfiable, and therefore so is* (2.7)*.*

Note that the correctness of both Algorithm 2.1 and 2.2 relies on the (implicit) assumption that the constant domain is infinite, since otherwise the theory is not convex. As a counter-example, for the structure TA with domain $\mathcal{A} = \{a, b\}$, we have $\mathrm{Is}_{\mathsf{A}}(x) \models x = a \vee x = b$, but neither $\mathrm{Is}_{\mathsf{A}}(x) \models x = a$ nor $\mathrm{Is}_{\mathsf{A}}(x) \models x = b$. We shall see (in Section 2.4.4) that our algorithm extends Oppen's algorithm to structures with finite constant domain.

2.4 Decision Procedures for Quantifier-free Theories

In this section we present decision procedures for $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$, the quantifier-free theory of term algebras augmented with a length function and Presburger arithmetic. First, in Section 2.4.1 we describe the theory, and then, in Section 2.4.2, outline our generic approach for constructing decision procedures for this theory. This approach is refined in Section 2.4.3 into a decision procedure for structures with an infinite constant domain, $\mathrm{Th}^{\forall}(\mathrm{TA}^{\omega}_{\mathbb{Z}})$ and then further refined for structures with finite constant domain, $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ in Section 2.4.4. Section 2.4.5 outlines the approach to obtain decision procedures for structures whose constant domain contains relations besides equality. Section 2.4.6 discusses the complexity of the decision procedures for the quantifier-free theories.

2.4.1 Term Algebras with Integers

Definition 2.7 (Term Algebras with Integers). *The structure of term algebras with integers is*

$$\mathrm{TA}_{\mathbb{Z}} = \langle\, \mathrm{TA};\, \mathrm{PA};\, |\cdot| : \mathbb{T} \to \mathbb{N} \,\rangle \ ,$$

where TA *is a term algebra,* PA *is Presburger arithmetic, and* $|\cdot|$ *is the length function defined recursively by*

1. *for any constant* a, $|a| = 1$, *and*
2. *for a term* $\alpha(t_1, \ldots, t_k)$, $|\alpha(t_1, \ldots, t_k)| = 1 + \sum_{i=1}^{k} |t_i|$.

The extended language is denoted by $\mathscr{L}^{\mathbb{Z}}_{\mathbb{T}}$. *We use* $\mathrm{TA}^{\omega}_{\mathbb{Z}}$ *to denote a term algebra with integers and with an infinite constant domain.*

The length function given in Definition 2.7 was chosen for ease of presentation. Generalizing it into a weight function that assigns an arbitrary nonnegative integer to each symbol, or a height function that gives the length of the maximum path does not require any essential changes to our techniques. In fact we can have more than one length function. An example is given in Section 2.5.

We use subscripts $\mathbb{T}$, $\mathbb{Z}$ (or prefixes TA-, PA-) to denote notions related to term sort and integer sort, respectively. For example, $\Phi_{\mathbb{T}}$ denotes a formula in $\mathscr{L}_{\mathbb{T}}$, the language of pure term algebras, $\Phi_{\mathbb{Z}}$ denotes a formula in $\mathscr{L}_{\mathbb{Z}}$, the language of Presburger arithmetic, $\mathcal{V}_{\mathbb{T}}$ denotes the collection of variables in $\mathscr{L}_{\mathbb{T}}$ and $\mathcal{V}_{\mathbb{Z}}$ denotes the collection of variables in $\mathscr{L}_{\mathbb{Z}}$. Although $\mathscr{L}^{\mathbb{Z}}_{\mathbb{T}}$ contains two equality predicates: *term equality* $=_{\mathbb{T}}$ in $\mathscr{L}_{\mathbb{T}}$ and *integer equality* $=_{\mathbb{Z}}$ in $\mathscr{L}_{\mathbb{Z}}$, we use $=$ in both cases unless there is a chance of confusion. Recall that we do not distinguish syntactic TA-terms in $\mathscr{L}_{\mathbb{Z}}$ and semantic TA-terms in $\mathbb{T}$. We also use "term" for "TA-" when there is no confusion. For example, by *term variables* we mean TA-variables and by *term quantifiers* we mean quantifiers on term variables.

A TA-term can occur inside the length function. Such occurrence is called an *integer occurrence* to be distinguished from the normal *term occurrence*. From now on, we freely use integer terms $|t|$

to form Presburger formulas. For example, car(x) is a TA-term and |car(x)| is a PA-term. The first occurrence of car(x) is a term occurrence and the second one is an integer occurrence.

To save space, we use an integer term $|t(\bar{\mathbf{x}})|$ in two ways; one as the length function $|\cdot|$ applied to $t(\bar{\mathbf{x}})$ (when $t(\bar{\mathbf{x}})$ is in discussion), and the other as a special integer variable (called *pseudo integer variable*). In the latter case, if $\bar{\mathbf{x}} \in \mathcal{V}_{\mathbb{T}}$, then $|\bar{\mathbf{x}}| = \bar{\mathbf{z}}$ denotes the formula of the form $\bigwedge_i |t_i(\bar{\mathbf{x}})| = z_i$ where $\bar{\mathbf{z}} \in \mathcal{V}_{\mathbb{Z}}$ and $t_i(x)$ enumerates all TA-terms containing x in the context.

Example 2.6 (Notation for Pseudo Integer Variables). *In the context of the formula* $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) : x_1 = \text{cons}(x_2, x_3)$, $|\bar{\mathbf{x}}| = \bar{\mathbf{z}}$ *denotes*

$$|x_1| = z_1 \ \wedge \ |x_2| = z_2 \ \wedge \ |x_3| = z_3 \ \wedge \ |\text{cons}(x_2, x_3)| = z_4 \ .$$

In the context of the formula $\Phi_{\mathbb{Z}}(\bar{\mathbf{x}}) : |x_1| = |\text{car}(\text{cdr}(x_2))|$, $|\bar{\mathbf{x}}| = \bar{\mathbf{z}}$ *denotes*

$$|x_1| = z_1 \ \wedge \ |x_2| = z_2 \ \wedge \ |\text{cdr}(x_2)| = z_3 \ \wedge \ |\text{car}(\text{cdr}(x_2))| = z_4 \ .$$

Suppose $\Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is in a context in which all occurrences of $\bar{\mathbf{x}} \in \mathcal{V}_{\mathbb{T}}$ are integer occurrences. Then by $\Phi(\bar{\mathbf{z}}, \bar{\mathbf{y}})$ we mean the formula obtained by substituting a true integer variable z ($z \in \bar{\mathbf{z}}$) for each pseudo integer variable $|t(\bar{\mathbf{x}})|$. (Here we actually overload the symbol Φ, but the risk of confusion is minimal.) We use $|\bar{\mathbf{x}}| \leftarrow \bar{\mathbf{z}}$ to denote such a substitution. For example, in Example 2.6 $\Phi_{\mathbb{Z}}(\bar{\mathbf{z}})$ denotes $z_1 = z_4$. If $\sigma_{\mathbb{T}}$ is an assignment for $\mathcal{V}_{\mathbb{T}}$, then $|\sigma_{\mathbb{T}}|$ denotes the corresponding assignment for pseudo integer variables. For example, if $\Phi_{\mathbb{T}}$ is $x_1 = \text{cons}(x_2, x_3)$ and $\sigma_{\mathbb{T}}$ is

$$\{\, x_1 := \text{nil},\ x_2 := \text{cons}(\text{nil}, \text{nil}), x_3 := \text{nil} \,\} \ ,$$

then $|\sigma_{\mathbb{T}}|$ is

$$\{\, |x_1| := 1,\ |x_2| := 3,\ |x_3| := 1 \,\} \ .$$

It is easily seen that the general purpose combination method in [39] is not directly applicable to $\text{TA}_{\mathbb{Z}}$ due to the presence of the length function.

Example 2.7 (Hidden Length Constraints). *The constraints*

$$\Phi_{\text{list}} : \ x = \text{cons}(\text{car}(y), y) \ , \qquad\qquad \Phi_{\mathbb{Z}} : \ |x| < 2|\text{car}(x)|$$

are clearly satisfiable, respectively, in List *and* PA. *However, since* Φ_{list} *implies that* car(x) = car(y), x *contains two copies of* car(y) *and so its length should be at least two times the length of* car(x). *Therefore,* $\Phi_{\mathbb{Z}} \wedge \Phi_{\text{list}}$ *is unsatisfiable in* $\text{List}_{\mathbb{Z}}$.

A simple but crucial observation is that $\Phi_{\mathbb{T}}$ induces an *implicit* length constraint Φ_{Δ}, in addition to the *explicit* constraint $\Phi_{\mathbb{Z}}$ given in the input. The unsatisfiability is due to the fact that Φ_{Δ} contradicts $\Phi_{\mathbb{Z}}$. In Example 2.7, Φ_{list} in fact implies $|x| \geq 2|\text{car}(x)|$, resulting in a contradiction to $\Phi_{\mathbb{Z}}$.

Implicit length constraints are induced not only by the structures of objects, but also by the size of the constant domain.

Example 2.8 (More Hidden Length Constraints). *Consider the constraint*

$$\Phi_{\text{list}} : x \neq \text{cons}(\text{cons}(\text{nil}, \text{nil}), \text{nil}) \wedge x \neq \text{cons}(\text{nil}, \text{cons}(\text{nil}, \text{nil})) \ , \qquad \Phi_{\mathbb{Z}} : |x| = 5$$

in List *and in* PA*, respectively. Clearly, both are satisfiable in the respective structures. In the combined structure* $\text{List}_{\mathbb{Z}}$*, however, there are exactly two term trees with length* 5 *and* Φ_{list} *states that* x *is not equal to either of them. As a consequence,* Φ_{list} *implies* $\Phi_{\Delta} : |x| \neq 5$*, contradicting* $|x| = 5$*.*

Intuitively, if we can extract from $\Phi_{\mathbb{T}}$ the implicit Φ_{Δ} that exactly characterizes the solution set of $\Phi_{\mathbb{T}}$, then the satisfiability of $\Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$ reduces to the satisfiability of $\Phi_{\Delta} \wedge \Phi_{\mathbb{Z}}$. As a consequence, we can derive decision procedures for the combined theory by utilizing the decision procedures for PA and TA.

2.4.2 A Generic Decision Procedure for $\text{Th}^{\forall}(\text{TA}_{\mathbb{Z}})$

Given a term constraint $\Phi_{\mathbb{T}}$ our objective is to construct a Presburger formula Φ_{Δ}, called a *Length Constraint Completion*, that is satisfiable if and only if $\Phi_{\mathbb{T}}$ is satisfiable.

Definition 2.8 (Length Constraint Completion (LCC) in $\text{TA}_{\mathbb{Z}}$**).** *An* $\mathscr{L}_{\mathbb{Z}}$*-formula* $\Phi_{\Delta}(\bar{\mathbf{x}})$ *is a* length constraint completion *(LCC) for* $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})$ *if the following formulas are valid:*

$$(\forall \bar{\mathbf{x}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \rightarrow \ (\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big(\Phi_{\Delta}(\bar{\mathbf{z}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] \ , \tag{2.8}$$

$$(\forall \bar{\mathbf{z}} : \mathbb{Z}) \Big[\Phi_{\Delta}(\bar{\mathbf{z}}) \ \rightarrow \ (\exists \bar{\mathbf{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] \ . \tag{2.9}$$

Let Φ_{Δ} be an LCC for $\Phi_{\mathbb{T}}$. Condition (2.8) says that for any satisfying assignment $\sigma_{\mathbb{T}}$ of $\Phi_{\mathbb{T}}$, $|\sigma_{\mathbb{T}}|$ is a satisfying assignment for Φ_{Δ}. In other words, $|\cdot|$ maps a satisfying assignment for $\Phi_{\mathbb{T}}$ in TA to a satisfying assignment for Φ_{Δ} in PA. We say that Φ_{Δ} satisfying (2.8) is *sound* with respect to $\Phi_{\mathbb{T}}$. On the other hand, condition (2.9) says that for any satisfying assignment σ_{Δ} of Φ_{Δ} there exists a satisfying assignment $\sigma_{\mathbb{T}}$ of $\Phi_{\mathbb{T}}$ such that $|\sigma_{\mathbb{T}}| = \sigma_{\Delta}$. In other words, any satisfying assignment in PA is the image under $|\cdot|$ of a satisfying assignment in TA. We say that Φ_{Δ} satisfying (2.9) is *realizable* by $\Phi_{\mathbb{T}}$. In particular, if $\Phi_{\mathbb{T}}$ is unsatisfiable, then so is Φ_{Δ}.

Let Φ_{Δ} be a formula satisfying both (2.8) and (2.9). Let $\Phi_{\Delta+}$ and $\Phi_{\Delta-}$, respectively, be any two formulas satisfying (2.8) and (2.9) (when in place of Φ_{Δ}). If we identify these constraints with their

corresponding solution sets, we have

$$\Phi_{\Delta-} \subseteq \Phi_{\Delta} \subseteq \Phi_{\Delta+} \ . \tag{2.10}$$

Thus Φ_Δ is the exact projection of $\Phi_\mathbb{T}$ from TA to PA, while $\Phi_{\Delta+}$, $\Phi_{\Delta-}$ are over and under approximations of Φ_Δ respectively. Let Φ_Δ and $\Phi_{\Delta'}$ both be LLCs for $\Phi_\mathbb{T}$. By (2.10) we have $\Phi_{\Delta'} \subseteq \Phi_\Delta \subseteq \Phi_{\Delta'}$ and hence $\Phi_\Delta = \Phi_{\Delta'}$ (with respect to the corresponding solution sets). Therefore, for a term constraint there exists a unique LCC up to equivalence.

Example 2.9 (LCC in $\mathrm{List}_\mathbb{Z}$). *Consider in* List *the formulas* $\Phi_{\text{list}} : \text{cons}(x, y) = z$. *The constraint*

$$\Phi_{\Delta+} : \ |z| > |x| \ \wedge \ |z| > |y| \ \wedge \ |x| > 0 \ \wedge \ |y| > 0 \ \wedge \ 2 \nmid |x| \ \wedge \ 2 \nmid |y|$$

is sound but it is not realizable for Φ_{list}, *as the integer assignment*

$$\sigma_\Delta : \{ |x| := 3, \ |y| := 3, \ |z| := 4 \}$$

can not be realized. On the other hand, the constraint

$$\Phi_{\Delta-} : \ |x| + |y| + 1 = |z| \ \wedge \ |x| > 5 \ \wedge \ |y| > 0 \ \wedge \ 2 \nmid |x| \ \wedge \ 2 \nmid |y|$$

is realizable for Φ, *but it is not sound because it is not satisfied by the data assignment*

$$\sigma_\mathbb{T} : \{ x := \text{nil}, \ y := \text{nil}, \ z := \text{cons}(\text{nil}, \text{nil}) \} \ .$$

Finally, the constraint

$$\Phi_\Delta : \ |x| + |y| + 1 = |z| \ \wedge \ |x| > 0 \ \wedge \ |y| > 0 \ \wedge \ 2 \nmid |x| \ \wedge \ 2 \nmid |y|$$

is both sound and realizable, and hence is the induced length constraint of Φ_{list}.

We have a decision procedure for $\mathrm{Th}^\forall(\mathrm{TA}_\mathbb{Z})$ if Φ_Δ can be effectively computed from $\Phi_\mathbb{T}$.

Theorem 2.1 (Reduction with LCC [60]). *Let* Φ_Δ *be an LCC for* $\Phi_\mathbb{T}$. *Then* $\mathrm{TA}_\mathbb{Z} \models_\exists \Phi_\mathbb{T} \wedge \Phi_\mathbb{Z}$ *if and only if* $\mathrm{PA} \models_\exists \Phi_\Delta \wedge \Phi_\mathbb{Z}$.

Proof. Conditions (2.8) and (2.9) give the "⇒" and "⇐" directions, respectively. □

By this theorem the decision problem for quantifier-free theories reduces to computing the LCC in Presburger arithmetic. To obtain an LCC, we need a normalization process to transform $\Phi_\mathbb{T}$ to an equivalent disjunction in which each disjunct is of the form $\Phi'_\mathbb{T} \wedge \theta'_\mathbb{Z}$. We call such a transformation a *partitioning* and each disjunct a *partition*. (We do not require partitions to be mutually exclusive.)

In the subsequent sections, we shall show in detail each of the normalization procedures. First, we extend Definition 2.8 to deal with newly generated integer constraints in the normalization.

Definition 2.9 (Relativized LCC (RLCC) in $\mathrm{TA}_{\mathbb{Z}}$). *A formula $\Phi_\Delta(\bar{\mathbf{x}})$ is a* length constraint completion *for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})$ relativized to $\theta_{\mathbb{Z}}(\bar{\mathbf{x}})$, (in short, $\Phi_\Delta(\bar{\mathbf{x}})$ is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})/\theta_{\mathbb{Z}}(\bar{\mathbf{x}})$), if the following formulas are valid:*

$$(\forall \bar{\mathbf{x}} : \mathbb{T})\Big[\, \Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \wedge\ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}) \ \rightarrow\ (\exists \bar{\mathbf{z}} : \mathbb{Z})\Big(\Phi_\Delta(\bar{\mathbf{z}}) \ \wedge\ |\bar{\mathbf{x}}| = \bar{\mathbf{z}}\Big)\Big] \ , \tag{2.11}$$

$$(\forall \bar{\mathbf{z}} : \mathbb{Z})\Big[\, \Phi_\Delta(\bar{\mathbf{z}}) \ \rightarrow\ (\exists \bar{\mathbf{x}} : \mathbb{T})\Big(\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \wedge\ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}) \ \wedge\ |\bar{\mathbf{x}}| = \bar{\mathbf{z}}\Big)\Big] \ . \tag{2.12}$$

Example 2.10 (RLCC in List**).** *Consider* List*. Let*

$$\bar{\mathbf{x}} : \{x_1, x_2, x_3\} \ , \qquad \Phi_{\mathbb{T}}(\bar{\mathbf{x}}) : \mathrm{cons}(x_1, x_2) = x_3 \ , \qquad \theta_{\mathbb{Z}}(\bar{\mathbf{x}}) : |x_1| < |x_2| \ .$$

Consider the following formulas:

$$\begin{aligned}
\Phi_{div} &: \ |x_1| > 0 \ \wedge\ |x_2| > 0 \ \wedge\ 2 \nmid |x_1| \ \wedge\ 2 \nmid |x_2| \ , \\
\Phi_{\Delta} &: \ |x_1| + |x_2| + 1 = |x_3| \ \wedge\ |x_1| < |x_2| \ \wedge\ \Phi_{div} \ , \\
\Phi_{\Delta+} &: \ |x_1| < |x_3| \ \wedge\ |x_2| < |x_3| \ \wedge\ |x_1| < |x_2| \ \wedge\ \Phi_{div} \ , \\
\Phi_{\Delta-} &: \ |x_1| + |x_2| + 1 = |x_3| \ \wedge\ |x_1| \leq 3 \ \wedge\ |x_2| > 3 \ \wedge\ \Phi_{div} \ .
\end{aligned}$$

It is not hard to prove that Φ_Δ is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})/\theta_{\mathbb{Z}}(\bar{\mathbf{x}})$. But neither $\Phi_{\Delta+}$ nor $\Phi_{\Delta-}$ is such an RLCC. Although $\Phi_{\Delta+}$ satisfies (2.11)*, it does not satisfy* (2.12)*, as the assignment*

$$\{\, |x_1| := 2,\ |x_2| := 3,\ |x_3| = 4 \,\}$$

can not be realized by any assignment for $\bar{\mathbf{x}}$. On the other hand, $\Phi_{\Delta-}$ satisfies (2.12)*, but not* (2.11)*, as the assignment*

$$\{\, x_1 := \mathrm{nil},\ x_2 := \mathrm{cons}(\mathrm{nil}, \mathrm{nil}),\ x_3 := \mathrm{cons}(\mathrm{nil}, \mathrm{cons}(\mathrm{nil}, \mathrm{nil})) \,\}$$

falsifies $\Phi_{\Delta-}$.

Comparing (2.8) and (2.9) with (2.11) and (2.12), we see that an LCC is an RLCC with $\theta_{\mathbb{Z}} \equiv \mathrm{true}$. Like LCCs, up to equivalence, there exists a unique RLCC with respect to $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})/\theta_{\mathbb{Z}}(\bar{\mathbf{x}})$. In addition, RLCCs have the following easily proved "additive" property.

Proposition 2.1 (Additivity of RLCC). *If Φ_Δ is an RLCC for $\Phi_{\mathbb{T}}/\theta_{\mathbb{Z}}$, then for any $\theta'_{\mathbb{Z}}$, $\Phi_\Delta \wedge \theta'_{\mathbb{Z}}$ is an RLCC for $\Phi_{\mathbb{T}}/(\theta_{\mathbb{Z}} \wedge \theta'_{\mathbb{Z}})$.*

In particular, by letting $\theta'_{\mathbb{Z}} := \Phi_{\mathbb{Z}}$ and $\theta_{\mathbb{Z}} :=$ true, we see that if Φ_Δ is an LCC for $\Phi_{\mathbb{T}}$, then $\Phi_\Delta \wedge \Phi_{\mathbb{Z}}$ is an RLCC for $\Phi_{\mathbb{T}}/\Phi_{\mathbb{Z}}$. So Theorem 2.1 is in fact a special case of the following theorem.

Theorem 2.2 (Reduction with RLCC). *Let Φ_Δ be an RLCC for $\Phi_{\mathbb{T}}/\theta_{\mathbb{Z}}$. Then* $\mathrm{TA}_{\mathbb{Z}} \models_\exists \Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ *if and only if* $\mathrm{PA} \models_\exists \Phi_\Delta$.

Proof. Conditions (2.11) and (2.12) give the "⇒" and "⇐" directions, respectively. □

This theorem motivates the strategy of our decision procedures. In the normalization process, with introduction of auxiliary integer constraints, we partition the original search space for $\Phi_{\mathbb{T}}$ such that $\Phi_{\mathbb{T}} \leftrightarrow \bigcup_i \Phi^{(i)}_{\mathbb{T}} \wedge \theta^{(i)}_{\mathbb{Z}}$, until we easily compute the RLCC $\Phi^{(i)}_\Delta$ for each $\Phi^{(i)}_{\mathbb{T}}/\theta^{(i)}_{\mathbb{Z}}$. By Proposition 2.1, $\Phi^{(i)}_\Delta \wedge \Phi_{\mathbb{Z}}$ is an RLCC for $\Phi^{(i)}_{\mathbb{T}}/(\theta^{(i)}_{\mathbb{Z}} \wedge \Phi_{\mathbb{Z}})$. Then $\mathrm{TA}_{\mathbb{Z}} \models_\exists \Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$ if and only if for some i, $\mathrm{TA}_{\mathbb{Z}} \models_\exists \Phi^{(i)}_{\mathbb{T}} \wedge \theta^{(i)}_{\mathbb{Z}} \wedge \Phi_{\mathbb{Z}}$, which, by Theorem 2.2 (set $\Phi_{\mathbb{T}} := \Phi^{(i)}_{\mathbb{T}}$, $\Phi_\Delta := \Phi^{(i)}_\Delta \wedge \Phi_{\mathbb{Z}}$, $\Phi_{\mathbb{Z}} := \Phi_{\mathbb{Z}} \wedge \theta^{(i)}_{\mathbb{Z}}$), reduces to determining whether $\mathrm{PA} \models_\exists \Phi^{(i)}_\Delta \wedge \Phi_{\mathbb{Z}}$. Note that $\Phi_{\mathbb{Z}}$ is not involved in computing an RLCC. Therefore, we can assume that $\theta^{(i)}_{\mathbb{Z}}$ includes constraints relevant to the corresponding partitioning and other constraints generated during the normalization procedure have been merged into $\Phi_{\mathbb{Z}}$.

Algorithm 2.3 (Generic Decision Procedure). *Input:* $\Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$.

1. *Return* FAIL *if* $\mathrm{TA} \not\models_\exists \Phi_{\mathbb{T}}$.
2. *For each partition* $\Phi^{(i)}_{\mathbb{T}} \wedge \theta^{(i)}_{\mathbb{Z}}$ *of* $\Phi_{\mathbb{T}}$*:*
   (a) *Compute an RLCC* $\Phi^{(i)}_\Delta$ *for* $\Phi^{(i)}_{\mathbb{T}}/\theta^{(i)}_{\mathbb{Z}}$.
   (b) *Return* SUCCESS *if* $\mathrm{PA} \models_\exists \Phi^{(i)}_\Delta \wedge \Phi_{\mathbb{Z}}$.
3. *Return* FAIL.

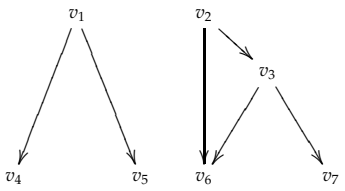
2.4.3 A Decision Procedure for $\mathrm{Th}^\forall(\mathrm{TA}^\omega_{\mathbb{Z}})$

The easiest arithmetic extension of term algebras is $\mathrm{Th}^\forall(\mathrm{TA}^\omega_{\mathbb{Z}})$, the quantifier-free theory of term algebras with integers and with an infinite constant domain. In $\mathrm{TA}^\omega_{\mathbb{Z}}$, an LCC can be derived directly from the DAG for the formula. (We do not need the notion of RLCC in this case.) Before we present the algorithm we define the following integer predicates on lengths of terms in a DAG:

$$\mathrm{Tree}(x) \quad : \quad \exists x_1, \ldots, x_n \geq 0 \left(x = 1 + \sum_{i=1}^{n} d_i x_i \right) , \tag{2.13}$$

$$\mathrm{Node}^\alpha(x, \bar{\mathbf{x}}_\alpha) \quad : \quad x = 1 + \sum_{i=1}^{\mathrm{ar}(\alpha)} x_i \ , \tag{2.14}$$

$$\mathrm{Tree}^\alpha(x) \quad : \quad \exists \bar{x}_\alpha \Big(\mathrm{Node}^\alpha(x, \bar{\mathbf{x}}_\alpha) \wedge \bigwedge_{0<i\leq \mathrm{ar}(\alpha)} \mathrm{Tree}(x_i) \Big) \ , \tag{2.15}$$



$v_1 : x$ $\quad$ $v_5 : \mathrm{cdr}(x)$
$v_2 : \mathrm{cons}(\mathrm{car}(y), y)$ $\quad$ $v_6 : \mathrm{car}(y)$
$v_3 : y$ $\quad$ $v_7 : \mathrm{cdr}(y)$
$v_4 : \mathrm{car}(x)$

Figure 2-3: The DAG of $x = \mathrm{cons}(\mathrm{car}(y), y) \wedge |\mathrm{cons}(\mathrm{car}(y), y)| < 2|\mathrm{car}(x)| \wedge \mathrm{Is}_{\mathrm{cons}}(y)$.

where $\bar{\mathbf{x}}_\alpha$ denotes $x_1, \ldots, x_{\mathrm{ar}(\alpha)}$ and $d_1, \ldots, d_n$ are the distinct arities of non-nullary constructors. The predicate $\mathrm{Tree}(x)$ is true if and only if x is the length of a well-formed tree, since whenever a leaf expands one level with outgoing degree d, the length of the tree increases by d. The second predicate expresses that the length of an α-typed node with known children is the sum of its children's lengths plus 1. The last predicate states the length constraint for an α-typed tree. With these predicates the construction of an LCC is given by the following algorithm.

Algorithm 2.4 (Computation of LCC in $\mathrm{TA}^\omega_\mathbb{Z}$). *Let $\Phi_\mathbb{T}$ be a type-complete term constraint, $G_\mathbb{T}$ the DAG of $\Phi_\mathbb{T}$ and $R\downarrow\uparrow$ the bidirectional closure obtained by Algorithm 2.1. Initially set $\Phi_\Delta = \emptyset$. For each term t add the following to Φ_Δ.*

- $|t| = 1$, *if t is a constant or asserted to be a constant (i. e., $\mathrm{Is}_\mathsf{A}(t)$ is in $\Phi_\mathbb{T}$);*
- $|t| = |s|$, *if $(t, s) \in R\downarrow\uparrow$;*
- $\mathrm{Node}^\alpha(|t|, |t_1|, \ldots, |t_{\mathrm{ar}(\alpha)}|)$, *if t is an α-typed vertex with children $t_1, \ldots, t_{\mathrm{ar}(\alpha)}$;*
- $\mathrm{Tree}^\alpha(|t|)$, *if t is an α-typed leaf vertex.*

Proposition 2.2 (LCC in $\mathrm{TA}^\omega_\mathbb{Z}$). *$\Phi_\Delta$ obtained by Algorithm 2.4 is expressible in a quantifier-free Presburger formula linear in the size of $\Phi_\mathbb{T}$.*

Theorem 2.3 (LCC in $\mathrm{TA}^\omega_\mathbb{Z}$). *$\Phi_\Delta$ obtained by Algorithm 2.4 is an LCC for $\Phi_\mathbb{T}$.*

Algorithm 2.5 (Decision Procedure for $\mathrm{Th}^\forall(\mathrm{TA}^\omega_\mathbb{Z})$). *Input: $\Phi_\mathbb{T} \wedge \Phi_\mathbb{Z}$ where $\Phi_\mathbb{T}$ is type-complete.*

1. *Call Algorithm 2.1 on $\Phi_\mathbb{T}$; return* FAIL *if* $\mathrm{TA} \not\models_\exists \Phi_\mathbb{T}$.
2. *Construct Φ_Δ from the DAG $G_\mathbb{T}$ using Algorithm 2.4.*
   - *Return* SUCCESS *if* $\mathrm{PA} \models_\exists \Phi_\Delta \wedge \Phi_\mathbb{Z}$.
   - *Return* FAIL *otherwise.*

The correctness of the algorithm follows from Theorem 2.1 and Theorem 2.3.

Example 2.11 (LCC in $\mathrm{TA}^{\omega}_{\mathbb{Z}}$). *We here assume that* List *contains infinitely many constants. Figure 2-3 shows the DAG of*

$$x = \mathrm{cons}(\mathrm{car}(y), y) \ \wedge \ |\mathrm{cons}(\mathrm{car}(y), y)| < 2|\mathrm{car}(x)| \ \wedge \ \mathrm{Is}_{\mathrm{cons}}(y) \ , \tag{2.16}$$

where y is asserted not to be a constant. The computed $R\downarrow^{\uparrow}$ is $\{(v_1, v_2), (v_3, v_5), (v_4, v_6)\}$. By Algorithm 2.4 Φ_{Δ} is

$$\begin{array}{c} |x| = |\mathrm{cons}(\mathrm{car}(y), y)| \ \wedge \ |\mathrm{car}(x)| = |\mathrm{car}(y)| \ \wedge \ |\mathrm{cdr}(x)| = |y| \\ \wedge \\ |x| = |\mathrm{car}(x)| + |\mathrm{cdr}(x)| + 1 \ \wedge \ |y| = |\mathrm{car}(y)| + |\mathrm{cdr}(y)| + 1 \\ \wedge \\ |\mathrm{cons}(\mathrm{car}(y), y)| = |\mathrm{car}(y)| + |y| + 1 \\ \wedge \\ 2 \nmid |\mathrm{car}(x)| \ \wedge \ 2 \nmid |\mathrm{cdr}(x)| \ \wedge \ 2 \nmid |\mathrm{car}(y)| \ \wedge \ 2 \nmid |\mathrm{cdr}(y)| \\ \wedge \\ |\mathrm{car}(x)| \geq 1 \ \wedge \ |\mathrm{cdr}(x)| \geq 1 \ \wedge \ |\mathrm{car}(y)| \geq 1 \ \wedge \ |\mathrm{cdr}(y)| \geq 1 \end{array} \tag{2.17}$$

which implies

$$|\mathrm{cons}(\mathrm{car}(y), y)| \geq 2|\mathrm{car}(x)| \ , \tag{2.18}$$

contradicting $|\mathrm{cons}(\mathrm{car}(y), y)| < 2|\mathrm{car}(x)|$. *If y is asserted to be a constant, then v_3, v_6, v_7 are merged. In this case Φ_{Δ} also implies* (2.18)*, and therefore* (2.16) *is unsatisfiable.*

Note that the last two lines of (2.17) *are the result of simplification of constraints of the form* $\mathrm{Tree}(\cdot)$*; according to our definition of the length function, the length of any term (tree) in* List *is a positive odd number.*

2.4.4 A Decision Procedure for $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$

Algorithm 2.4 can produce an incorrect LCC in $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$, the quantifier-free theory of term algebras with integers and with a finite constant domain, as illustrated by the following example.

Example 2.12 (Hidden Length Constraints in $\mathrm{TA}_{\mathbb{Z}}$). *Consider* List *with $\mathcal{A} = \{\mathrm{nil}\}$. The constraint*

$$|x| = 5 \ \wedge \ \mathrm{Is}_{\mathsf{A}}(y) \ \wedge \ x \neq \mathrm{cons}(\mathrm{cons}(y, y), y) \ \wedge \ x \neq \mathrm{cons}(y, \mathrm{cons}(y, y)) \tag{2.19}$$

is unsatisfiable while Φ_{Δ} obtained by Algorithm 2.4 is

$$|y| = 1 \ \wedge \ |\mathrm{cons}(y, y)| = 3 \ \wedge \ |\mathrm{cons}(\mathrm{cons}(y, y), y)| = 5 \ \wedge \ |\mathrm{cons}(y, \mathrm{cons}(y, y)| = 5$$

which is obviously satisfiable together with $|x| = 5$.

The reason is that if $\mathcal{A}$ is finite, then there are only finitely many terms of length n for any $n > 0$. If a term t is forced to be distinct from all of them, then t cannot have length n. Therefore Φ_Δ needs to include constraints that count the number of distinct terms of a certain length.

Definition 2.10 (Counting Constraints). *A* counting constraint *is a predicate* $\mathrm{CNT}^{\alpha}_{k,n}(x)$ *(*$k > 0, n \geq 0$*) that is* true *if and only if there are at least* $n+1$ *different* α*-terms of length* x *in* TA *with* $|\mathcal{A}| = k$. $\mathrm{CNT}_{k,n}(x)$ *is similarly defined with* α*-terms replaced by* TA*-terms.*

Example 2.13 (Counting Constraints). *For* List *with* $\mathcal{A} = \{\mathrm{nil}\}$, $\mathrm{CNT}^{\mathrm{cons}}_{1,n}(x)$ *is* $x \geq 2m-1 \wedge 2 \nmid m$ *where* m *is the least number such that the* m*-th* Catalan number

$$C_m = \frac{1}{m}\binom{2m-2}{m-1}$$

is greater than n*. This is not surprising as* C_m *gives the number of binary trees with* m *leaves (that tree has* $2m-1$ *nodes).*

The following two *monotonicity* properties are easily proven: for any $l \geq k > 0$ and $m \geq n > 0$,

$$\mathrm{CNT}^{\alpha}_{k,n}(x) \to \mathrm{CNT}^{\alpha}_{l,n}(x) \ , \qquad \mathrm{CNT}^{\alpha}_{k,m}(x) \to \mathrm{CNT}^{\alpha}_{k,n}(x) \ .$$

In general we have the following result.

Theorem 2.4 (Counting Constraints in $\mathrm{TA}_{\mathbb{Z}}$**).** $\mathrm{CNT}^{\alpha}_{k,n}(x)$ *and* $\mathrm{CNT}_{k,n}(x)$ *are expressible by quantifier-free Presburger formulas that can be computed in time* $O(n)$.

In order to construct counting constraints, we need equality information between terms.

Definition 2.11 (Equality Completeness). $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ *is* equality complete *if for any two terms* u *and* v *in* $\Phi_{\mathbb{T}}$

- *either* $u = v$ *or* $u \neq v$ *(but not both) is in* $\Phi_{\mathbb{T}}$*, and*
- *either* $|u| = |v|$ *or* $|u| \neq |v|$ *(but not both) is in* $\theta_{\mathbb{Z}}$.

Equality completeness is a syntactical notion similar to a variable partition in the Nelson-Oppen combination method. We can make a quantifier-free formula $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ (which does not contain contradictory literals) equality complete by adding exactly one of $u = v$ and $u \neq v$ to $\Phi_{\mathbb{T}}$, and exactly one of $|u| = |v|$ and $|u| \neq |v|$ to $\theta_{\mathbb{Z}}$. Let us call the resulting formula an *equality completion* of $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$. Similarly, we can define *equality completion* for sets of terms. Like type completion, $\Phi'_{\mathbb{T}} \wedge \theta'_{\mathbb{Z}}$ is a *compatible* equality completion of $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ if the satisfiability of $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ implies the satisfiability of $\Phi'_{\mathbb{T}} \wedge \theta'_{\mathbb{Z}}$.

Example 2.14 (Equality Completion). *Let* Φ_{list} *be* $y \neq \text{cons}(x,z) \wedge \text{Is}_{\text{cons}}(x,y,z)$. *A possible equality completion of* Φ_{list} *(*$\theta_{\mathbb{Z}} = \emptyset$*) is*

$$|y| = |\text{cons}(x,z)| \ \wedge\ |x| = |z| \ \wedge\ |y| \neq |x| \ \wedge\ \text{Is}_{\text{cons}}(x,y,z) \ \wedge \bigwedge_{t,t' \in \mathcal{S}; t \neq t'} t \neq t' \ , \tag{2.20}$$

where $\mathcal{S} = \{x, y, z, \text{cons}(x,z)\}$.

Strictly speaking, the formula (2.20) is not an equality completion of $\Phi_{\mathbb{T}}$; to save space, we omitted equalities and disequalities that follow from equality substitution. In general, equality completion could add $O(n^2)$ literals. It however can be more succinctly represented as assertions of the form $eq(t_1, \ldots, t_n)$ or $neq(t_1, \ldots, t_n)$ that state that a set of terms are all equal or all pairwise distinct, respectively.

We partition the search space for $\Phi_{\mathbb{T}}$ by computation of equality completion. To save notation, $\Phi_{\mathbb{T}}$ and $\theta_{\mathbb{Z}}$ always refer to the updated version for one of the partitions. By $\text{CLS}^{\alpha}_{n}(x_0, x_1, \ldots, x_n)$ we denote the conjunction of literals expressing that $x_0, \ldots, x_n$ are α-typed terms having the same length but pairwise distinct.

Algorithm 2.6 (Computation of RLCC in $\text{TA}_{\mathbb{Z}}$**).** *Input:* $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ *(type and equality complete); k, the cardinality of* $\mathcal{A}$.

1. *Call Algorithm 2.4 to obtain* Φ_{Δ}.
2. *Set* $\Phi_{\Delta} := \Phi_{\Delta} \wedge \theta_{\mathbb{Z}}$.
3. *Add to* Φ_{Δ} *the constraints* $\text{CNT}^{\alpha}_{k,n}(|t|)$ *if* $\text{CLS}^{\alpha}_{n}(t, t_1, \ldots, t_n)$ *is induced by* $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ *for some* $t_1, \ldots, t_n$.

Note that $\text{CNT}^{\alpha}_{k,n}(|t|) \rightarrow \text{CNT}^{\alpha}_{k,m}(|t|)$ for $n \geq m$. In step (3), therefore, it suffices to add $\text{CNT}^{\alpha}_{k,n}(|t|)$ only if the set $\{t, t_1, \ldots, t_n\}$ is maximal in the sense that $\text{CLS}^{\alpha}_{n}(t, t_1, \ldots, t_n)$ occurs in $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$, but for no $t'_1, \ldots, t'_{n+1}$ does $\text{CLS}^{\alpha}_{n+1}(t, t'_1, \ldots, t'_{n+1})$ also occur in $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$. In addition, due to symmetry, there is no need to add $\text{CNT}^{\alpha}_{k,m}(|t_1|)$, …, $\text{CNT}^{\alpha}_{k,m}(|t_n|)$.

Proposition 2.3 (RLCC in $\text{TA}_{\mathbb{Z}}$**).** Φ_{Δ} *obtained by Algorithm 2.6 is expressible in a quantifier-free Presburger formula of size linear in the size of* $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$.

Theorem 2.5 (RLCC in $\text{TA}_{\mathbb{Z}}$**).** Φ_{Δ} *obtained by Algorithm 2.6 is an RLCC for* $\Phi_{\mathbb{T}}/\theta_{\mathbb{Z}}$.

Algorithm 2.7 (Decision Procedure for $\text{Th}^{\forall}(\text{TA}_{\mathbb{Z}})$**).** *Input :* $\Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$.

1. *Guess a type and equality completion of* $\Phi_{\mathbb{T}}$*, denoted by* $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$.
2. *Call Algorithm 2.1 on* $\Phi_{\mathbb{T}}$. *Return* FAIL *if* $\text{TA} \not\models_{\exists} \Phi_{\mathbb{T}}$.
3. *Construct* Φ_{Δ} *from* $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ *using Algorithm 2.6.*

- *Return* SUCCESS *if* PA $\models_\exists \Phi_\Delta \wedge \Phi_\mathbb{Z}$.
- *Return* FAIL *otherwise.*

The correctness of Algorithm 2.7 follows from Theorems 2.2 and 2.5. Notice that, when $\Phi_\mathbb{Z}$ is empty, the algorithm can be viewed as an extension of Oppen's original algorithm for structures with a finite constant domain.

Example 2.15 (RLCC in $\mathrm{TA}_\mathbb{Z}$). *Let us return to Example 2.12. Constraint* (2.19) *has exactly one compatible completion, namely*

$$\mathrm{CLS}_2^{\mathrm{cons}}(x, \mathrm{cons}(\mathrm{cons}(y, y), y), \mathrm{cons}(y, \mathrm{cons}(y, y))) \ .$$

This results in an RLCC containing the counting constraint

$$\mathrm{CNT}_{1,2}^{\mathrm{cons}}(|x|) \ : \ |x| > 5 \ \wedge \ 2 \nmid |x| \ ,$$

contradicting $|x| = 5$ *in* (2.19).

2.4.5 Richer Theories on Constant Domain

Up to now we assumed that the constant domain is purely equational, i. e., we can only express equality and disequality between constants. It is fairly easy, however, to relax this assumption and allow a constant domain with richer constructs provided the enriched structure is decidable and the signature of this structure is disjoint with the signature of $\mathrm{TA}_\mathbb{Z}$ except for constants. (In fact, disjointness is trivially satisfied because in $\mathscr{L}_\mathbb{T}^\mathbb{Z}$ we only have the equality predicate on constants.) We outline the approach below.

Let $\mathfrak{A}_c$ denote the new constant structure and $\mathscr{L}_c$ the corresponding language. Let $\mathrm{TA}_\mathbb{Z}^+$ denote the extended structure of term algebras with integers and Φ^+ a constraint in $\mathrm{TA}_\mathbb{Z}^+$. Without loss of generality, we assume that both Φ^+ is equality and type complete with respect to TA-terms. There is a standard way to purify Φ^+ to $\Phi \wedge \Phi_c \wedge \mathrm{Abs}$ where Φ is a constraint in $\mathscr{L}_{\mathrm{TA}}^\mathbb{Z}$, Φ_c is a constraint in $\mathscr{L}_c$, and Abs is the set of equalities of the form $\{v_i = t_i\}_i$ where v_i are fresh variables and t_i are TA-terms in Φ which are of constant type but are not constants directly. Both Φ and Φ_c contains the same equality completion (up to the isomorphic mapping $\{v_i \to t_i\}_i$) for terms of constant type. It follows from the Nelson-Oppen combination method that $\mathrm{TA}_\mathbb{Z}^+ \models_\exists \Phi^+$ if and only if $\mathrm{TA}_\mathbb{Z} \models_\exists \Phi$ and $\mathfrak{A}_c \models_\exists \Phi_c$.

2.4.6 Complexity

The complexity of the decision problems for the quantifier-free theories is NP-complete. Let n be the input size. First it is not hard to see that decision problems for both $\mathrm{Th}^\forall(\mathrm{TA}_\mathbb{Z})$ and $\mathrm{Th}^\forall(\mathrm{TA}_\mathbb{Z}^\omega)$

are NP-hard as they are super-theories of $\mathrm{Th}^{\forall}(\mathrm{TA})$ and $\mathrm{Th}^{\forall}(\mathrm{PA})$, both of which are NP-complete [45], [20, pp. 336-340]. Second, Algorithm 2.4 computes Φ_Δ in $O(n)$ (see Proposition 2.2) and so does Algorithm 2.6. Third, the size of any type and equality completion of Φ is bounded by $O(n^2)$ as there are at most n^2 pairs of terms. By the nondeterministic nature of our algorithms, we see that each branch of computation (in Algorithms 2.5 and 2.7 respectively) is in P. Therefore both $\mathrm{Th}^{\forall}(\mathrm{TA}^{\omega}_{\mathbb{Z}})$ and $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ are NP-complete.

2.5 Case Study: Verification of Red-Black Tree Algorithm

In this section we present formalisms for analyzing the red-black tree algorithm. We also extend our method to a theory of a term algebra with two length functions. Let us first consider a simple example of tree manipulation.

Algorithm 2.8 (Left-Rotation of Trees).
Input: List x. **Require:** $\mathrm{cdr}(x) \neq \mathrm{nil}$; **Insure:** $|y| = |x|$.

1: $t := x$
2: $y := \mathrm{cdr}(t)$
3: $\mathrm{cdr}(t) := \mathrm{car}(y)$
4: $\mathrm{car}(y) := t$
5: ***return*** y

Algorithm 2.8 is a tree balancing routine used in the red-black tree algorithm. It takes an input tree x and "shuffles" the root node of x with the root node of the right child of x. Our decision procedure can prove that the size of the tree returned by Algorithm 2.8 is equal to the size of x.

We carry out the post-condition computation as follows. Let $\bar{\mathbf{v}}$ denote the variables in the current state and $\bar{\mathbf{v}}'$ denote the corresponding variables in the next state. The transition relation of a statement q is denoted by $\rho_q(\bar{\mathbf{v}}, \bar{\mathbf{v}}')$. The post-condition $post(q, \varphi)$ of $\varphi(\bar{\mathbf{v}})$ after executing a statement q is

$$(\exists \bar{\mathbf{v}}^0)\Big(\ \rho_q(\bar{\mathbf{v}}^0, \bar{\mathbf{v}})\ \wedge\ \varphi(\bar{\mathbf{v}}^0)\ \Big)\ .$$

For example, the state φ_0 before statement 1 is $\mathrm{cdr}(x) \neq \mathrm{nil}$. The post-condition after executing statement 1 is

$$(\exists x\ \exists t\ \exists y)\Big[\ \mathrm{cdr}(x) \neq \mathrm{nil}\ \wedge\ t' = x\ \wedge\ x' = x\ \wedge\ y' = y\ \Big]\ ,$$

which simplifies to $\mathrm{cdr}(x') \neq \mathrm{nil} \wedge t' = x'$, and after renaming, to $\mathrm{cdr}(x) \neq \mathrm{nil} \wedge t = x$, called φ_1.

The transition relation of two sequential statements can be computed as follows. Let $\rho_q(\bar{\mathbf{v}}, \bar{\mathbf{v}}^1)$ and $\rho_r(\bar{\mathbf{v}}^1, \bar{\mathbf{v}}')$ be the transition relations for statements q and r respectively. Then the transition

relation of the composite statement $\langle q; r\rangle$ is

$$(\exists \bar{\mathbf{v}}^1)\Big(\ \rho_q(\bar{\mathbf{v}}, \bar{\mathbf{v}}^1)\ \wedge\ \rho_r(\bar{\mathbf{v}}^1, \bar{\mathbf{v}}')\ \Big)\ .$$

The validity checking of a Hoare triples $\{\varphi\}q\{\psi\}$ is equivalent to proving that $post(q, \varphi) \rightarrow \psi$.

Below we list the result of each step of the computation.

$$\begin{array}{lcl}
\varphi_0 & : & \mathrm{cdr}(x) \neq \mathrm{nil}\ , \\
\varphi_1 & : & \mathrm{cdr}(x) \neq \mathrm{nil}\ \wedge\ t = x\ , \\
\varphi_2 & : & \mathrm{cdr}(x) \neq \mathrm{nil}\ \wedge\ t = x\ \wedge\ y = \mathrm{cdr}(t)\ , \\
\varphi_3 & : & \mathrm{cdr}(x) \neq \mathrm{nil}\ \wedge\ y = \mathrm{cdr}(x)\ \wedge\ \mathrm{cdr}(t) = \mathrm{car}(y)\ \wedge\ \mathrm{car}(t) = \mathrm{car}(x)\ , \\
\varphi_4 & : & \mathrm{cdr}(x) \neq \mathrm{nil}\ \wedge\ \mathrm{cdr}(t) = \mathrm{car}(\mathrm{cdr}(x))\ \wedge\ \mathrm{car}(t) = \mathrm{car}(x)\ \wedge\ \mathrm{car}(y) = t\ \wedge\ \mathrm{cdr}(y) = \mathrm{cdr}(\mathrm{cdr}(x))\ ,
\end{array}$$

where φ_4 describes the state after executing statement 4. It is easily seen that $\varphi_4 \rightarrow |x| = |y|$ is valid.

2.5.1 The Theory of Red-Black Trees

In the above example, the integer constraints come from the simple length function. Our method for deriving decision procedures, however, can apply to theories with more than one integer function on terms. In this section we present a theory of a term algebra with two integer functions to express the properties of red-black trees.

Definition 2.12 (Red-black Trees [13]). *A* red-black *tree is a binary tree with the following coloring properties:*

1. *Every node is either red or black.*
2. *Every leaf node is black.*
3. *The root is black.*
4. *Every red node has two black children.*
5. *All paths from the root to leaf nodes contain the same number of black nodes.*

Properties (1)-(3) can be modeled in a theory of term algebras as follows:

Definition 2.13 (Colored Trees). *The structure of red-black colored trees is*

$$\mathrm{RB} = \langle\ \mathbb{T}_{\mathrm{rb}};\ \{\mathrm{red}, \mathrm{black}, \mathrm{nil}\},\ \{\mathrm{nil}\},\ \{\mathrm{Is}_{\mathrm{red}}, \mathrm{Is}_{\mathrm{black}}, \mathrm{Is}_{\mathrm{nil}}, \mathrm{Is}_{\mathrm{A}}\}\ \rangle\ ,$$

where $\mathbb{T}_{\mathrm{rb}}$ *denotes the domain,* nil *denotes a leaf,* red *and* black *are binary constructors. The corresponding language is denoted by* $\mathscr{L}_{\mathrm{RB}}$.

We opt not to include selectors in the language for simplicity. As before we use *terms* and *trees*, respectively, to refer to syntactic objects and semantic objects. We call terms (trees) of red-type (resp. of black-type) red-*terms (-trees)* (resp. black-*terms (-trees)*).

We extend RB with PA to express balancing properties (4)-(5):

Definition 2.14 (Structure of Red-black Trees). *The structure of red-black trees is*

$$\mathrm{RB}_{\mathbb{Z}} = \langle\, \mathrm{RB};\, \mathrm{PA};\, |\cdot|_{\max}, |\cdot|_{\min} : \mathbb{T}_{\mathrm{rb}} \to \mathbb{N} \,\rangle \ ,$$

where, $|\cdot|_{\max}$ *and* $|\cdot|_{\min}$ *are two integer functions defined recursively as*

$$|x|_{\star} = \begin{cases} 1 & x = \mathrm{nil} \ , \\ 0 & \mathrm{Vio}(x) \ , \\ \star(|x_1|_{\star}, |x_2|_{\star}) + 1 & \mathrm{GB}(x, x_1, x_2) \ , \\ \star(|x_1|_{\star}, |x_2|_{\star}) & \mathrm{GR}(x, x_1, x_2) \ . \end{cases}$$

where $\star \in \{\max, \min\}$ *and* $\mathrm{GB}(x, x_1, x_2)$, $\mathrm{GR}(x, x_1, x_2)$ *and* $\mathrm{Vio}(x)$ *are*

$$\begin{aligned} \mathrm{Vio}(x) &\overset{\mathrm{def}}{=\!=} x \neq \mathrm{nil} \ \wedge\ \forall x_1 \forall x_2 \left(\neg \mathrm{GB}(x, x_1, x_2) \ \vee\ \neg \mathrm{GR}(x, x_1, x_2) \right) \ , \\ \mathrm{GB}(x, x_1, x_2) &\overset{\mathrm{def}}{=\!=} x = \mathrm{black}(x_1, x_2) \ \wedge\ |x_1|_{\max} \neq 0 \ \wedge\ |x_2|_{\max} \neq 0 \ , \\ \mathrm{GR}(x, x_1, x_2) &\overset{\mathrm{def}}{=\!=} x = \mathrm{red}(x_1, x_2) \ \wedge\ |x_1|_{\max} \neq 0 \ \wedge\ |x_2|_{\max} \neq 0 \\ &\qquad \wedge\ \neg \mathrm{Is}_{\mathrm{red}}(x_1) \ \wedge\ \neg \mathrm{Is}_{\mathrm{red}}(x_2) \ . \end{aligned}$$

We denote the corresponding language by $\mathscr{L}^{\mathbb{Z}}_{\mathrm{RB}}$.

$\mathrm{Vio}(x)$ states that x violates property (4) of red-black trees. $\mathrm{GB}(x, x_1, x_2)$ states x is a black tree with two good subtrees x_1 and x_2. Similarly for $\mathrm{GR}(x, x_1, x_2)$. $|x|_{\max}$ (resp. $|x|_{\min}$) gives the maximal (resp. minimal) number of black nodes that x can have on a maximal path. A maximal path of x that contains the largest (resp. smallest) number of black nodes is called a *maximal black path* (resp. *minimal black path*) of x. We call $|x|_{\max}$ the *maximal black length* of x, $|x|_{\min}$ the *minimal black length*, and the pair $(|x|_{\max}, |x|_{\min})$ the *measure* of x, denoted by $\|x\|$.

In this theory, properties (1) and (2) of Definition 2.12, which state that every node is either black or red, and that a nil node is black, are trivially satisfied by the choice of signature and the integer

functions. Therefore x is a red-black tree if x satisfies the following three conditions.

(§1)	$\|x\|_{\max} = \|x\|_{\min}$	*any maximal path of x contains the same number of black nodes,*
(§2)	$\|x\|_{\max} > 0$	*any red node of x must have two black children,*
(§3)	$\mathrm{Is}_{\mathrm{black}}(x)$	*the root of x is black.*

We denote by $\varphi^{-}_{\mathrm{RB}}(x)$ the conjunction of (§1) and (§2), and by $\varphi_{\mathrm{RB}}(x)$ the conjunction of (§1)-(§3). We note that $\varphi_{\mathrm{RB}}(x)$ defines a subdomain of $\mathbb{T}_{rb}$ and the theory of this subdomain can be obtained by relativizing quantifiers to $\varphi_{\mathrm{RB}}(x)$. Formally, $\forall x(\varphi_{\mathrm{RB}}(x) \to \Phi(x))$ (resp. $\exists x(\varphi_{\mathrm{RB}}(x) \wedge \Phi(x))$) expresses that Φ is a universal (resp. existential) property of red-black trees.

We define three counting constraints $\mathrm{CNT}^{\mathrm{rb}}_n(x_1, x_2)$, $\mathrm{CNT}^{r}_n(x_1, x_2)$ and $\mathrm{CNT}^{b}_n(x_1, x_2)$ as follows.

$$
\begin{array}{lll}
\mathrm{CNT}^{\mathrm{rb}}_n(x_1, x_2) & \text{iff} & |\{\, t \in \mathbb{T}_{\mathrm{rb}} \mid |t|_{\max} = x_1 \wedge |t|_{\min} = x_2 \,\}| > n \ , \\
\mathrm{CNT}^{r}_n(x_1, x_2) & \text{iff} & |\{\, t \in \mathbb{T}_{\mathrm{rb}} \mid |t|_{\max} = x_1 \wedge |t|_{\min} = x_2 \wedge \mathrm{Is}_{\mathrm{red}}(t) \,\}| > n \ , \\
\mathrm{CNT}^{b}_n(x_1, x_2) & \text{iff} & |\{\, t \in \mathbb{T}_{\mathrm{rb}} \mid |t|_{\max} = x_1 \wedge |t|_{\min} = x_2 \wedge \mathrm{Is}_{\mathrm{black}}(t) \,\}| > n \ .
\end{array}
$$

$\mathrm{CNT}_n(x_1, x_2)$ states that there are more than n distinct terms of measure (x_1, x_2). Similarly for $\mathrm{CNT}^{r}_n(x_1, x_2)$ and $\mathrm{CNT}^{b}_n(x_1, x_2)$.

Theorem 2.6 (Counting Constraints in $\mathrm{RB}_{\mathbb{Z}}$). *$\mathrm{CNT}_n(x_1, x_2)$, $\mathrm{CNT}^{r}_n(x_1, x_2)$ and $\mathrm{CNT}^{b}_n(x_1, x_2)$ are expressible by quantifier-free Presburger formulas that can be computed in $O(n)$.*

Similar to Definition 2.9 we have the notion of RLCC.

Definition 2.15 (Relativized LCC (RLCC) in $\mathrm{RB}_{\mathbb{Z}}$). *An $\mathscr{L}^{\mathbb{Z}}_{\mathrm{RB}}$-formula $\Phi_{\Delta}(\bar{\mathbf{x}})$ is an LCC for $\Phi_{\mathrm{RB}}(\bar{\mathbf{x}})$ relativized to $\theta_{\mathbb{Z}}(\bar{\mathbf{x}})$, (in short, $\Phi_{\Delta}(\bar{\mathbf{x}})$ is an RLCC for $\Phi_{\mathrm{RB}}(\bar{\mathbf{x}})/\theta_{\mathbb{Z}}(\bar{\mathbf{x}})$), if the following formulas are valid:*

$$
(\forall \bar{\mathbf{x}} : \mathbb{T}_{rb}) \Big[\Phi_{\mathrm{RB}}(\bar{\mathbf{x}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}) \ \to \ (\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big(\Phi_{\Delta}(\bar{\mathbf{z}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] \ , \tag{2.21}
$$

$$
(\forall \bar{\mathbf{z}} : \mathbb{Z}) \Big[\Phi_{\Delta}(\bar{\mathbf{z}}) \ \to \ (\exists \bar{\mathbf{x}} : \mathbb{T}_{rb}) \Big(\Phi_{\mathrm{RB}}(\bar{\mathbf{x}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] \ . \tag{2.22}
$$

Similar to Definition 2.11 we have the notions of equality completeness and completion. By $\mathrm{CLS}^{r}_n(x, x_1, \ldots, x_n)$ we denote the conjunction of literals expressing that $x_0, \ldots, x_n$ are red-terms having the same measure but pairwise distinct. Similar to Algorithms 2.4 and 2.6, we have

Algorithm 2.9 (Computation of RLCC in $\mathrm{RB}_{\mathbb{Z}}$). *Input: $\Phi_{\mathrm{RB}} \wedge \theta_{\mathbb{Z}}$ (type and equality complete). Initially set $\Phi_{\Delta} = \theta_{\mathbb{Z}}$. For each term t add the following to Φ_{Δ}.*

- $|t|_{\max} = 1$ *and* $|t|_{\min} = 1$, *if* $t \equiv \mathrm{nil}$;
- $|t|_{\max} = |s|_{\max}$ *and* $|t|_{\min} = |s|_{\min}$ *if* $t = s$ *is present in* Φ_{RB};
- $|t|_{\max} = \max(|t_1|_{\max}, |t_2|_{\max}) + 1$ *and* $|t|_{\min} = \min(|t_1|_{\min}, |t_2|_{\min}) + 1$, *if* $t \equiv \mathrm{black}(t_1, t_2)$;

- $|t|_{\max} = 0$ *and* $|t|_{\min} = 0$ *if* $t \equiv \mathrm{red}(t_1, t_2)$ *and either* $\mathrm{Is}_{\mathrm{red}}(t_1)$ *or* $\mathrm{Is}_{\mathrm{red}}(t_2)$ *are present in* Φ_{RB}*;*
- $|t|_{\max} = |t_1|_{\max} = |t_2|_{\max}$ *and* $|t|_{\min} = |t_1|_{\max} = |t_2|_{\min}$ *if*
  1. $t \equiv \mathrm{red}(t_1, t_2)$*,*
  2. *either* $\mathrm{Is}_{\mathrm{black}}(t_1)$ *or* $\mathrm{Is}_{\mathrm{nil}}(t_1)$ *is present in* Φ_{RB}*,*
  3. *either* $\mathrm{Is}_{\mathrm{black}}(t_2)$ *or* $\mathrm{Is}_{\mathrm{nil}}(t_2)$ *is present in* Φ_{RB}*;*
- $\mathrm{CNT}^r_n(|t|_{\max}, |t|_{\min})$ *if* $\mathrm{CLS}^r_n(t, t_1, \ldots, t_n)$ *is induced by* $\Phi_{\mathrm{RB}} \wedge \theta_{\mathbb{Z}}$ *for some* $t_1, \ldots, t_n$*;*
- $\mathrm{CNT}^b_n(|t|_{\max}, |t|_{\min})$ *if* $\mathrm{CLS}^b_n(t, t_1, \ldots, t_n)$ *is induced by* $\Phi_{\mathrm{RB}} \wedge \theta_{\mathbb{Z}}$ *for some* $t_1, \ldots, t_n$*.*

Theorem 2.7 (RLCC in $\mathrm{RB}_{\mathbb{Z}}$). *Φ_Δ obtained by Algorithm 2.9 is an RLCC for $\Phi_{\mathrm{RB}}/\theta_{\mathbb{Z}}$ and is expressible in a quantifier-free Presburger formula of size linear in the size of $\Phi_{\mathrm{RB}} \wedge \theta_{\mathbb{Z}}$.*

Finally we have

Theorem 2.8 (Decidability of $\mathrm{Th}^\forall(\mathrm{RB}_{\mathbb{Z}})$). *The decision problem for $\mathrm{Th}^\forall(\mathrm{RB}_{\mathbb{Z}})$ is NP-complete.*

Proof. Similar argument as in Section 2.4.6 with the help of Theorem 2.7. □

2.5.2 Algorithm and Example

In this section we consider the insertion operation of red-black trees represented by Algorithm 2.10, a slightly modified version of the algorithm given in [13] . We illustrate the algorithm on the same example as in [13], inserting 4 at the bottom of the tree and showing how the algorithm restores the red-black tree property.

Note that our language does not have an update function to express the relation between the original tree and updated tree if the update happens at an unbounded depth inside the tree. However, we know that the restoring updates will begin at the newly inserted node and traverse upwards to the root and that all local updates will happen on this path. We represent the tree as a sequence of subtrees indexed by nodes on the path from the root to the newly inserted node. We treat the path as a doubly linked list (denoted by T in the algorithm) in which each element contains three fields, *.color*, *.dir* and *.tree*. Field *.color* denotes the type of the node. Field *.dir* indicates whether the subtree at this node is the left child or the right child. Field *.tree* denotes the sibling subtree of this node. We have $root.dir = \perp$ and $root.tree = \perp$. For simplicity, we omit the value field as it has no role in restoring the red-black tree property. We treat *root* and x as iterators and we use array notation $T[x]$ to denote the element pointed to by x. We use $x + 1$ and $x - 1$ to denote previous iterator and next iterator of x, respectively. For example, the statement $x + 3 - 1 = x - 1$ at line 19 means $x.pre.pre.pre.next := x.pre$.

Figure 2-4 shows the results of the operations performed to restore the balanced-tree property after inserting 4. Figure 2-5 gives a more detailed picture of the data structures of the nodes on the

Algorithm 2.10 (RB-Insertion with Left-Rotation and Right-Rotation Integrated).
Input: root, T, x.

```
1:  while (x ≠ root and T[x-1].color = RED) do
2:      if T[x-1].dir=right then
3:          if (T[x-1].tree IS RED) {Case 1} then
4:              T[x-1].tree := BLACK(car(T[x-1].tree),cdr(T[x-1].tree))
5:              T[x-1].color := BLACK
6:              T[x-2].color := RED
7:              x := x-2
8:          else
9:              if (T[x].dir=left) {Case 2} then
10:                 swap(T[x].tree, T[x+1].tree)
11:                 T[x].dir := right
12:                 T[x+1].dir := left
13:             end if
14:             T[x-1].color := BLACK {Case 3}
15:             T[x].tree := RED(T[x].tree, T[x-1].tree)
16:             T[x-1].tree := T[x-2].tree
17:             T[x-1].dir := T[x-2].dir
18:             if (x-2 ≠ root) then
19:                 x-3+1:=x-1
20:             else
21:                 root := x-1
22:             end if
23:         end if
24:     else if (T[x-1].dir=left) then
25:         similar code as the then clause with left and right swapped
26:     end if
27: end while
28: T[root].color := BLACK
```

path from the root to x. Figure 2-4 (b) shows the tree obtained by recoloring. The new violation now corresponds to Case 2 in Algorithm 2.10. Figure 2-4 (c) shows the tree obtained from a left rotation. There is still a violation which corresponds to Case 3 in Algorithm 2.10. Figure 2-4 (d) shows a new red-black tree after a right rotation. Figures 2-5 (b)-(d) show the corresponding changes of the data structure during the run.

2.5.3 Verification Conditions

We now show how to use $\mathscr{L}^{\mathbb{Z}}_{\mathrm{RB}}$ to express the verification conditions for statements restoring that red-black tree property in Algorithm 2.10. Recall that in the algorithm x is an iterator and $T[x]$ is a node pointed by x in a linked list and it contains three fields, *.dir*, *.color* and *.tree*. At the semantic level, however, we view x as an integer index and $T[x]$ as a subtree indexed by x. If $x \neq root$, then $T[x]$.*tree* denotes the sibling tree of $T[x]$, and $T[x-1]$ represents the immediate super-tree containing

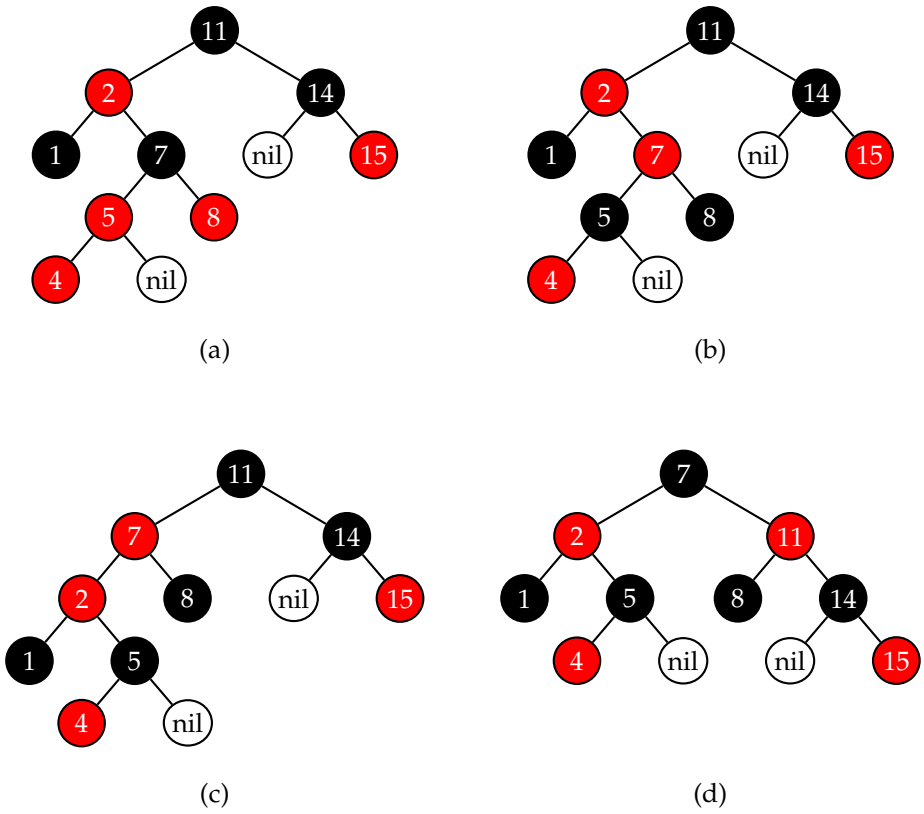


Figure 2-4: A run of RB-Insertion.

$T[x]$. For example, if $T[x].dir$ is right and $T[x-1].color$ is red, then $T[x-1] = \text{red}(T[x], T[x].tree)$. We have three field operators, *.dir*, *.color* and *.tree*. Among them *.dir* can only take three values, $left$, *right* and $\perp$, so expressions involving *.dir* can be removed by disjunctive splitting. Similar for *.color*, but it can be directly expressed in $\mathscr{L}_{\text{RB}}$ as below.

$$\begin{aligned} T[x].color = \text{red} &\stackrel{\text{def}}{=\!=} \text{Is}_{\text{red}}(T[x]) \ , \\ T[x].color = \text{black} &\stackrel{\text{def}}{=\!=} x = \text{nil} \ \vee \ \text{Is}_{\text{black}}(T[x]) \ . \end{aligned}$$

With the help of *.dir*, *.tree* can be expressed in $\mathscr{L}_{\text{RB}}$ as follows.

$$\begin{aligned} T[x].tree = y \neq \perp &\stackrel{\text{def}}{=\!=} x \neq root \ \wedge \ \big((y = \text{car}(T[x-1]) \ \wedge \ T[x].dir = right) \\ &\qquad \vee \ (y = \text{cdr}(T[x-1]) \ \wedge \ T[x].dir = left)\big) \ , \\ T[x].tree = \perp &\stackrel{\text{def}}{=\!=} x = root \ . \end{aligned}$$

Therefore from now on we treat field access expressions as abbreviations in $\mathscr{L}_{\text{RB}}$. Note that we use array and record notations for clarity. At the formula level terms of index access or field access are

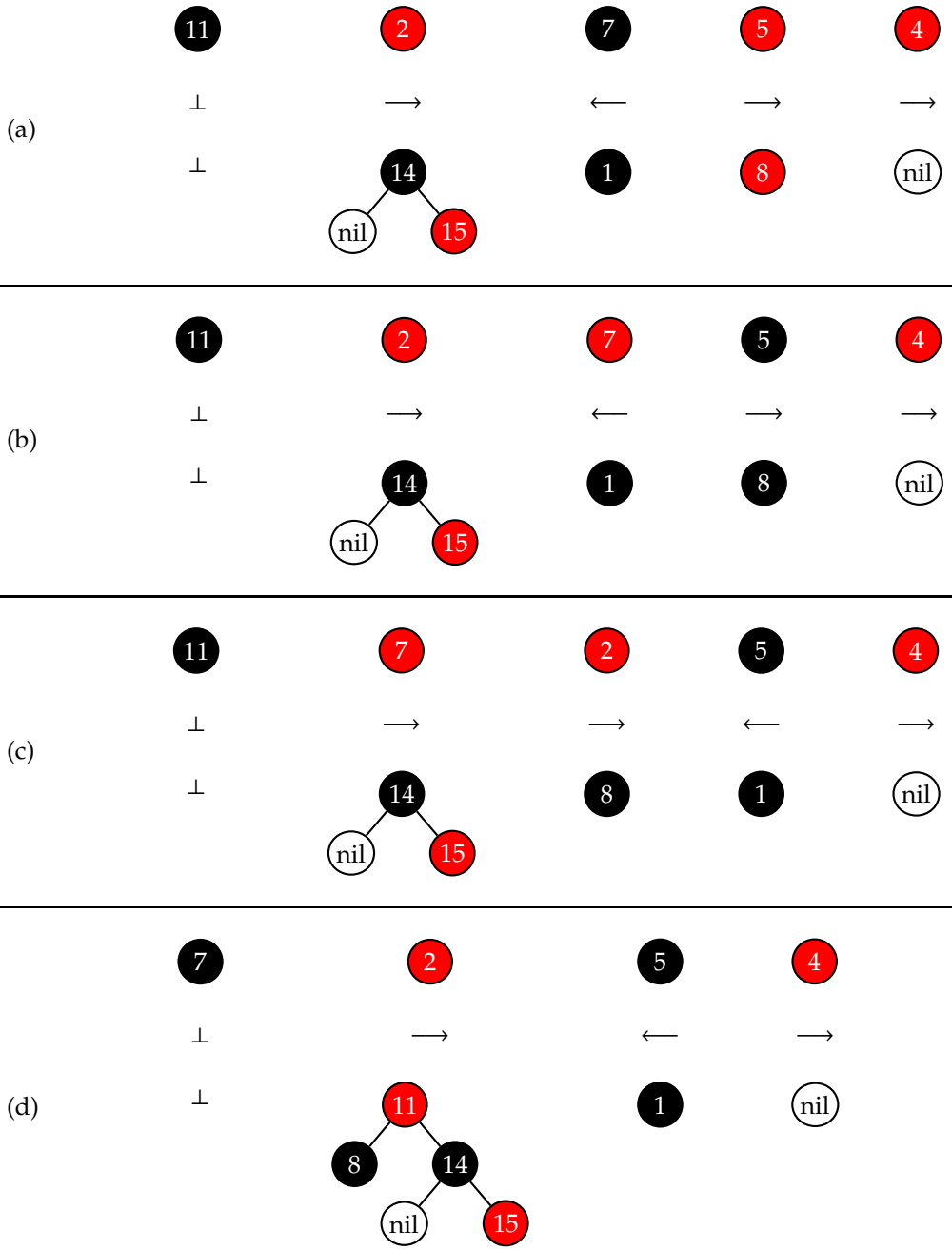


Figure 2-5: Paths from the root of the tree to x. In each of (a)-(d), the first row shows the sequence of nodes from the root to x; the second row shows whether the node above it is a left (←) or right (→) sibling; the third row shows the sibling tree of the node in the top row.

simply variables. For example, $T[x]$, $T[x].tree$, $T[x].color$ and $T[x].dir$ can be represented by variables f_x, g_x, h_x and k_x indexed by x, respectively. Similarly for terms indexed by $x - i$ and $x + i$.

The lack of update functions makes it impossible to express the tree operational semantics precisely in a finite formula. For example, when $T[x]$ is changed, not only should $T'[x]$ appear in $\rho_q(v, v')$, but also all ancestors of $T[x]$. In fact $\rho_q(v, v')$ has an unbounded number of conjuncts of the form $\mathrm{car}(T'[x - i]) = T'[x - i + 1]$ or $\mathrm{cdr}(T'[x - i]) = T'[x - i + 1]$. We can still, however, prove *safety properties* about tree operations with the help of an informal induction. As an example, we show that $\varphi^-_{\mathrm{RB}}(T[x])$, introduced in Section 2.5.1, is an invariant with respect to each code fragment (corresponding to Case 1, 2 or 3 in Algorithm 2.10). This can be obtained by establishing the Hoare triple $\{\varphi\}Q\{\psi\}$ where φ is the pre-condition

$$x \neq root \ \wedge \ x - 1 \neq root \ \rightarrow \ \left(\varphi^-_{\mathrm{RB}}(T[x]) \ \wedge \ \varphi^-_{\mathrm{RB}}(T[x].tree) \ \wedge \ \neg\varphi^-_{\mathrm{RB}}(T[x-1]) \ \wedge \ \varphi^-_{\mathrm{RB}}(T[x-1].tree)\right)$$

ψ is the post-condition $\varphi^-_{\mathrm{RB}}(T[x])$, and Q is a code fragment corresponding to Case 1, 2 or 3. Here we need an invariant $\forall x(x \neq root \rightarrow \varphi^-_{\mathrm{RB}}(T[x].tree))$. This invariant can not be formally proved in our theory because of the universal quantification on indexes. But it is easy to verify that the parametric Hoare triples

$$\{x \neq root \rightarrow \varphi^-_{\mathrm{RB}}(T[x \pm i].tree)\} \quad q \quad \{x \neq root \rightarrow \varphi^-_{\mathrm{RB}}(T[x \pm i].tree)\}$$

can be established for each statement q not modifying index x (see below for the transition relations of those statements). In the following we list *local* transition relations of all statements involving tree update and use guard conditions to simplify those transition relations.

Case 1 is implemented by statements 4-7. The guard conditions are $x \neq root \wedge \mathrm{Is}_{\mathrm{red}}(T[x-1])$ (line 1), $T[x-1].dir = right$ (line 2) and $\mathrm{Is}_{\mathrm{red}}(T[x-1].tree)$ (line 3). Under these conditions the transition relations for statements 4-7 are, respectively,

$$T'[x-1].tree = \mathrm{cdr}(T'[x-2]) = \mathrm{black}(\mathrm{car}(T[x-1].tree), \mathrm{cdr}(T[x-1].tree)) \ , \tag{S-4}$$

$$\mathrm{car}(T'[x-2]) = T'[x-1] = \mathrm{black}(\mathrm{car}(T[x-1]), \mathrm{cdr}(T[x-1])) \ , \tag{S-5}$$

$$T'[x-2] = \mathrm{red}(\mathrm{car}(T[x-2]), \mathrm{cdr}(T[x-2])) \ , \tag{S-6}$$

$$x' = x - 2 \ . \tag{S-7}$$

The composite transition relation for statements 4-7 is

$$\begin{aligned} & T'[x-1].tree = \mathrm{black}(\mathrm{car}(T[x-1].tree), \mathrm{cdr}(T[x-1].tree)) \\ \wedge \quad & T'[x-1] = \mathrm{black}(\mathrm{car}(T[x-1]), \mathrm{cdr}(T[x-1])) \\ \wedge \quad & T'[x-2] = \mathrm{red}(T'[x-1], T'[x-1].tree) \\ \wedge \quad & x' = x - 2 \ . \end{aligned}$$

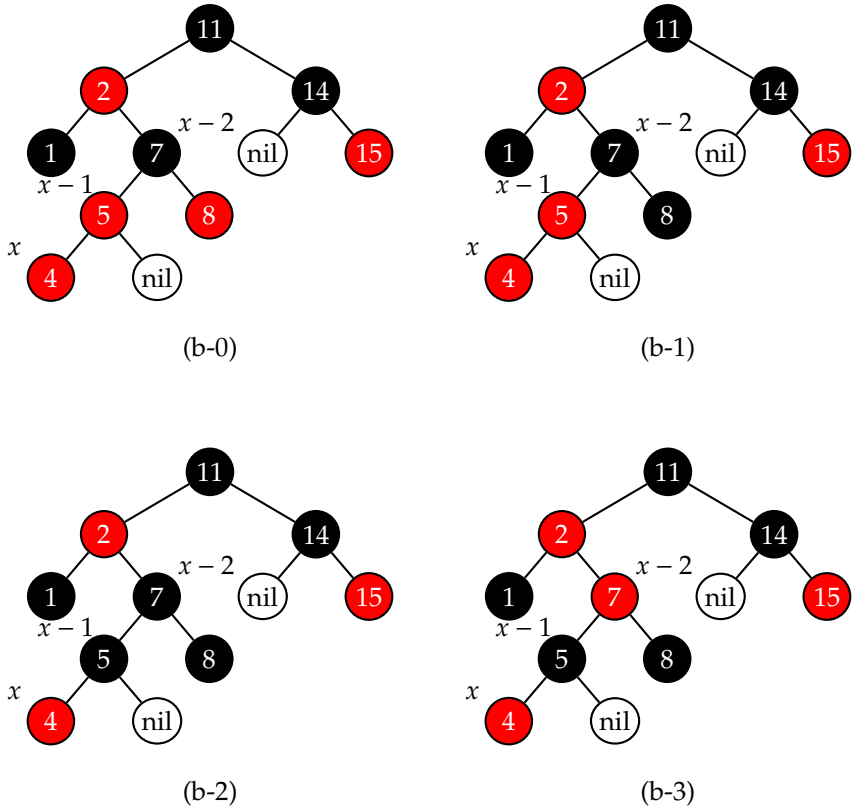


Figure 2-6: A detailed run of RB-Insertion step (b).

Recall that $T[x]$, $T[x].tree$, $T[x].color$ and $T[x].dir$ are just more informative aliases of indexed variables f_x, g_x, h_x and k_x, respectively. Similarly for terms indexed by $x-1$ and $x-2$. The next state variable for $T[x]$ should be $T'[x']$, but by default we write $T'[x]$ when $x' = x$. When we do transition relation composition, statement 7 requires us to hard code the integer indexing properties in the formula by adding equalities like $T'[x-1] = T'[x'+1]$, $T'[x-2].tree = T'[x'].tree$ and so on. To save space, however, we omit them in the above example.

Figure 2-6 illustrates Case 1 by a run on tree (a) in Figure 2-4 (copied as (b-0)). Trees (b-1), (b-2) and (b-3) are the outcomes of statements 4, 5 and 6, respectively.

The code fragment for Case 2 consists of statements 10-12. We take into account the conditions $T[x-1].color = \text{red}$ (line 1), $T[x-1].dir = right$ (line 2) and $T[x].dir = left$ (line 9). Under these condition the transition relations for statements 10-12 are, respectively,

$$\begin{aligned} \text{cdr}(T'[x-1]) = T'[x] \ \wedge \ & (T'[x+1].tree = \text{cdr}(T'[x]) = T[x].tree) \\ \wedge \ & (T'[x].tree = \text{car}(T'[x-1]) = T[x+1].tree) \ , \end{aligned} \tag{S-10}$$

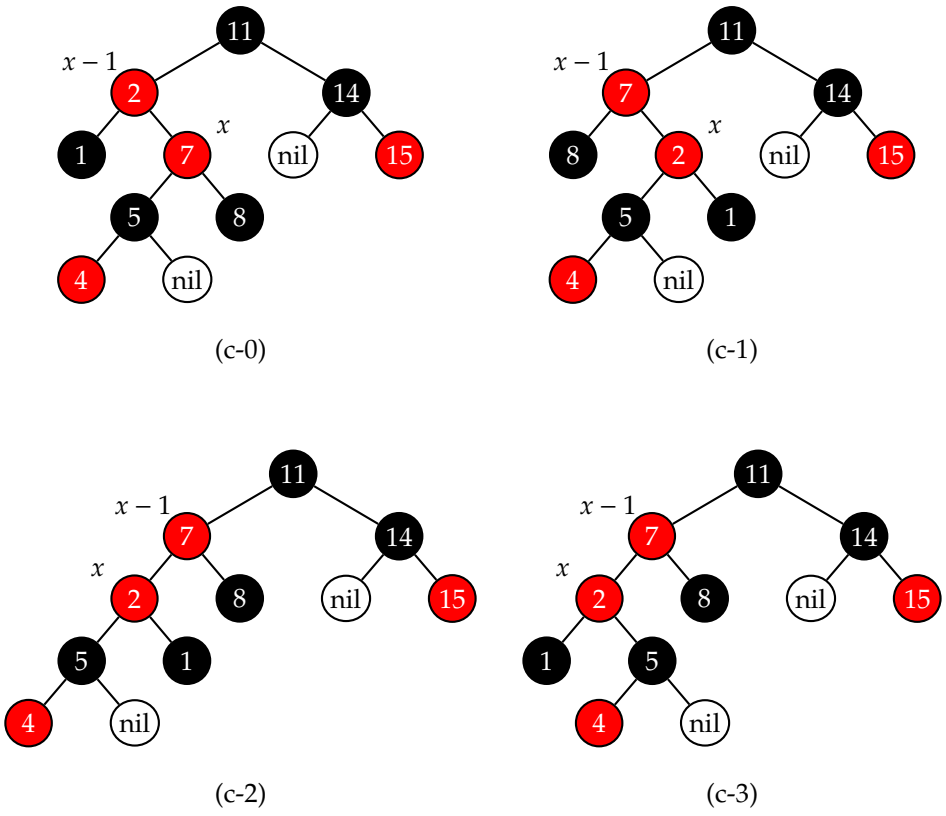


Figure 2-7: A detailed run of RB-Insertion step (c).

$$T'[x].dir = right \ \wedge \ T'[x-1] = \text{red}(\text{cdr}(T[x-1]), \text{car}(T[x-1])) \ , \tag{S-11}$$

$$T'[x+1].dir = left \ \wedge \ \text{car}(T'[x-1]) = T'[x] \\ \wedge \ T'[x] = \text{red}(\text{cdr}(T[x]), \text{car}(T[x])) \ . \tag{S-12}$$

The composite transition relation for statements 10-12 is

$$\begin{aligned} & T'[x+1].tree = T[x].tree \ \wedge \ T'[x].tree = T[x+1].tree \\ \wedge \quad & T'[x].dir = right \ \wedge \ T'[x-1] = \text{red}(T'[x], T[x+1].tree) \\ \wedge \quad & T'[x+1].dir = left \ \wedge \ T'[x] = \text{red}(T[x].tree, \text{car}(T[x])) \ . \end{aligned}$$

Figure 2-7 illustrates Case 2 by a run on tree (b) in Figure 2-4 (copied as (c-0)). Trees (c-1), (c-2) and (c-3) are the outcomes of statements 10, 11 and 12, respectively. Recall that we ignored value labels at internal nodes as they are irrelevant to the red-black tree properties. But the binary search tree property may be violated without adjusting the value labels. So for the sake of illustration we switched positions of 2 and 7 in Figure 2-7 (c-1) although statement 10 does not have this effect.

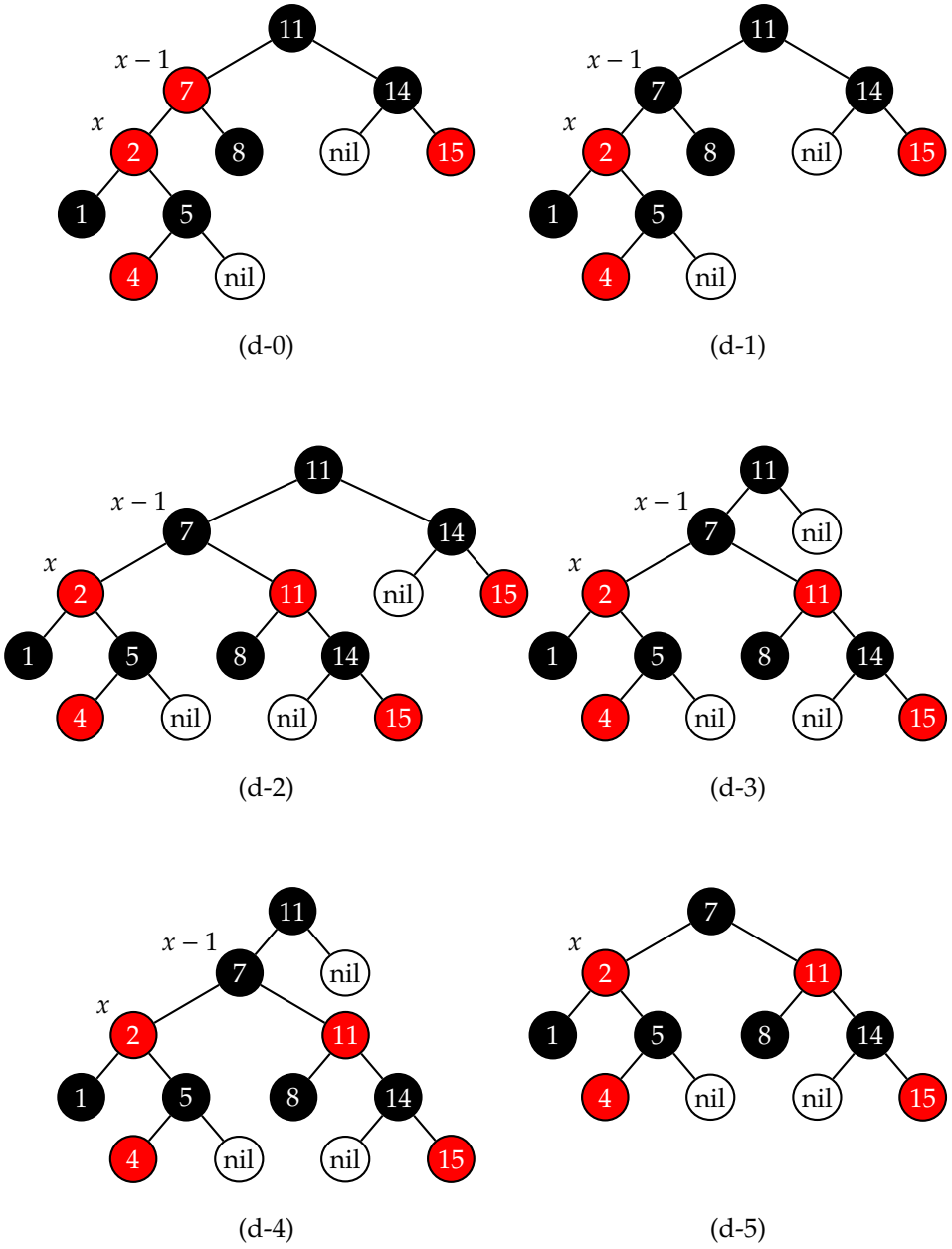


Figure 2-8: A detailed run of RB-Insertion step (d) with $x - 2 = root$.

Case 3 consists of statements 14-21. We take into account the conditions $T[x-1].color = \text{red}$ (line 1), $T[x-1].dir = right$ (line 2) and $T[x].dir = right$ (line 11). Under these conditions the transition relations for statements 14-21 are, respectively,

$$\text{car}(T'[x-2]) = T'[x-1] = \text{black}(\text{car}(T[x-1]), \text{cdr}(T[x-1])) \ , \tag{S-14}$$

$$\mathrm{cdr}(T'[x-1]) = T'[x].tree = \mathrm{red}(T[x].tree, T[x-1].tree) \ , \tag{S-15}$$

$$T'[x-1].tree = T[x-2].tree \ \wedge \ (x-2 \neq root \ \rightarrow \ \mathrm{cdr}(T'[x-2]) = T[x-2].tree) \ , \tag{S-16}$$

$$T'[x-1].dir = T[x-2].dir \ \wedge \ \Big(x-2 \neq root \ \wedge \ T[x-1].dir \neq T[x-2].dir \ \rightarrow \ T'[x-2] = \mathrm{black}(\mathrm{cdr}(T[x-2]), \mathrm{car}(T[x-2]))\Big) \ , \tag{S-17}$$

$$x'-2 = x-3 \ \wedge \ \Big((\mathrm{cdr}(T'[x-3]) = T[x-1] \ \wedge \ T[x-2].dir = left) \ \vee \ (\mathrm{car}(T'[x-3]) = T[x-1] \ \wedge \ T[x-2].dir = right)\Big) \ , \tag{S-19}$$

$$x'-1 = root \ . \tag{S-21}$$

Assuming $x-2 = root$, the composite transition relation for statements 14-21 is

$$\begin{aligned} & \mathrm{car}(T'[x-2]) = T'[x-1] = \mathrm{black}(\mathrm{car}(T[x-1]), T'[x].tree) \\ \wedge \quad & T'[x].tree = \mathrm{red}(T[x].tree, T[x-1].tree) \\ \wedge \quad & T'[x-1].tree = T[x-2].tree \ \wedge \ T'[x-1].dir = T[x-2].dir \\ \wedge \quad & x'-1 = root \ . \end{aligned}$$

Assuming $x-2 \neq root$, the composite transition relation for statements 14-21 is

$$\begin{aligned} & \mathrm{car}(T'[x-2]) = T'[x-1] = \mathrm{black}(\mathrm{car}(T[x-1]), T'[x].tree) \\ \wedge \quad & T'[x].tree = \mathrm{red}(T[x].tree, T[x-1].tree) \\ \wedge \quad & T'[x-1].tree = T[x-2].tree \ \wedge \ \mathrm{cdr}(T'[x-2]) = T[x-2].tree \\ \wedge \quad & T'[x-1].dir = T[x-2].dir \ \wedge \ \Big(T[x-1].dir \neq T[x-2].dir \ \rightarrow \ T'[x-2] = \mathrm{black}(\mathrm{cdr}(T[x-2]), \mathrm{car}(T[x-2]))\Big) \\ \wedge \quad & x'-2 = x-3 \ \wedge \ \Big((\mathrm{cdr}(T'[x-3]) = T[x-1] \ \wedge \ T[x-2].dir = left) \ \vee \ (\mathrm{car}(T'[x-3]) = T[x-1] \ \wedge \ T[x-2].dir = right)\Big) \ . \end{aligned}$$

Figure 2-8 illustrates Case 3 by a run on tree (c) in Figure 2-4 (copied as (d-0)). Trees (d-1)-(d-5) are the outcomes of statements 14-17 and 21, respectively, under the assumption that $x-2 = root$. Here (d-3) and (d-4) are the same because $T[x-1].dir = T[x-2].dir$ and hence statement 17 has no effect. Figures 2-9 and 2-10 illustrate Case 3 under the assumption $x-2 \neq root \wedge T[x-2].dir = right$. Trees (d'-1)-(d'-5) corresponds to the outcomes of statements 14-17 and 19, respectively. Similarly as before (d'-3) and (d'-4) are the same because $T[x-1].dir = T[x-2].dir$. Figures 2-11 and 2-12 illustrate Case 3 under the assumption $x-2 \neq root \wedge T[x-2].dir = left$. Trees (d''-1)-(d''-5) corresponds to the outcomes of statements 14-17 and 19, respectively. As before we keep the binary search tree property by adjusting value labels 11, 16 and -16.

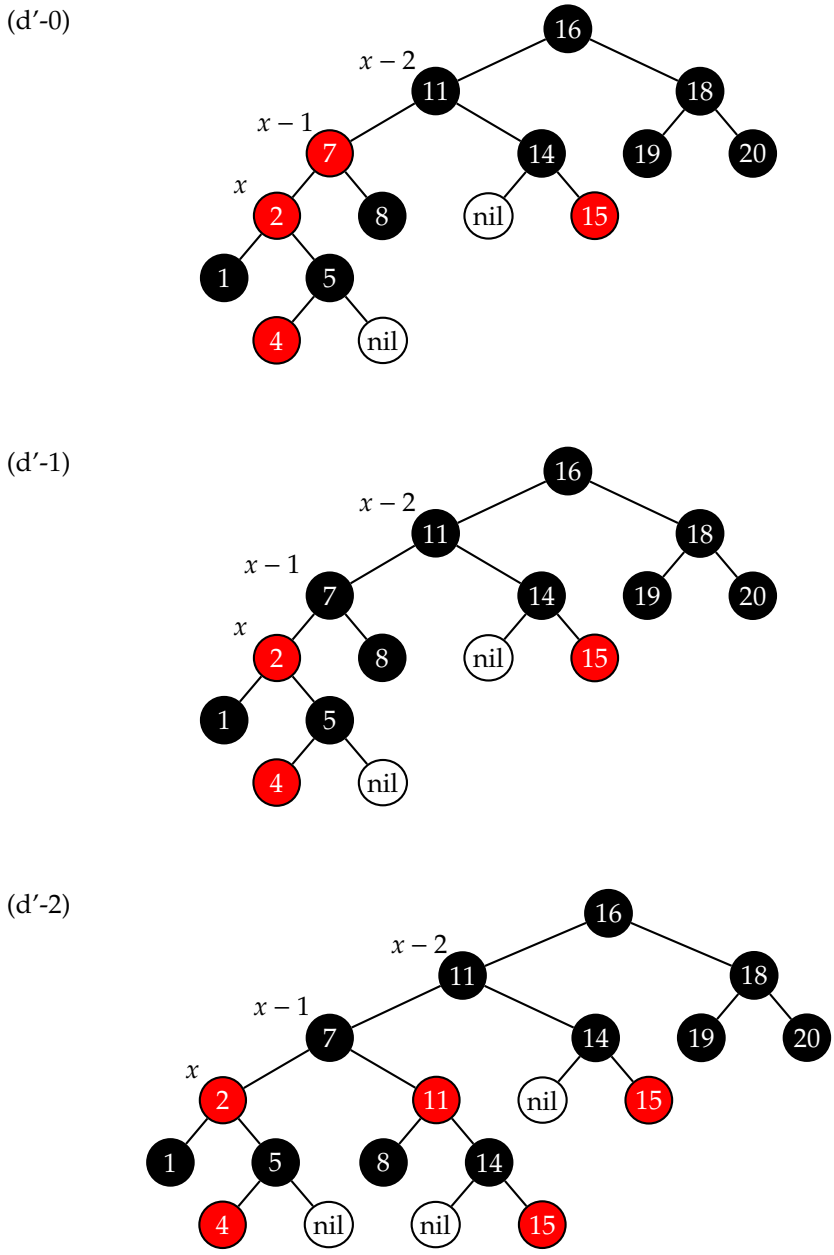


Figure 2-9: A detailed run of RB-Insertion step (d) with $x - 2 \neq root \wedge T[x-2].dir = right$.

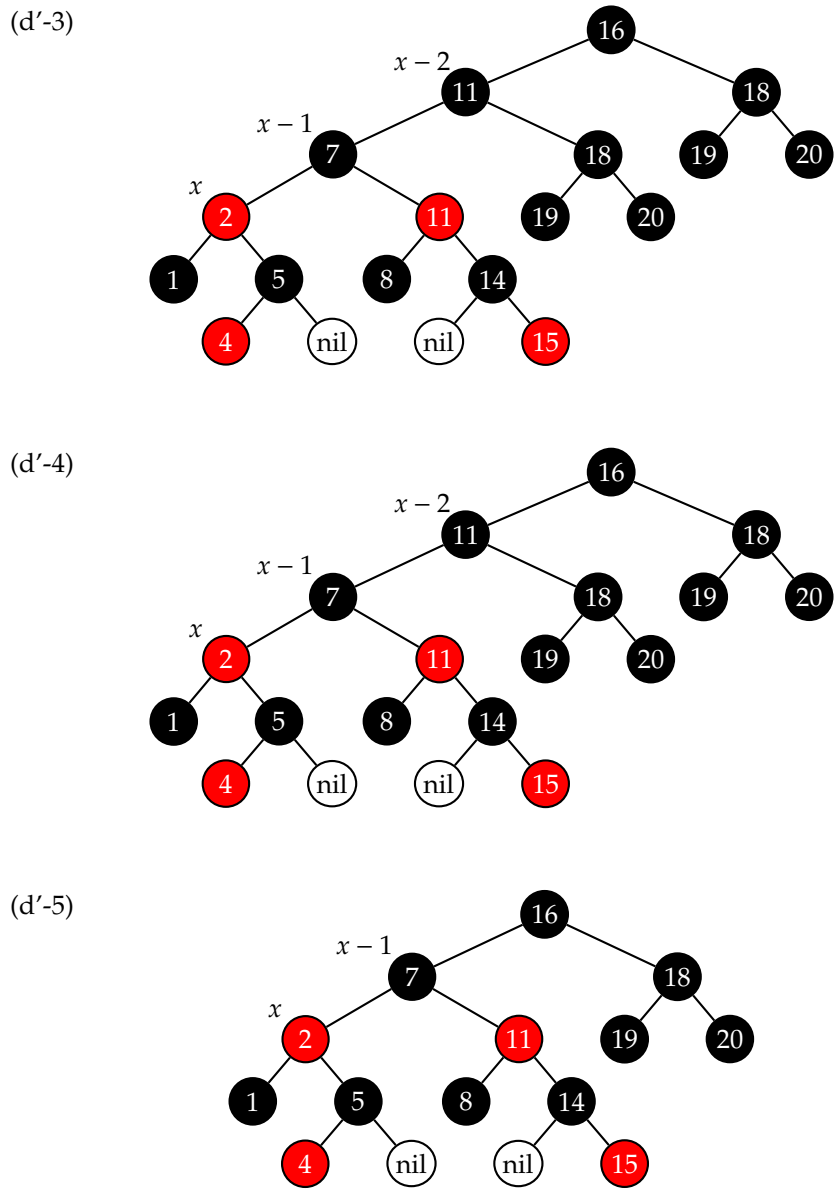


Figure 2-10: A detailed run of RB-Insertion step (d) with $x - 2 \neq root \wedge T[x-2].dir = right$ (continued).

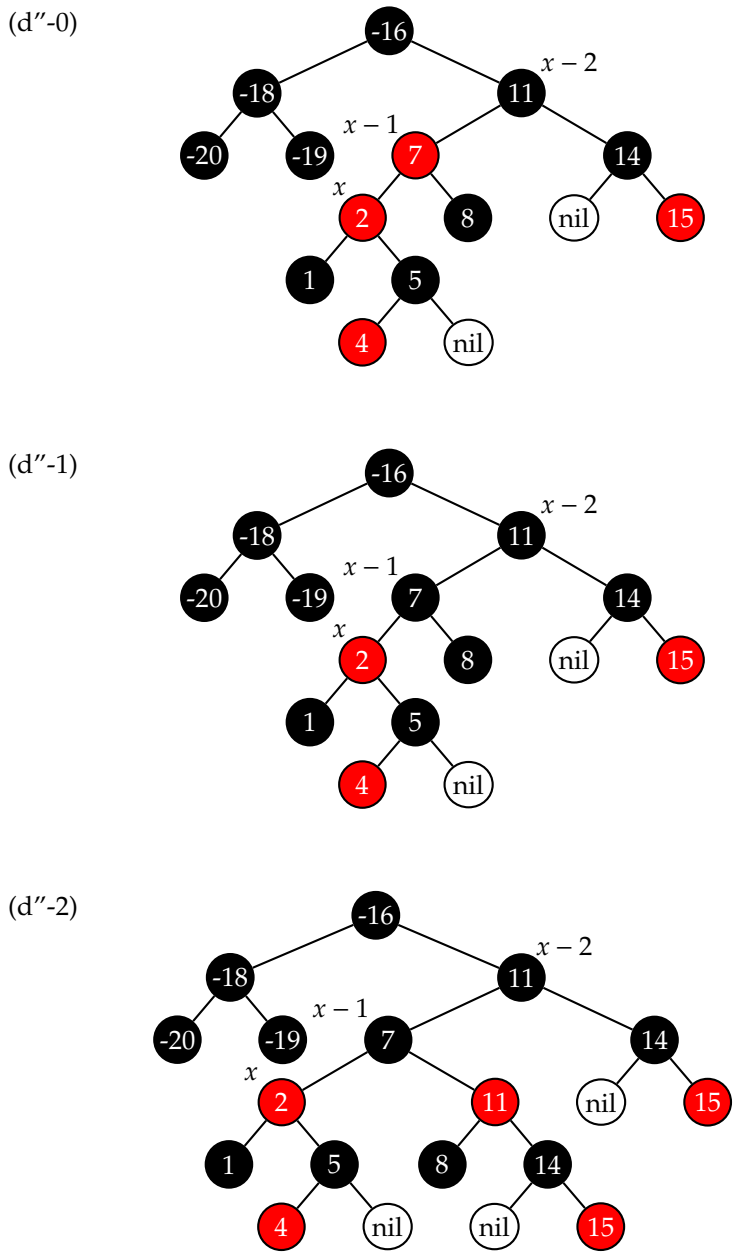
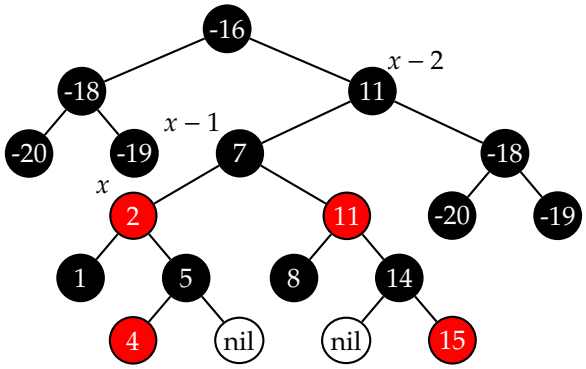
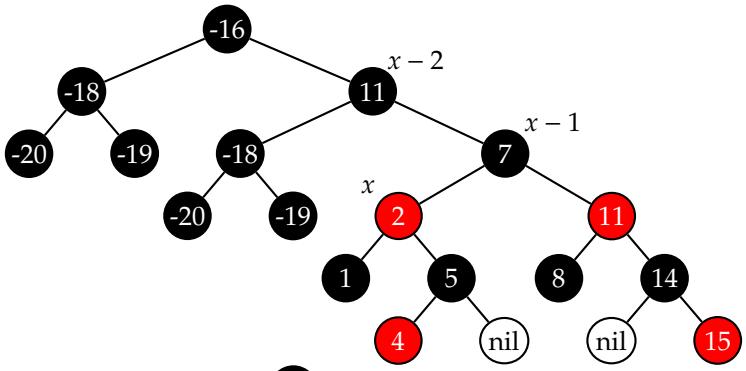


Figure 2-11: A detailed run of RB-Insertion step (d) with $x - 2 \neq root \wedge T[x-2].dir = left$.

(d″-3)



(d″-4)



(d″-5)

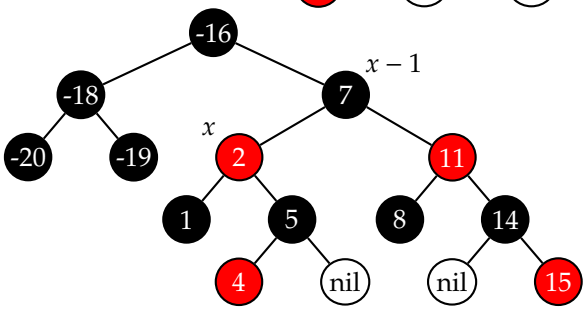


Figure 2-12: A detailed run of RB-Insertion step (d) with $x - 2 \neq root \wedge T[x-2].dir = left$ (continued).

2.6 A New Quantifier Elimination Procedure for Th(TA)

In this section we present a new quantifier elimination algorithm for Th(TA), the first-order theory of term algebras, and show that the algorithm only needs exponential time to eliminate a block of quantifiers of the same kind. The algorithm works mainly in the constructor language while using selectors as auxiliary tools. It is the basis for the elimination procedure for the extended theory presented in the next section. In this section we assume a finite constant domain. We drop the subscript $\mathbb{T}$ in this section.

Primitive Form. Recall that eliminating of arbitrary quantifiers reduces to eliminating existential quantifiers from formulas in the form

$$(\exists \bar{\mathbf{x}}) \left[A_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \cdots \wedge A_n(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \right], \tag{1.1}$$

where $A_i(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ $(0 < i \leq n)$ are literals [19]. Taking into account the specific properties of term algebras, we can assume that the literals A_i are not of the form $x = t$ when x does not appear inside selectors. For $\exists x(x = t \wedge \Phi(x, \bar{\mathbf{y}}))$ simplifies to $\Phi(t, \bar{\mathbf{y}})$ if x does not occur in t, to $\exists x \Phi(x, \bar{\mathbf{y}})$ if $t \equiv x$, and to false by Axiom (A1) if t is a term properly containing x. In all algorithms we assume these simplifications are performed in the end of each step to automatically restore the primitive form.

Nondeterminism. Recall that all transformations in quantifier elimination are done on formulas of the form (1.1). Again as in the presentation for quantifier-free theories, whenever we say "guess ϕ", we mean to add a valid (with respect to the context) disjunction $\bigvee_i \phi_i$ (where ϕ is one of the disjuncts) to the matrix of (1.1). When we replace ϕ by $\bigvee_i \phi_i$ or directly introduce $\bigvee_i \phi_i$, it should be understood that an implicit disjunctive splitting is carried out and we work on each resultant disjunct of the form (1.1) "simultaneously".

Simplification. For simplicity, in the description of algorithms, we omit tester literals unless they are needed for the correctness proof. We may also assume that the matrix of (1.1) is type complete and basic simplifications are carried out whenever applicable: for a nonempty selector sequence L, we replace $Lx \neq x$ by true and $Lx = x$ by false; if $t(x)$ is a term properly containing x and x does not appear in selector terms, we replace $t(x) \neq x$ by true and $t(x) = x$ by false.

Notation. In the algorithm we use the following notation: $\bar{\mathbf{x}}$ denote the set of existentially quantified variables; $\bar{\mathbf{y}}$ denote the set of parameters (implicitly universally quantified variables); s, t, u denote TA-terms; L, F, G, H denote (possibly empty) selector sequences; f, g, h, p, q denote index functions with ranges clear from the context; i, j, k, l denote indexes; numerical superscripts are parenthesized. Index functions are used to differentiate multiple occurrences of the same variables.

In each step of the transformations the algorithm manipulates the formula $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ to produce a version of the same form (or multiple versions of the same form in case disjunctions are introduced), and thus in each step $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ refers to the updated version rather than to the original input formula.

Outline. The elimination is performed as follows. A sequence of equivalence-preserving transformations will bring the input formula into a disjunction of formulas in solved form which have solutions under any instantiation of parameters. Therefore, the whole block of existential quantifiers $\exists \bar{\mathbf{x}}$ can be eliminated by removing all literals containing $\bar{\mathbf{x}}$ in the matrix.

Definition 2.16 (Solved Form in TA**).** *We say* $\Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ *is* solved *in* $\bar{\mathbf{x}}$*, if* $\bar{\mathbf{x}}$ *do not appear in equalities, are not asserted to be constants and are not inside selector terms. We say* $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ *is in solved form if* $\Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ *is solved in* $\bar{\mathbf{x}}$*.*

A solved form can be obtained by the following normalization procedure. The normalization can be viewed as an explicit syntactical procedure comparable with the DAG construction and computation of the bidirectional closure.

Algorithm 2.11 (Normalization in TA**).** *Input:* $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$*.*

1. *Type Completion. Guess a type completion of* $\Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ *and simplify every selector term to a proper one.*
2. *Selector Elimination. Replace all selector terms containing* $\bar{\mathbf{x}}$ *by the corresponding equivalent constructor terms according to Axiom (A6).*
3. *Decomposition. Call Algorithm 2.12 to decompose equalities and disequalities between constructor terms and equalities containing* $\bar{\mathbf{x}}$*.*
4. *Constant Elimination. If some* $x \in \bar{\mathbf{x}}$ *is asserted to be a constant (i. e.,* $\mathrm{Is}_{\mathsf{A}}(x)$ *appears), we instantiate* x *to each constant to eliminate* $\exists x$ *since* $\mathcal{A}$ *is finite.*

After step (1) of Algorithm 2.11 terms can have three forms:

$$(i)\ Lx\ , \qquad (ii)\ Ly\ , \qquad (iii)\ \alpha(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}))\ ,$$

where L may be empty. Terms of the form (iii) are constructor terms built recursively from terms in (i)-(ii) using non-nullary constructors with $t_i(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ of the form (i), (ii) or (iii). These three forms give rise to the following six types of equality literals:

$$Lx = L'x'\ , \tag{2.23}$$

$$Lx = L'y\ , \tag{2.24}$$

$$Lx = \alpha(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}))\ , \tag{2.25}$$

$$Ly = L'y' \ , \tag{2.26}$$

$$Ly = \alpha(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \ , \tag{2.27}$$

$$\alpha(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) = \alpha(t'_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t'_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \ . \tag{2.28}$$

Similarly, we have six types of disequalities, the negations of (2.23)-(2.28).

Step (2) transforms equalities of the forms (2.23)-(2.25), and similarly for disequalities of the same form. Thus after application of this step, we can assume that $\bar{\mathbf{x}}$ does not appear inside selector terms, that is, equality literals have the forms

$$Ly = L'y' \ , \tag{2.29}$$

$$Ly = \alpha(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \ , \tag{2.30}$$

$$\alpha(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) = \alpha(t'_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t'_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \ , \tag{2.31}$$

and disequality literals are in the forms of the negations of (2.29)-(2.31) and in the forms of

$$x \neq x' \ , \qquad x \neq Ly \ , \qquad x \neq \alpha(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_{\mathrm{ar}(\alpha)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \ .$$

Step (2) may generate literals like $x = t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$. The reinstatement of the primitive form, however, does not put any of $\bar{\mathbf{x}}$ inside selector terms. Step (2) may also linearly increase the size of the matrix. In general, elimination of selectors adds more existential quantifiers of sort term. The newly added quantifiers, however, will be removed in one step together with the original ones. The following example illustrates step (2).

Example 2.16 (Elimination of Selectors). *Step* (2)*, selector elimination, first converts the formula*

$$(\exists x) \Big[\ \mathrm{car}(x) = y_2 \ \wedge \ \mathrm{cdr}(x) \neq y_2 \ \wedge x \ \neq y_3 \ \Big] \tag{2.32}$$

into

$$(\exists x_1 \exists x_2) \Big[\ x_1 = y_2 \ \wedge \ x_2 \neq y_2 \ \wedge \ \mathrm{cons}(x_1, x_2) \neq y_3 \ \Big] \ , \tag{2.33}$$

which, by substitution of x_1 for y_2, simplifies to

$$(\exists x_2) \Big[\ x_2 \neq y_2 \ \wedge \ \mathrm{cons}(y_2, x_2) \neq y_3 \ \Big] \ . \tag{2.34}$$

Step (3) converts equalities and disequalities of form (2.31) and equalities of form (2.30). After step (3) the matrix, if it did not simplify to false, contains only literals of the form

$$x \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ , \qquad Ly \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ , \qquad Ly = L'y' \ . \tag{2.35}$$

where t is: (i) existentially quantified variables $\bar{\mathbf{x}}$, (ii) implicitly universally quantified parameters $\bar{\mathbf{y}}$, (iii) selector terms of parameters in the form Ly ($y \in \bar{\mathbf{y}}$), (iv) constants in $\mathcal{A}$, or (v) constructor terms built recursively from terms in (i)-(iv) using non-nullary constructors.

Algorithm 2.12 (Decomposition of Constructor Terms). *Input:*$(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$.

1. *Decomposition of Equalities between Constructor Terms. Replace*

$$\alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha)}) = \alpha(t'_1, \ldots, t'_{\mathrm{ar}(\alpha)}) \tag{2.36}$$

   by $\bigwedge_{1 \le i \le \mathrm{ar}(\alpha)} t_i = t'_i$. *Repeat until no equality of the form* (2.36) *appears.*

2. *Decomposition of Disequalities between Constructor Terms. Replace*

$$\alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha)}) \neq \alpha(t'_1, \ldots, t'_{\mathrm{ar}(\alpha)}) \tag{2.37}$$

   by $\bigvee_{1 \le i \le \mathrm{ar}(\alpha)} t_i \neq t'_i$. *Repeat until no disequality of the form* (2.37) *appears.*

3. *Decomposition of Equalities Containing* $\bar{\mathbf{x}}$. *Solve equalities of the form* $Ly = t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, *where* $t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ *is a constructor term containing* $\bar{\mathbf{x}}$, *in terms of* Ly *such that the result is a set of equalities in the selector language.*

In step (1) of Algorithm 2.12, recall that literals like $x = t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ can always be eliminated together with $(\exists x)$ and hence after this step, we can assume no such literals appear in the matrix.

After step (2) we can assume that literals have one of the following forms:

$$Ly = L'y' \ , \tag{2.38}$$

$$Ly \neq L'y' \ , \tag{2.39}$$

$$x \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ , \tag{2.40}$$

$$Ly \neq \alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha)}) \ , \tag{2.41}$$

$$Ly = \alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha)}) \ . \tag{2.42}$$

Step (3) solves equalities of form (2.42), and thus we are left only with literals of forms (2.38)-(2.41). Notice that these are the same as those in (2.35), where $Ly \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ represents both (2.39) and (2.41).

The following example illustrates how equalities are solved.

Example 2.17 (Solving Equalities). *The literal* $\mathrm{cdr}(y) = \mathrm{cons}(\mathrm{cons}(x_1, y_1), y_2)$ *is converted to the solution set*

$$x_1 = \mathrm{car}(\mathrm{car}(\mathrm{cdr}(y))), \qquad y_1 = \mathrm{cdr}(\mathrm{car}(\mathrm{cdr}(y))), \qquad y_2 = \mathrm{cdr}(\mathrm{cdr}(y)).$$

Given an input formula $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, the normalization procedure in Algorithm 2.11 lets us effectively eliminate all quantifiers $(\exists \bar{\mathbf{x}})$ in one step. Indeed, after application of Algorithm 2.11 we can assume, according to (2.35), that $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is in the form

$$(\exists \bar{\mathbf{x}}) \Big[\bigwedge_i x_{p(i)} \neq t_i(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big] \wedge \bigwedge_k F_k y_{f(k)} \neq u_k(\bar{\mathbf{y}}) \wedge \bigwedge_l G_l y_{g(l)} = H_l y_{h(l)} \ . \quad (2.43)$$

Here t_i, s_j are:

(i) existentially quantified variables $\bar{\mathbf{x}}$,

(ii) implicitly universally quantified parameters $\bar{\mathbf{y}}$,

(iii) selector terms of parameters in the form Ly $(y \in \bar{\mathbf{y}})$,

(iv) constants in $\mathcal{A}$, or

(v) constructor terms built recursively from terms in (i)-(iv) using constructors,

where we require that s_j contain at least one occurrence of a variable in $\bar{\mathbf{x}}$ (otherwise, the corresponding literal should have been moved out of the scope of $(\exists \bar{\mathbf{x}})$). The term $u_k(\bar{\mathbf{y}})$ can be one of (ii)-(v) above, where in (v) recursion is limited to (ii)-(v).

We claim that the first part of (2.43),

$$(\exists \bar{\mathbf{x}}) \Big[\bigwedge_i x_{p(i)} \neq t_i(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big] \ , \quad (2.44)$$

is valid, and hence (2.43) is equivalent to

$$\bigwedge_k F_k y_{f(k)} \neq u_k(\bar{\mathbf{y}}) \ \wedge \ \bigwedge_l G_l y_{g(l)} = H_l y_{h(l)} \ . \quad (2.45)$$

Thus the algorithm for elimination of quantifiers can be given as

Algorithm 2.13 (Quantifier Elimination in TA**).** *Input:* $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$.

1. *Call Algorithm 2.11 to normalize* $(\exists \bar{\mathbf{x}})\, \Phi(\bar{\mathbf{x}}, \bar{\mathbf{y}})$.

2. *Remove* $(\exists \bar{\mathbf{x}})$ *and all literals containing* $\bar{\mathbf{x}}$.

Theorem 2.9 (Soundness of Algorithm 2.13). *All transformations in Algorithm 2.13 keep equivalence.*

Theorem 2.10 (Complexity of Algorithm 2.13). *Algorithm 2.13 removes a block of quantifiers in time* $2^{O(n)}$.

Example 2.18 (Quantifier Elimination in TA**).** *Let us look at an example in* List*. Consider a type complete formula*

$$(\exists x)\left[\ \mathrm{cons}(\mathrm{car}(x), y_1) = y_2 \ \wedge\ y_2 \neq x\ \right]\ . \tag{2.46}$$

(Because of the assumption of type completeness, we know $\mathrm{Is}_{\mathrm{cons}}(y_2)$ *and* $\mathrm{Is}_{\mathrm{cons}}(x)$ *must be present in the side condition. We omit type information of* y_1*, because it is irrelevant here.) Step* (2) *of Algorithm 2.11 removes the selectors on x by converting* (2.46) *into*

$$(\exists x_1 \exists x_2)\left[\ \mathrm{cons}(x_1, y_1) = y_2 \ \wedge\ y_2 \neq \mathrm{cons}(x_1, x_2)\ \right]\ . \tag{2.47}$$

Step (3) *of Algorithm 2.11 (Algorithm 2.12) solves the equality* $\mathrm{cons}(x_1, y_1) = y_2$*, with solution set*

$$\{\, x_1 = \mathrm{car}(y_2),\ y_1 = \mathrm{cdr}(y_2)\,\}\ ,$$

resulting in

$$(\exists x_1 \exists x_2)\left[\ \mathrm{car}(y_2) = x_1 \ \wedge\ \mathrm{cdr}(y_2) = y_1 \ \wedge\ y_2 \neq \mathrm{cons}(x_1, x_2)\ \right]\ , \tag{2.48}$$

which, by standard substitution and quantifier manipulation, reduces to

$$(\exists x_2)\left[\ y_2 \neq \mathrm{cons}(\mathrm{car}(y_2), x_2)\ \right] \ \wedge\ \mathrm{cdr}(y_2) = y_1\ . \tag{2.49}$$

As $(\exists x_2)[\ y_2 \neq \mathrm{cons}(\mathrm{car}(y_2), x_2)\ ]$ *is in solved form, it is valid, and hence* (2.49) *reduces to* $\mathrm{cdr}(y_2) = y_1$*, or more formally, to* $\mathrm{cdr}(y_2) = y_1 \wedge \mathrm{Is}_{\mathrm{cons}}(y_2)$*.*

2.7 Decision Procedures for Quantified Theories

In this section we present decision procedures for the theory $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$ of term algebras with integers and parameters. In Section 2.7.1 we first refine the notions and techniques from Section 2.4 for the construction of a quantifier elimination procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$, and then, in Section 2.7.2, present the quantifier elimination procedure itself. Throughout Sections 2.7.1 and 2.7.2 we assume a finite language. In Section 2.7.3 we further generalize the result to structures whose constant domain has an internal structure and admits quantifier elimination. In Section 2.7.4 we discuss how the procedure can be adapted for infinite languages.

2.7.1 Term Algebras with Integers and Parameters

In this section we refine the notions and techniques from Section 2.4 for the construction of a quantifier elimination procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$, which is given in the next section.

We first refine the notion of equality completion. As we have seen, to get an RLCC (LCC) we need to express in Presburger arithmetic the set of legitimate lengths such that a certain number of distinct terms of any length in the set can co-exist. This can be supported by equality completion. Equality completion, however, in general introduces more literals, especially disequalities, which may again destroy the completion because it may cause generation of new terms in the subsequent operation (see *disequality splitting* in Algorithm 2.16). To avoid compromising convergence, we introduce the notion of *clusters* which is weaker than equality completion but contains sufficient information to allow extracting counting constraints. Intuitively, it suffices to have the equality information only between terms of the same length and of the same type.

Definition 2.17 (Clusters). *Let Φ be a conjunction of literals in $\mathscr{L}_{\mathbb{T}}^{\mathbb{Z}}$. Term equality literals appearing in Φ induce equivalence classes on* TA*-terms occurring in Φ. Let $[t]$ denote such an equivalence class containing the term t. We say that $C = \{[t_0], \ldots, [t_n]\}$ is an* α-cluster *(of Φ) if $CLS_n^\alpha(t'_0, \ldots, t'_n)$ is induced by Φ, i. e., Φ expresses that $t_0, \ldots, t_n$ are pairwise distinct α-terms of the same length.*

The notion of clusters is syntactic modulo $=_{\mathbb{T}}$ and $=_{\mathbb{Z}}$ (equality substitution). For example,

$$t = s \ \wedge \ s = u \ \wedge \ u \neq v \ \wedge \ |t| = |v| \ \wedge \ \mathrm{Is}_\alpha(t, s, u, v) \ , \tag{2.50}$$

induces the α-cluster $\{[t], [v]\}$ (where $[t] = \{t, s, u\}$, $[v] = \{v\}$). Formally speaking, a cluster C_Φ is induced by the closure of Φ under equality substitution. We chose this definition to limit the form of disequalities generated in transformations of Algorithm 2.16.

Below we will drop the subscript Φ if Φ is clear from the context. For clarity, we view a cluster as a set consisting of terms that are chosen representatives of their corresponding equivalence classes. The choice of representatives is arbitrary unless stated otherwise. For example, $\{[t_0], \ldots, [t_n]\}$ will also be written simply as $\{t_0, \ldots, t_n\}$.

A cluster is *maximal* if no superset of it is a cluster. A cluster C is *closed* if C is maximal and it is disjoint with any other maximal clusters. Two clusters C and C' are called *connected* if there exists $t \in C$ and $t' \in C'$ (or more formally $[t] \in C$ and $[t'] \in C'$) such that either (i) $t = t'$ occurs in the defining formula (i. e., C and C' intersect), or (ii) $t \neq t'$ occurs in the defining formula while $|t| \neq |t'|$ does not. Two clusters are called *mutually independent* if they are not connected. The *size* of a cluster is the number of equivalence classes it contains. The *rank* of a cluster C, written $\mathrm{rk}(C)$, is $|t|$ for an arbitrarily chosen term t occurring in C. Clusters are partially ordered by their ranks; for two clusters C, C' we write $C < C'$ if $\mathrm{rk}(C) = t$, $\mathrm{rk}(C') = t'$ and $|t| < |t'|$ is logically implied by φ.

Example 2.19 (Mutually Independent Clusters). *Consider again the formula from Example 2.14,*

$$|y| = |\mathrm{cons}(x,z)| \ \wedge\ |x| = |z| \ \wedge\ |y| \neq |x| \ \wedge\ \mathrm{Is}_{\mathrm{cons}}(x,y,z) \ \wedge \bigwedge_{t,t' \in \mathcal{S}; t \neq t'} t \neq t' \ , \tag{2.51}$$

where $\mathcal{S} = \{x, y, z, \mathrm{cons}(x,z)\}$. *This formula induces two mutually independent* cons*-clusters,*

$$C_1 : \{\, x,\ z \,\} \ , \qquad\qquad C_2 : \{\, y,\ \mathrm{cons}(x,z) \,\}$$

with $\mathrm{rk}(C_1) < \mathrm{rk}(C_2)$.

A conjunction of literals Φ is *cluster complete* if all maximal clusters of Φ are mutually independent. A conjunction of literals Φ' is a *cluster completion* of Φ if $\Phi \subseteq \Phi'$ and Φ' is cluster complete. Φ' is *compatible* with Φ if satisfiability of Φ implies satisfiability of Φ. Like equality completion, we are only interested in compatible cluster completions.

Example 2.20 (Cluster Completeness). *The* $\mathrm{List}_{\mathbb{Z}}$ *formula*

$$x \neq y \ \wedge\ x \neq z \ \wedge\ x \neq w \ \wedge\ |x| = |y| \ \wedge\ |x| = |z| \ \wedge\ \mathrm{Is}_{\mathrm{cons}}(x,y,z,w) \tag{2.52}$$

gives three maximal cons*-clusters*

$$C_1 : \{\, x,\ y \,\} \ , \qquad\qquad C_2 : \{\, x,\ z \,\} \ , \qquad\qquad C_3 : \{\, w \,\} \ .$$

C_3 is closed, but neither C_1 nor C_2 is. If we conjoin (2.52) *with $y \neq z$, C_1 and C_2 are merged, making the resulting cluster $\{x,\ y,\ z\}$ maximal. If we conjoin* (2.52) *with $y = z$, then C_1 and C_2 become identical. Neither strengthening, however, results in a cluster completion of* (2.52)*, because C_3 is unaffected by either changes, and in both cases is still connected to other maximal clusters. The first case can be made cluster complete by conjoining it with $|w| \neq |x|$ or $|w| = |x| \wedge w \neq y \wedge w \neq z$. Similarly for the second case.*

As demonstrated in Example 2.19, equality completeness implies cluster completeness. The converse, however, does not hold: it is not necessary to have complete equality and type information on all terms to induce a set of mutually independent clusters. For example,

$$\bigwedge_{0<i\leq n} \Big(|x_i| = |y_i| \ \wedge\ x_i \neq y_i \ \wedge\ \mathrm{Is}_{\alpha}(x_i, y_i) \Big)$$

is not equality complete, but induces n maximal and mutually independent α-clusters, namely $\{\{x_i, y_i\} \mid 1 \leq i \leq n\}$. This in fact is what we want: a constraint weaker than an equality and type completion.

We also need to refine the notion of RLCC to deal with parameters.

Definition 2.18 (RLCC with Parameters in $\mathrm{TA}_{\mathbb{Z}}$). *Consider*

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \left[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \right] ,$$

where $\bar{\mathbf{y}}$ are parameters. Let $\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}})$ be the maximum subset of $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ not containing $\bar{\mathbf{x}}$ and $\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = \Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \setminus \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}})$. A formula $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is an RLCC in $\bar{\mathbf{x}}$ for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ relativized to $\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, (in short, $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$), if the following formulas are valid:

$$(\forall \bar{\mathbf{x}}, \bar{\mathbf{y}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \rightarrow \ (\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big(\Phi_{\Delta}(\bar{\mathbf{z}}, \bar{\mathbf{y}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] , \tag{2.53}$$

$$(\forall \bar{\mathbf{y}} : \mathbb{T})(\forall \bar{\mathbf{z}} : \mathbb{Z}) \Big[\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ \wedge \ \Phi_{\Delta}(\bar{\mathbf{z}}, \bar{\mathbf{y}}) \ \rightarrow \ (\exists \bar{\mathbf{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] . \tag{2.54}$$

In the viewpoint of logical equivalence, we can assume $\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}})$ is empty as it can be moved out of the scope of $(\exists \bar{\mathbf{x}})$. But we cannot make such an assumption with respect to the computation of RLCC, as its existence in general affects cluster completeness. For example,

$$(\exists x : \mathbb{T}) \left[x \neq y_1 \ \wedge \ x \neq y_2 \ \wedge \ \mathrm{Is}_{\alpha}(x, y_1, y_2) \ \wedge \ |x| = |y_1| \ \wedge \ |x| = |y_2| \right]$$

is not cluster complete without equality information between y_1 and y_2.

For the construction of an RLCC, we require that $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ be cluster complete and in *strongly solved form*.

Definition 2.19 (Strongly Solved Form). *We say that $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is* strongly solved *in $\bar{\mathbf{x}}$ if $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is solved in $\bar{\mathbf{x}}$ and all literals of the form $Ly \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, where $y \in \bar{\mathbf{y}}$ and $t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is a constructor term (properly) containing $\bar{\mathbf{x}}$, are redundant. We say that $(\exists \bar{\mathbf{x}} : \mathbb{T})[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})]$ is in* strongly solved form *if $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is strongly solved in $\bar{\mathbf{x}}$.*

Example 2.21 (Strongly Solved Form). *The formula $\Phi_{\mathrm{list}} \wedge \theta_{\mathbb{Z}}$*

$$\mathrm{Is}_{\mathrm{cons}}(y) \ \wedge \ y \neq \mathrm{cons}(x, z) \ \wedge \bigwedge_{t, t' \in \mathcal{S}; t \not\equiv t'} t \neq t' \ \wedge \ |y| = |\mathrm{cons}(x, z)| \ \wedge \ |x| = |z| \ \wedge \ |y| \neq |x| \ , \tag{2.55}$$

with $\mathcal{S} = \{x, y, z, \mathrm{cons}(x, z)\}$ is not *in strongly solved form. It can be made into strongly solved form, however, by adding $\mathrm{car}(y) \neq x$ or $\mathrm{cdr}(y) \neq z$ to Φ_{list}, or by changing $|\mathrm{cons}(x, z)| = |y|$ to $|\mathrm{cons}(x, z)| \neq |y|$ in $\theta_{\mathbb{Z}}$. Either one will make the literal $y \neq \mathrm{cons}(x, z)$ redundant.*

Recall that $\mathrm{CLS}^{\alpha}_{n}(x_0, x_1, \ldots, x_n)$ states that $x_0, \ldots, x_n$ is an α-cluster of $n + 1$ elements. We claim that Algorithm 2.6 computes an RLCC (with parameters $\bar{\mathbf{y}}$ being treated as ordinary variables).

Theorem 2.11 (RLCC with Parameters in $\mathrm{TA}_{\mathbb{Z}}$). *If $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is strongly solved in $\bar{\mathbf{x}}$ and cluster complete, then $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ computed by Algorithm 2.6 is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$.*

2.7.2 A Quantifier Elimination Procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$

In this section we expand Algorithm 2.11 to an elimination procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$, the first-order theory of term algebras with integers. Since $\mathscr{L}_{\mathbb{T}}^{\mathbb{Z}}$ has two sorts, namely $\mathbb{Z}$ and $\mathbb{T}$, we need to show elimination of integer quantifiers as well as term quantifiers.

Elimination of Quantifiers on Integer Variables

We assume that formulas with quantifiers on integer variables are in the form

$$(\exists \bar{\mathbf{z}} : \mathbb{Z}) \left[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \right] , \tag{2.56}$$

where $\bar{\mathbf{y}}$, $\bar{\mathbf{z}}$ are integer variables and $\bar{\mathbf{x}}$ are term variables. Since $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})$ is in $\mathscr{L}_{\mathbb{T}}$, we can move $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})$ out of the scope of $\exists \bar{\mathbf{z}}$, obtaining

$$\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \wedge \ (\exists \bar{\mathbf{z}} : \mathbb{Z})\, \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ . \tag{2.57}$$

For this reason, neither $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})$ nor $\Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ is required to be quantifier-free. Now we can put $(\exists \bar{\mathbf{z}} : \mathbb{Z})\, \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ into a quantifier-free form using Cooper's method [12, 46]. For the sake of efficiency, however, we can defer the actual elimination on $(\exists \bar{\mathbf{z}} : \mathbb{Z})[\varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})]$ until all term quantifiers have been eliminated. The reason, as we shall see soon, is that the elimination of term quantifiers does not require the integer constraint in (2.58) to be quantifier-free.

Elimination of Quantifiers on Term Variables

We assume that formulas with quantifiers on term variables are in the form

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \left[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \right] , \tag{2.58}$$

where $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ are term variables, $\bar{\mathbf{z}}$ are integer variables, $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is quantifier-free. The following algorithm is based on Algorithm 2.11.

Algorithm 2.14 (Normalization with Parameters $\mathrm{TA}_{\mathbb{Z}}$). *Input:* (2.58). *Apply the following sub-procedures successively.*

1. *Basic Normalization. Apply Algorithm 2.11 to $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ and update $\Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ accordingly.*

2. *Cluster Completion. Normalize* (2.58) *to a cluster completion (Algorithm 2.15).*

3. *Decomposition of Disequalities. Decompose disequalities of the Form $Ly \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ (Algorithm 2.16).*

The purpose of steps (1)-(3) is to transform (2.58) into a formula which is in strongly solved form and cluster complete, such that we can use Algorithm 2.6 to construct an RLCC. Having the RLCC allows us to reduce term quantifiers to integer quantifiers. Step (1) gives us a formula of the form (omitting integer literals)

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \Big[\bigwedge_i x_{p(i)} \neq t_i(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big] \ . \tag{2.44}$$

Algorithm 2.15 produces cluster completions of the input by merging mutually dependent maximal clusters.

Algorithm 2.15 (Cluster Completion). *Input:* (2.44). *Apply the following sub-procedures repeatedly until cluster completeness is obtained.*

1. *Select Clusters. Let*

$$C_1 = \{t_1, \ldots, t_n\} \ , \qquad\qquad C_2 = \{s_1, \ldots, s_m\}$$

   be two connected maximal clusters with witness either $t_i = s_j$ or $t_i \neq s_j$.

2. *Merge Clusters.*

   (a) *If $t_i = s_j$ occurs. Guess an equality completion for $C_1 \cup C_2$.*

   (b) *If $t_i \neq s_j$ occurs, but not $|t_i| \neq |s_j|$. Split on*

$$|t_i| = |s_j| \ \vee \ |t_i| \neq |s_j| \ .$$

   For the $|t_i| = |s_j|$ branch, guess an equality completion for $C_1 \cup C_2$.

3. *Renormalization. Apply Algorithm 2.12 to $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ and update $\Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ accordingly.*

The termination of Algorithm 2.15 can be seen as follows. Step (2) may increase the number of literals and hence the number of term occurrences, but it does not increase the number of equivalence classes on TA-terms. Step (3) (Algorithm 2.12) is called afterwards to restore the primitive form. Clearly, steps (1)-(2) of Algorithm 2.12 do not introduce new terms. Step (3) of Algorithm 2.12 solves equalities containing existentially quantified variables $\bar{\mathbf{x}}$ in terms of parameters $\bar{\mathbf{y}}$. It however will only introduce new terms into existing equivalence classes. In particular, it removes at least one variable in $\bar{\mathbf{x}}$ by substitution. So there can be at most n rounds of the run of step (2) where n is the number of distinct terms in (2.44). It is easily seen that the resulting formula is still in the form of (2.44) (omitting integer literals).

Algorithm 2.16 decomposes disequalities in $\Phi_{\mathbb{T}}$ that are of the form $Ly \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ such that $|Ly| = |t(\bar{\mathbf{x}}, \bar{\mathbf{y}})|$ is implied by $\Phi_{\mathbb{T}}$ and $t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is a constructor term (properly) containing $\bar{\mathbf{x}}$. The

decomposition consists of a sequence of disjunctive splittings, where in each step the matrix of (2.58) is updated accordingly. It is assumed that when Algorithm 2.16 is called, the matrix $\Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$ of (2.58) is cluster complete. The algorithm preserves this completeness: all resulting branches are cluster complete.

Algorithm 2.16 (Decomposition of Disequalities Containing $\bar{\mathbf{x}}$). *Input: $\mathcal{D}$, a set of disequalities of the form $Ly \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ such that $|Ly| = |t(\bar{\mathbf{x}}, \bar{\mathbf{y}})|$, in the context of* (2.58)*.*

Repeat until $\mathcal{D}$ is empty.

1. *Disequality Removal. Select $\mathcal{D}' \subseteq \mathcal{D}$ such that for some Ly*

$$\mathcal{D}' = \{\, Ly \neq \alpha(t_1^{(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_k^{(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \mid 0 < i \leq m \,\} \ .$$

   (a) *Disequality Splitting. Remove $\mathcal{D}'$ from $\mathcal{D}$ and add to $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$*

$$\bigwedge_{i=1}^{m} \Big(\mathsf{s}^{\alpha}_{p(i)} Ly \neq t^{(i)}_{p(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \bigwedge_{1 \leq j < p(i)} \mathsf{s}^{\alpha}_{j} Ly = t^{(i)}_{j}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big) \ , \tag{2.59}$$

   where p is an index function with domain $[1, m]$ and range $[1, k]$.

   (b) *Disequality Rewriting. Let $\delta \in [1, m]$, $I \subseteq [1, m]$ be such that*

$$p(\delta) = sup\,\{\, p(i) \mid 0 < i \leq m \,\} \ ,$$
$$I = \{\, i \in [1, m] \mid p(i) < p(\delta) \,\} \ .$$

   For any $i \in I$, replace

$$\mathsf{s}^{\alpha}_{p(i)} Ly \neq t^{(i)}_{p(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \qquad by \qquad t^{(\delta)}_{p(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \neq t^{(i)}_{p(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ .$$

2. *Cluster Completion. Call Algorithm 2.15 to restore cluster completeness.*

In step (1a) we rewrite $Ly \neq \alpha(t_1^{(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_k^{(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}))$ to

$$\bigvee_{0 < j \leq k} \Big(\mathsf{s}^{\alpha}_{j} Ly \neq t^{(i)}_{j}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \bigwedge_{0 < l < j} \mathsf{s}^{\alpha}_{l} Ly = t^{(i)}_{l}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big)$$

instead of more directly to

$$\bigvee_{1 \leq j \leq k} \Big(\mathsf{s}^{\alpha}_{j} Ly \neq t^{(i)}_{j}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big) \ .$$

Introduction of equalities helps limit the number of generated terms by consolidating most of them with existing ones. Thus, among newly generated terms, exactly one (namely $\mathsf{s}^{\alpha}_{p(\delta)} Ly$) is asserted

unequal to old terms that may contain $\bar{\mathbf{x}}$ thanks to the rewriting in step (1a) and an additional requirement on clusters explained below. Each run of step (1a) replaces a set of disequalities

$$\{\, Ly \neq \alpha(t_1^{(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_k^{(i)}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \mid i \in [1, m] \,\}$$

with a new set of disequalities

$$\{\, \mathsf{s}^{\alpha}_{p(\delta)} Ly \neq t^{(i)}_{p(\delta)}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \mid i \in I \,\} \ .$$

Clearly, $\mathsf{s}^{\alpha}_{p(\delta)} Ly$ must reside in a cluster of lower rank than that of Ly. As rank ordering is well-founded, the size of $\mathcal{D}$ will eventually decrease, and hence Algorithm 2.16 terminates. In addition, it can be shown that the total number of newly generated terms is bounded quadratically by the input size. As a result, we can obtain one exponential upper bound on the complexity of Algorithm 2.16.

Besides the rewriting in step (1a), we also need to prevent step (2) (Algorithm 2.15) from generating unwanted disequalities. This can be done by requiring, upon calling Algorithm 2.15, that newly generated terms are not made representatives for existing equivalence classes.

Example 2.22 (Removal of Disequalities). *To illustrate step* (1)*, assume* $\alpha = (\mathsf{s}^{\alpha}_1, \mathsf{s}^{\alpha}_2, \mathsf{s}^{\alpha}_3, \mathsf{s}^{\alpha}_4)$*, and*

$$\mathcal{D}' = \left\{ \begin{array}{ll} y \neq \alpha(u_1, u_2, u_3, u_4), & y \neq \alpha(v_1, v_2, v_3, v_4) \\ y \neq \alpha(w_1, w_2, w_3, w_4), & y \neq \alpha(t_1, t_2, t_3, t_4) \end{array} \right\} \ .$$

Let us consider below one of the disjuncts obtained by step (1a)*, written in matrix style with the conjunction connective omitted:*

$$\left(\begin{array}{llll} \mathsf{s}^{\alpha}_1 y = u_1 & \mathsf{s}^{\alpha}_2 y = u_2 & \mathsf{s}^{\alpha}_3 y \neq u_3 & \\ \mathsf{s}^{\alpha}_1 y = v_1 & \mathsf{s}^{\alpha}_2 y = v_2 & \mathsf{s}^{\alpha}_3 y = v_3 & \mathsf{s}^{\alpha}_4 y \neq v_4 \\ \mathsf{s}^{\alpha}_1 y = w_1 & \mathsf{s}^{\alpha}_2 y \neq w_2 & & \\ \mathsf{s}^{\alpha}_1 y \neq t_1 & & & \end{array} \right) \ .$$

It follows that $\delta = 2$ *and* $p(\delta) = 4$*. In step* (1b) *we use* v_1, v_2, v_3*, respectively, to replace* $\mathsf{s}^{\alpha}_1 y, \mathsf{s}^{\alpha}_2 y, \mathsf{s}^{\alpha}_3 y$ *that occur in newly generated disequalities, obtaining a new matrix of conjuncts*

$$\left(\begin{array}{llll} \mathsf{s}^{\alpha}_1 y = u_1 & \mathsf{s}^{\alpha}_2 y = u_2 & v_3 \neq u_3 & \\ \mathsf{s}^{\alpha}_1 y = v_1 & \mathsf{s}^{\alpha}_2 y = v_2 & \mathsf{s}^{\alpha}_3 y = v_3 & \mathsf{s}^{\alpha}_4 y \neq v_4 \\ \mathsf{s}^{\alpha}_1 y = w_1 & v_2 \neq w_2 & & \\ v_1 \neq t_1 & & & \end{array} \right) \ .$$

In this case only $\mathsf{s}^{\alpha}_4 y$ *(among the newly generated terms), having a lower rank than* y*, involves in a disequality*

that may need to be split further.

Proposition 2.4 (Strongly Solved Form). *Algorithm 2.14 produces a formula which is in strongly solved form and cluster complete.*

After normalization (Algorithm 2.14) and removal of redundant disequalities, the resulting formula is in the form

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \Big[\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ , \tag{2.60}$$

where $\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is of the form $\bigwedge_i x_{f(i)} \neq t_i(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ and $\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}})$ does not contain $\bar{\mathbf{x}}$. $\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is the integer constraint obtained from runs of Algorithm 2.15 (step (2) of Algorithm 2.14 and step (2) of Algorithm 2.16). The resulting formula is guaranteed to be in strongly solved form and cluster complete, and hence by Theorem 2.11, we can compute an RLCC $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ for $(\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}))/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, producing the equivalent

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \Big[\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ \wedge \ \Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ , \tag{2.61}$$

which reduces to

$$\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ \wedge \ (\exists \bar{\mathbf{u}} : \mathbb{Z}) \Big[\Phi_{\Delta}(\bar{\mathbf{u}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ , \tag{2.62}$$

because $\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ has been completely characterized by Φ_{Δ}, and all occurrences of $\bar{\mathbf{x}}$ in Φ_{Δ} are integer occurrences and hence integer variables.

Thus, the final algorithm for elimination of term quantifiers can be given as follows.

Algorithm 2.17 (Elimination of Term Quantifiers in $\mathrm{TA}_{\mathbb{Z}}$). *Input:* (2.58)

1. *Call Algorithm 2.14 to normalize the input, obtaining* (2.60)*.*

2. *Call Algorithm 2.6 to get the RLCC $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ and add it to the matrix of* (2.60)*, obtaining* (2.61)*.*

3. *Remove $\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ from the matrix of* (2.61)*, pull $\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ out of $(\exists \bar{\mathbf{x}} : \mathbb{T})$ and then reduce $(\exists \bar{\mathbf{x}} : \mathbb{T})$ to $(\exists \bar{\mathbf{z}} : \mathbb{Z})$, obtaining* (2.62)*.*

Theorem 2.12 (Soundness of Algorithm 2.17). *All transformations in Algorithm 2.17 keep equivalence.*

Theorem 2.13 (Complexity of Algorithm 2.17). *Algorithm 2.17 eliminates a block of quantifiers in time $2^{O(n^2 \lg n)}$.*

Example 2.23 (Quantifier Elimination in $\mathrm{TA}_{\mathbb{Z}}$). *Let us modify Example 2.18 by conjoining the matrix of* (2.46) *with an integer constraint $\varphi_{\mathbb{Z}}(x)$. (The concrete form of $\varphi_{\mathbb{Z}}(x)$ is irrelevant.) Now we have*

$$(\exists x : \mathrm{list}) \left[\mathrm{cons}(\mathrm{car}(x), y_1) = y_2 \ \wedge \ y_2 \neq x \ \wedge \ \varphi_{\mathbb{Z}}(x) \right] . \tag{2.63}$$

Running Algorithm 2.11 on (2.63) *and updating integer constraints accordingly, we arrive at*

$$\mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \mathrm{list}) \left[y_2 \neq \mathrm{cons}(\mathrm{car}(y_2), x_2) \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |x_2| + 1) \right] \tag{2.64}$$

(cf. (2.49)*). Below we will follow a single branch produced in the disjunctive splittings.*

We run Algorithm 2.15 to obtain a compatible cluster completion of (2.64)*, producing*

$$\begin{aligned} \mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \mathrm{list}) \Big[& y_2 \neq \mathrm{cons}(\mathrm{car}(y_2), x_2) \ \wedge \ |y_2| = |\mathrm{car}(y_2)| + |x_2| + 1 \\ & \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |x_2| + 1) \Big] . \end{aligned} \tag{2.65}$$

Here we omitted obvious disequalities implied by literals listed. The matrix of (2.65) *induces three mutually independent clusters:*

$$\{ x_2 \} , \qquad \{ \mathrm{car}(y_2) \} , \qquad \{ y_2, \ \mathrm{cons}(\mathrm{car}(y_2), x_2) \} .$$

The formula (2.65)*, however, is not in strongly solved form as x_2 appears in the disequality $y_2 \neq \mathrm{cons}(\mathrm{car}(y_2), x_2)$, and hence we need to run Algorithm 2.16. Choosing $\mathrm{cdr}(y_2) \neq x_2$ in step* (1a)*, we obtain*

$$\begin{aligned} \mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \mathrm{list}) \Big[& \mathrm{cdr}(y_2) \neq x_2 \ \wedge \ |y_2| = |\mathrm{car}(y_2)| + |x_2| + 1 \\ & \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |x_2| + 1) \Big] , \end{aligned} \tag{2.66}$$

which induces four clusters:

$$\{ \mathrm{car}(y_2) \} , \qquad \{ x_2 \} , \qquad \{ \mathrm{cdr}(y_2) \} , \qquad \{ y_2 \}$$

that are not mutually independent as neither $\{x_2\}$ nor $\{\mathrm{cdr}(y_2)\}$ is closed. Then step (2) *of Algorithm 2.16 (Algorithm 2.15) is called upon to fix the problem of mutual independence. Choosing $|\mathrm{cdr}(y_2)| = |x_2|$ we have*

$$\begin{aligned} \mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \mathrm{list}) \Big[& \mathrm{cdr}(y_2) \neq x_2 \ \wedge \ |y_2| = |\mathrm{car}(y_2)| + |x_2| + 1 \\ & \wedge \ |\mathrm{cdr}(y_2)| = |x_2| \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2) + |x_2| + 1) \Big] , \end{aligned} \tag{2.67}$$

which is in strongly solved form and induces three mutually independent clusters:

$$\{ \mathrm{car}(y_2) \} , \qquad \{ x_2, \ \mathrm{cdr}(y_2) \} , \qquad \{ y_2 \} .$$

Identifying (2.67) *with* (2.60)*, we have*

$$\begin{array}{lcl}
\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) & : & \Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ , \\
\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) & : & \mathrm{cdr}(y_2) \neq x_2 \ , \\
\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) & : & \mathrm{cdr}(y_2) = y_1 \ , \\
\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) & : & |y_2| = |\mathrm{car}(y_2)| + |x_2| + 1 \ \wedge \ |\mathrm{cdr}(y_2)| = |x_2| \ , \\
\Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) & : & \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |x_2| + 1) \ .
\end{array}$$

We can now call Algorithm 2.6 to get the RLCC $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ *for* $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$*, which simplifies to*

$$\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ |x_2| > 3 \ \wedge \ 2 \nmid |x_2| \ .$$

By step (2) *of Algorithm 2.17,* (2.67) *is equivalent to*

$$\begin{array}{l}
\mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \mathrm{list}) \Big[\ \mathrm{cdr}(y_2) \neq x_2 \ \wedge \ |y_2| = |\mathrm{car}(y_2)| + |x_2| + 1 \ \wedge \ |\mathrm{cdr}(y_2)| = |x_2| \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge \ |x_2| > 3 \ \wedge \ 2 \nmid |x_2| \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |x_2| + 1) \Big] \ ,
\end{array} \tag{2.68}$$

which, by the reduction of term quantifiers to integer quantifiers, transforms to

$$\begin{array}{l}
\mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists z : \mathbb{Z}) \Big[\ |y_2| = |\mathrm{car}(y_2)| + z + 1 \ \wedge \ |\mathrm{cdr}(y_2)| = z \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge \ z > 3 \ \wedge \ 2 \nmid z \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + z + 1) \Big] \ .
\end{array} \tag{2.69}$$

By substituting $|\mathrm{cdr}(y_2)|$ *for* z *we can remove* $(\exists z : \mathbb{Z})$*. Using the fact that* $|y_2| = |\mathrm{car}(y_2)| + |\mathrm{cdr}(y_2)| + 1$*, we finally obtain*

$$\mathrm{cdr}(y_2) = y_1 \ \wedge \ |\mathrm{cdr}(y_2)| > 3 \ \wedge \ 2 \nmid |\mathrm{cdr}(y_2)| \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |\mathrm{cdr}(y_2)| + 1) \ . \tag{2.70}$$

It is easy to verify that (2.70) *implies* (2.63)*. The reverse direction, however, does not hold because we only showed one branch of the reduction.*

2.7.3 Richer Theories on Constant Domain

Similar to Section 2.4.5, we can have quantifier elimination for $\mathrm{TA}^{+}_{\mathbb{Z}}$ provided $\mathfrak{A}_c$ also admits quantifier elimination. It is easily seen that no change is needed for the elimination of quantifiers on integer variables. For the elimination of term quantifiers, we assume formulas are of the form

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \Big[\ \Phi_c(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ , \tag{2.71}$$

where $\Phi_c(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is a formula in $\mathscr{L}_c$. Thanks to step (2) in Algorithm 2.11, we can assume that none of $\bar{\mathbf{x}}$ appears inside selectors. Without loss of generality, we also assume $\bar{\mathbf{x}}$ are either all asserted to

be non-nullary constructor terms or all asserted to be constants. In the former case, we can assume $\Phi_c(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is just $\Phi_c(\bar{\mathbf{y}})$ because $\bar{\mathbf{x}}$ do not properly appear inside selectors and hence they do not have any place in Φ_c. So we can simply move $\Phi_c(\bar{\mathbf{y}})$ out of $\exists\bar{\mathbf{x}}$, obtaining a quantified formula in the same form as (2.58). In the latter case, we can assume that $\bar{\mathbf{x}}$ do not properly occur inside constructor terms either, because we can decompose constructor terms properly containing $\bar{\mathbf{x}}$ in the same way as in Algorithms 2.12 and 2.16. So literals containing $\bar{\mathbf{x}}$ in $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ are equalities or disequalities on terms of constant type, and hence we can rewrite $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ as $\Phi^{(a)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi^{(b)}_{\mathbb{T}}(\bar{\mathbf{y}})$ with $\Phi^{(a)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ being a constraint in $\mathscr{L}_c$. We can then assume that $\Phi^{(a)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is a part of $\Phi_c(\bar{\mathbf{x}}, \bar{\mathbf{y}})$. We also simplify $\Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ to $\Phi'_{\mathbb{Z}}(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ by instantiating all $|x_i|$ to 1. Now it is clear that (2.71) is equivalent to

$$\Phi^{(b)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ \wedge \ \Phi'_{\mathbb{Z}}(\bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \wedge \ (\exists\bar{\mathbf{x}} : \mathbb{T})\, \Phi_c(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ , \tag{2.72}$$

where the quantifiers can be handled by the quantifier elimination procedure for $\mathrm{Th}(\mathfrak{U}_c)$.

2.7.4 Adaptation for Infinite Languages

In this section we describe adaptations to aforementioned algorithms needed to deal with theories in a (countably) infinite language. We require the number of distinct arities of the language be finite, which is not a real restriction for practically interesting theories. We distinguish three types of infinite languages according to the cardinalities of the constant domain and non-nullary constructor domain:

1. infinitely many constants, finitely many non-nullary constructors;
2. finitely many constants, infinitely many non-nullary constructors;
3. infinitely many constants, infinitely many non-nullary constructors.

A finite language is called type 0.

Below we discuss the three aspects of our algorithms that are affected by an infinite signature: type completion, counting constraints, and constant instantiation.

Type Completion

Type completion is affected only by the cardinality of the domain of non-nullary constructors, since for type completion the identity of constants is not important; only the fact that a term is a constant is relevant, which is provided by Is_{A}. Thus, no adaptation is necessary for languages of type 1. For languages of type 2 and 3, note that a given formula φ can only contain finitely many non-nullary constructors. For type completion we can collect all non-nullary constructors not occurring in φ into one pseudo-constructor type U, thus reducing a type 2 (resp. type 3) language a type 0 (resp. type 1) language. Below, we show how to define the counting constraints for the new constructor type U.

Counting Constraints

Both an infinite constant domain and an infinite number of non-nullary constructors allow counting constraints to be relaxed. For languages with an infinite constant domain (type 1 and 3), $\mathrm{CNT}^{\alpha}_{\omega,n}(x)$ (resp. $\mathrm{CNT}_{\omega,n}(x)$) is equivalent to $\mathrm{Tree}^{\alpha}(x)$ (resp. $\mathrm{Tree}(x)$), as there are infinitely many trees of any legitimate tree length.

For languages with only an infinite number of non-nullary constructors (type 2), the situation is slightly more complicated, because, depending on the arities of those non-nullary constructors, an infinite subset of those non-nullary constructors may or may not be useful in forming terms of a particular length. First consider the simplest case in which the language has an infinite number of non-nullary constructors of arity i only. Then the number of distinct terms of length x is infinite if and only if $x - i$ is a legitimate tree length (and thus x is also a legitimate tree length). If $x - i$ is not a legitimate tree length, the counting constraint reduces to that of a finite language. Thus in this case the counting constraint $\mathrm{CNT}_{k,n,i}(x)$, where the additional subscript i denotes the arity with an infinite number of non-nullary constructors, can be defined as

$$\mathrm{CNT}_{k,n,i}(x) : \ \mathrm{Tree}(x - i) \ \vee \ \mathrm{CNT}_{k,n}(x) \ .$$

In general there can be a finite number of arities for which the number of non-nullary constructors is infinite. Let I be the set of these arities. Then the counting constraint $\mathrm{CNT}_{k,n,I}(x)$ can be expressed as

$$\mathrm{CNT}_{k,n,I}(x) : \ \bigvee_{i \in I} \mathrm{Tree}(x - i) \ \vee \ \mathrm{CNT}_{k,n}(x) \ ,$$

and the counting constraint $\mathrm{CNT}^{\alpha}_{k,n,I}(x)$ can be expressed as

$$\mathrm{CNT}^{\alpha}_{k,n,I}(x) : \ \bigvee_{i \in I} \mathrm{Tree}^{\alpha}(x - i) \ \vee \ \mathrm{CNT}^{\alpha}_{k,n}(x) \ .$$

Note that $\mathrm{CNT}_{k,n}(x)$ and $\mathrm{CNT}^{\alpha}_{k,n}(x)$ above are defined with respect to a finite sub-language containing all constants and all non-nullary constructors not having arities in I. It does not matter if the sub-language contains any non-nullary constructors having arities in I, because the presence of those constructors does not affect the truth value of $\mathrm{CNT}_{k,n}(x)$ (resp. $\mathrm{CNT}^{\alpha}_{k,n}(x)$) when $\bigvee_{i \in I} \mathrm{Tree}(x - i)$ (resp. $\bigvee_{i \in I} \mathrm{Tree}^{\alpha}(x - i)$) is false. Let S denote the signature of such a sub-language. For the pseudo-constructor type U introduced above we have

$$\mathrm{CNT}^{U}_{k,n,I}(x) : \ \bigvee_{i \in I} \mathrm{Tree}(x - i) \ \vee \ \bigvee_{\alpha \in S} \mathrm{CNT}^{\alpha}_{k,n}(x) \ ,$$

since U must necessarily include all non-nullary constructors of arities in I.

Constant Instantiation

Having an infinite number of constants prohibits the application of the constant elimination (step (4) of Algorithm 2.11). In that step all variables $\bar{\mathbf{x}}$ that are asserted to be constants are nondeterministically instantiated with constants, thereby allowing the elimination of the corresponding $\exists\bar{\mathbf{x}}$. With an infinite number of constants, direct instantiation obviously is not possible. These variables, however, can still be eliminated as follows. At step (4) of Algorithm 2.11 all existentially quantified variables appear in disequalities only (cf. (2.35)). Let $\bar{\mathbf{x}}_c \subseteq \bar{\mathbf{x}}$ be the set of existentially quantified variables that are asserted to be constants. Then for any given assignment σ of the parameters $\bar{\mathbf{y}}$ and the remaining variables $\bar{\mathbf{x}}\backslash\bar{\mathbf{x}}_c$, all disequalities containing variables in $\bar{\mathbf{x}}_c$ can be simultaneously satisfied by assigning them distinct constants occurring neither in $[\![\bar{\mathbf{y}}]\!]\sigma$ nor in the formula. Therefore all variables in $\bar{\mathbf{x}}_c$ can be removed from the formula (2.44), as desired.

2.8 Conclusion

We presented decision procedures for term algebras augmented with Presburger arithmetic, for quantifier-free theories and quantified theories. Our technique is based on the extraction of exact integer constraints from term constraints, and in case of quantified theories, combined with the reduction of term quantifiers to integer quantifiers.

In Chapter 3 we extend our results to queues, a type of non-recursive data structure in which an object can be constructed in more than one way [63]. In Chapter 4 we extend our results to prove the decidability of the first-order theory of Knuth-Bendix order.

We also plan to extend this work to reason about the combination of data structures with integers in richer languages such as the theory of term algebras with subterm relation [57].

2.9 Proofs

In the following proofs we need to express legitimate lengths of trees. To support those expressions we use the following proposition, which is a well-known result following from the Euclidean algorithm for computing greatest common divisors.

Proposition 2.5 (Linear Combination). *Let $d_1, \ldots, d_n$ be positive integers and gcd their* greatest common divisor. *Then*

$$\textit{if} \quad \exists x_1, \ldots, x_n \geq 0 \left(x = \sum_{i=1}^{n} x_i d_i \right) \quad \textit{then} \quad gcd \mid x \ . \tag{2.73}$$

In addition, there exists $N_{\bar{d}}$ such that for any $x \geq N_{\bar{d}}$ the reverse also holds, that is,

$$\textit{if} \quad gcd \mid x \quad \textit{then} \quad \exists x_1, \ldots, x_n \geq 0 \left(x = \sum_{i=1}^{n} x_i d_i \right) \ . \tag{2.74}$$

For $1 \leq x < N_{\bar{d}}$, however, there is no closed formula to decide if x is a non-negative linear combination of $d_1, \ldots, d_n$. Finding the smallest $N_{\bar{d}}$ for $gcd = 1$ is known as the Frobenius Coin Problem, and has been shown to be NP-hard.

Below we present the proofs not included in the main text of this chapter. For ease of reference we restate the propositions and theorems.

Proposition 2.2 (LCC in $\mathrm{TA}^{\omega}_{\mathbb{Z}}$). Φ_Δ obtained by Algorithm 2.4 is expressible in a quantifier-free Presburger formula linear in the size of $\Phi_{\mathbb{T}}$.

Proof. Let n be the size of $\Phi_{\mathbb{T}}$. Then the size of $G_{\mathbb{T}}$ is $O(n)$. For each node in $G_{\mathbb{T}}$ we add (on average) at most four integer constraints. For an equivalence class $\{t_1, \ldots, t_n\}$, we only need to add $n - 1$ equalities, namely $|t_1| = |t_2| = \cdots = |t_n|$. So it suffices to show that the integer constraints $\mathrm{Tree}(|t|)$ and $\mathrm{Tree}^{\alpha}(|t|)$ can be expressed in quantifier-free Presburger formulas linear in the size of $\Phi_{\mathbb{T}}$.

Let $d_1, \ldots, d_n$ be the distinct arities of the non-nullary constructors in the given language. Since n and d_i ($1 \leq i \leq n$) are constant values (in the given language), the predicates $\mathrm{Tree}(x)$ and $\mathrm{Tree}^{\alpha}(x)$ are of constant length (with quantifiers). By Cooper's method, there exist equivalent quantifier-free formulas with length at most triple exponential in the length of $\mathrm{Tree}(x)$ (or $\mathrm{Tree}^{\alpha}(x)$) [44]. Of course, these equivalent quantifier-free formulas are still of constant length.

In fact $\mathrm{Tree}(x)$ can be expressed directly by a quantifier-free formula. Indeed, $\mathrm{Tree}(x)$ states that $x - 1$ is a non-negative linear combination of $d_1, \ldots, d_n$. By Proposition 2.5 this condition is equivalent to $gcd \mid x - 1$ provided $x - 1 \geq N_{\bar{d}}$ where gcd and $N_{\bar{d}}$ are as stated in the proposition. Therefore $\mathrm{Tree}(x)$ is equivalent to

$$\bigvee_{s \in S} (x - 1 = s) \ \vee \ \left(x - 1 \geq N_{\bar{d}} \ \wedge \ gcd \mid x - 1 \right) \ .$$

where $S \subseteq \{1, \ldots, N_{\bar{d}} - 1\}$ can be precomputed for the given language to contain exactly those values $1 \leq s < N_{\bar{d}}$ such that

$$\exists x_1, \ldots, x_n \geq 0 \left(s = \sum_{i=1}^{n} x_i d_i \right) . \qquad \square$$

Theorem 2.3 (LCC in $\mathrm{TA}^{\omega}_{\mathbb{Z}}$). Φ_Δ obtained by Algorithm 2.4 is an LCC for $\Phi_\mathbb{T}$.

Proof. To show that Φ_Δ computed by Algorithm 2.4 is an LCC for $\Phi_\mathbb{T}$, we need to show, by Definition 2.8, the validity of

$$(\forall \bar{\mathbf{x}} : \mathbb{T}) \left[\Phi_\mathbb{T}(\bar{\mathbf{x}}) \ \rightarrow \ (\exists \bar{\mathbf{z}} : \mathbb{Z}) \left(\Phi_\Delta(\bar{\mathbf{z}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \right) \right] , \tag{2.8}$$

$$(\forall \bar{\mathbf{z}} : \mathbb{Z}) \left[\Phi_\Delta(\bar{\mathbf{z}}) \ \rightarrow \ (\exists \bar{\mathbf{x}} : \mathbb{T}) \left(\Phi_\mathbb{T}(\bar{\mathbf{x}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \right) \right] . \tag{2.9}$$

Clearly, from the description of Algorithm 2.4, for any variable assignment σ satisfying $\Phi_\mathbb{T}$, $|\sigma|$ satisfies $\Phi_\mathbb{T}$ and thus (2.8) holds.

To demonstrate the validity of (2.9), let σ_Δ be a satisfying assignment of Φ_Δ. We have to show that there exists a variable assignment $\sigma_\mathbb{T}$ such that $|\sigma_\mathbb{T}| = \sigma_\Delta$ and $\sigma_\mathbb{T}$ satisfies $\Phi_\mathbb{T}$. Let σ be an arbitrary variable assignment such that $|\sigma| = \sigma_\Delta$. Clearly such an assignment exists; it may not, however, satisfy $\Phi_\mathbb{T}$. We show how σ can be transformed into an assignment $\sigma_\mathbb{T}$ that is guaranteed to satisfy $\Phi_\mathbb{T}$. Let $G_\mathbb{T}$ be the DAG of $\Phi_\mathbb{T}$ and $R\!\downarrow\!\uparrow$ be the bidirectional closure induced by $\Phi_\mathbb{T}$. Let $G'_\mathbb{T}$ be the *extension* of $G_\mathbb{T}$ that represents the variable assignment σ, that is $G'_\mathbb{T}$ is obtained from $G_\mathbb{T}$ by replacing each leaf labeled by a variable v by the ground tree representing $\sigma(v)$. Without loss of generality we assume that all leaf vertexes in $G_\mathbb{T}$ are labeled by either constants or variables; this can be achieved by variable abstraction of selector terms, as illustrated in Example 2.24 below.

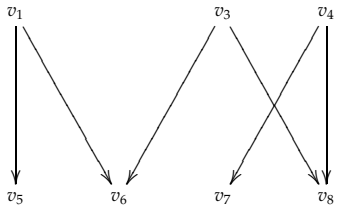
To obtain $\sigma_\mathbb{T}$ from σ, apply the following two steps:

1. substitute each variable asserted to be a constant in $\Phi_\mathbb{T}$ by a fresh constant. This is possible, since $\mathrm{TA}^{\omega}_{\mathbb{Z}}$ has infinitely many constants.

2. for each equivalence class $\{v_1, \ldots, v_k\}$ set $\sigma_\mathbb{T}(v_k) = \cdots = \sigma_\mathbb{T}(v_2) = \sigma(v_1)$.

We claim that $\sigma_\mathbb{T}$ is a satisfying assignment for $\Phi_\mathbb{T}$. Clearly $|\sigma_\mathbb{T}| = |\sigma| = \sigma_\Delta$ since the transformation does not affect the lengths of the terms. Moreover, $\sigma_\mathbb{T}$ respects $R\!\downarrow\!\uparrow$, that is, for any terms t and s, $(t, s) \in R\!\downarrow\!\uparrow$ implies $\sigma_\mathbb{T}(t) = \sigma_\mathbb{T}(s)$ (or $G'_t = G'_s$). It remains to show that for any two nonequivalent vertexes s and t in $G_\mathbb{T}$, $G'_s \neq G'_t$.

Let s and t be two vertexes such that $s \neq t$ in $\Phi_\mathbb{T}$, but suppose, for a proof by contradiction, that $G'_s = G'_t$. Let $\mathrm{ht}(s)$ denote the height of G'_s and similar for $\mathrm{ht}(t)$. Without loss of generality, assume that among all pairs of vertices t_1, t_2 such that $t_1 \neq t_2$ in $\Phi_\mathbb{T}$, but $G'_{t_1} = G'_{t_2}$, $h = \mathrm{ht}(t_1) = \mathrm{ht}(t_2)$ is minimal. If $h = 1$, then both s and t are variables in distinct equivalence classes, and thus they



$v_1 : x$ $\qquad$ $v_5 : \mathrm{car}(x)$
$v_2 : - - -$ $\qquad$ $v_6 : \mathrm{cdr}(x)$
$v_3 : \mathrm{cons}(\mathrm{cdr}(x), z)$ $\qquad$ $v_7 : y$
$v_4 : \mathrm{cons}(y, z)$ $\qquad$ $v_8 : z$

Figure 2-13: The DAG of $\mathrm{Is}_{\mathrm{cons}}(x) \ \wedge \ \mathrm{cons}(y,z) = \mathrm{cons}(\mathrm{cdr}(x), z) \ \wedge \ \mathrm{car}(x) \neq z$.

were assigned distinct constants in step (1), a contradiction. If $h > 1$, then $\sigma_{\mathbb{T}}(t) = \sigma_{\mathbb{T}}(s)$ implies that $\sigma_{\mathbb{T}}(t[1]) = \sigma_{\mathbb{T}}(s[1])$. But $\mathrm{ht}(t[1]) = \mathrm{ht}(s[1]) = h - 1 < h$, contradicting the minimality of h. □

Example 2.24 (Assignment in $\mathrm{TA}^{\omega}_{\mathbb{Z}}$). *To illustrate the variable abstraction and the construction of a satisfying variable assignment consider the following constraint*

$$\mathrm{Is}_{\mathrm{cons}}(x) \ \wedge \ \mathrm{cons}(y,z) = \mathrm{cons}(\mathrm{cdr}(x), z) \ \wedge \ \mathrm{car}(x) \neq z \ . \tag{2.75}$$

The DAG of (2.75)*, shown in Figure 2-13, is almost the same as that of Example 2.5 except that v_2 is not included here, and similar to that example, $R\downarrow\uparrow$ (without v_2) is*

$$\{\, (v_3, v_4),\ (v_6, v_7) \,\} \ .$$

To eliminate the selector terms labeling the leaf nodes v_5 and v_6, we introduce two new variables, x_1 and x_2, and let $x_1 = \mathrm{car}(x)$ and $x_2 = \mathrm{cdr}(x)$. Now x can be replaced by $\mathrm{cons}(x_1, x_2)$ and thus is not part of the variable assignment.

A satisfying integer assignment for Φ_Δ is

$$\sigma_\Delta : \{\, |x_1| := 1,\ |x_2| := 5,\ |y| := 5,\ |z| := 1 \,\} \ .$$

A corresponding term assignment σ such that $|\sigma| = \sigma_\Delta$ is

$$\sigma : \ \{\, x_1 := a,\ x_2 := \mathrm{cons}(\mathrm{cons}(a,a),a),\ y := \mathrm{cons}(a, \mathrm{cons}(a,a)),\ z := a \,\} \ .$$

The first step of the transformation produces

$$\{\, x_1 := a_1,\ x_2 := \mathrm{cons}(\mathrm{cons}(a_2, b_2), c_2),\ y := \mathrm{cons}(a_3, \mathrm{cons}(b_3, c_3)),\ z := a_4 \,\} \ .$$

Since x_2 and y are in the same equivalence class, the second step of the transformation produces

$$\sigma_{\mathbb{T}} : \{\, x_1 := a_1,\ x_2 := \mathrm{cons}(\mathrm{cons}(a_2, b_2), c_2),\ y := \mathrm{cons}(\mathrm{cons}(a_2, b_2), c_2),\ z := a_4 \,\} \ .$$

which satisfies (2.75).

Theorem 2.4 (Counting Constraints in $\mathrm{TA}_{\mathbb{Z}}$). $\mathrm{CNT}^{\alpha}_{k,n}(x)$ and $\mathrm{CNT}_{k,n}(x)$ are expressible by quantifier-free Presburger formulas that can be computed in time $O(n)$.

Proof. Let the language $\mathscr{L}_{\mathbb{T}}$ have m non-nullary constructors $\alpha_1, \ldots, \alpha_m$ with arities $d_1, \ldots, d_m$. Let $\mathscr{N}(x)$ denote the number of distinct term trees of length x. Recall that $\mathrm{CNT}_{k,n}(x)$ holds if and only if $\mathscr{N}(x) > n$ for a constant domain of size k. Thus it suffices to express $\mathscr{N}(x) > n$ as quantifier-free Presburger formula of size $O(n)$.

It is easily seen that $\mathscr{N}(0) = 0$ and $\mathscr{N}(1) = k$. For $x > 1$, $\mathscr{N}(x)$ can be expressed as a recurrence relation:

$$\mathscr{N}(x) \;=\; \sum_{i=1}^{m} \sum_{\{\langle x_1,\ldots,x_{d_i}\rangle \mid x_1+\cdots+x_{d_i}=x-1\}} \prod_{j=1}^{d_i} \mathscr{N}(x_j) \;. \tag{2.76}$$

The relation can be explained as follows: there are m ways to label the root of a tree; for a root with d children with lengths $x_1, \ldots, x_d$, respectively, there are $\prod_{j=1}^{d} \mathscr{N}(x_j)$ combinations. Using dynamic programming we can compute $\mathscr{N}(1), \mathscr{N}(2), \ldots$, until we reach the first $x_{\min}$ such that $\mathscr{N}(x_{\min}) > n$.

We first consider the special case that $d_1 = \cdots = d_m$, where we have for any $x_1, x_2 > 0$ the following monotonicity property:

$$\mathrm{Tree}(x_1) \;\wedge\; \mathrm{Tree}(x_2) \;\wedge\; x_1 > x_2 \quad\rightarrow\quad \mathscr{N}(x_1) > \mathscr{N}(x_2) \;. \tag{2.77}$$

The reason is that in this case term trees always "grow continuously", that is, the next larger tree is always obtained by expanding one of the vertexes of the previous tree.

For this special case $\mathscr{N}(x) > n$ reduces to $\mathrm{Tree}(x) \wedge x \geq x_{\min}$. To show that $\mathrm{Tree}(x) \wedge x \geq x_{\min}$ can be computed in $O(n)$ time, let d be the maximum arity. There are $O(x^{d-1})$ different sequences of positive numbers with sum $x - 1$, and thus $\mathscr{N}(x)$ can be obtained by $O(x^d)$ arithmetic operations. As $\mathscr{N}(x)$ grows exponentially in x, $x_{\min}$ is at the scale of $O(\lg n)$. Moreover, as all integers in the computation are less than n, any arithmetic operation costs time $O(\lg n)$. Therefore the search for such $x_{\min}$ can be done in $O(n)$ time.

Unfortunately, (2.77) does not hold in general when arities are different. For example, for $d_1 = 3$, $d_2 = 10$ and $k = 1$, $\mathscr{N}(10) = 12$, while $\mathscr{N}(11) = 1$. The problem is that in this case term trees do not necessarily grow continuously. Indeed, a tree of length 10 must consist of a root with three children, with lengths either 4, 4, and 1, or 7, 1, and 1. A tree of length 11, on the other hand, can only consist of a root with 10 children each of size 1. Consequently, a tree of length 10 cannot "grow" into a tree of length 11, and therefore $\mathscr{N}(10)$ and $\mathscr{N}(11)$ are completely unrelated.

However, there exists $N_{\bar{d}}$ such that for any $x \geq x_{\min} + N_{\bar{d}}$,

$$\mathscr{N}(x) > \mathscr{N}(x_{\min}) \qquad \text{iff} \qquad \text{Tree}(x) \ .$$

Let $N_{\bar{d}}$ be as in Proposition 2.5 and let *gcd* be the greatest common divisor of $d_1, \ldots, d_m$. The *"only if"* direction is trivial. For the *"if"* direction, suppose Tree(x) holds, that is, $x - 1$ can be expressed as a non-negative linear combination of $d_1, \ldots, d_m$. By Proposition 2.5, (2.73), $gcd \mid x - 1$. For the same reason, we have $gcd \mid x_{\min} - 1$, and since $x - x_{\min} = (x - 1) - (x_{\min} - 1)$ we have $gcd \mid x - x_{\min}$. By assumption, $x - x_{\min} \geq N_{\bar{d}}$, and therefore, by Proposition 2.5, (2.74), $x - x_{\min}$ can be expressed as a non-negative linear combination of $d_1, \ldots, d_m$, and thus a tree of length $x_{\min}$ can grow into a tree of length x by replacing one of its vertexes with a tree of size $x - x_{\min} + 1$. Therefore, $\mathscr{N}(x) > n$, and thus $\text{CNT}_{k,n}(x)$ can be expressed by

$$\bigvee_{s \in S_n} (x = s) \ \vee \ \Big(x \geq x_{\min} + N_{\bar{d}} \ \wedge \ \text{Tree}(x) \Big) \ ,$$

where

$$S_n = \{ x_{\min} \leq s < x_{\min} + N_{\bar{d}} \mid \mathscr{N}(s) > n \} \ .$$

Since $N_{\bar{d}}$ and the size of S_n are constant, $\text{CNT}_{k,n}(x)$ can be computed in $O(n)$ time as desired.

The proof for $\text{CNT}^{\alpha}_{k,n}(x)$ is similar. □

Proposition 2.3 (RLCC in $\text{TA}_{\mathbb{Z}}$). Φ_Δ obtained by Algorithm 2.6 is expressible in a quantifier-free Presburger formula of size linear in the size of $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$.

Proof. By Proposition 2.2, the call to Algorithm 2.4 to obtain Φ_Δ takes time $O(n)$ where n is the size of the input $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$. Next, for each α-cluster of size m, Algorithm 2.6 adds $\text{CNT}^{\alpha}_{k,m}(x)$ to Φ_Δ. By Theorem 2.4, $\text{CNT}^{\alpha}_{k,m}(x)$ can be computed in time $O(m)$. Since the sum of sizes of all clusters is $O(n)$, it follows that Φ_Δ can be computed in $O(n)$, and hence the size of Φ_Δ is $O(n)$. □

Theorem 2.5 (RLCC in $\text{TA}_{\mathbb{Z}}$). Φ_Δ obtained by Algorithm 2.6 is an RLCC for $\Phi_{\mathbb{T}}/\theta_{\mathbb{Z}}$.

Proof. Let $\Phi_{\mathbb{T}}(\bar{\mathbf{x}})$ be a type- and equality-complete term constraint and $\theta_{\mathbb{Z}}(|\bar{\mathbf{x}}|)$ be a Presburger formula. To show that Φ_Δ computed by Algorithm 2.6 is an RLCC for $\Phi_{\mathbb{T}}/\theta_{\mathbb{Z}}$, we need to show, by Definition 2.9, the validity of

$$(\forall \bar{\mathbf{x}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}) \ \rightarrow \ (\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big(\Phi_\Delta(\bar{\mathbf{z}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] \ , \tag{2.11}$$

$$(\forall \bar{\mathbf{z}} : \mathbb{Z}) \Big[\Phi_\Delta(\bar{\mathbf{z}}) \ \rightarrow \ (\exists \bar{\mathbf{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\mathbf{x}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{x}}) \ \wedge \ |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] \ . \tag{2.12}$$

For (2.11) consider an arbitrary variable assignment σ such that $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ is true. By Algorithm 2.6, Φ_Δ consists of $\Phi_{\Delta,4} \wedge \theta_{\mathbb{Z}} \wedge (\text{CNT}^{\alpha}_{k,n})_i$, where $\Phi_{\Delta,4}$ is the constraint computed by Algorithm 2.4 and

$(\mathrm{CNT}^{\alpha}_{k,n})_i$ are the counting constraints added in step (3). By Theorem 2.3, there exists an integer assignment σ_Δ such that $\sigma_\Delta = |\sigma_\mathbb{T}|$ such that $\Phi_{\Delta,4}$ is true, and obviously this assignment also makes $\theta_\mathbb{Z}$ true. Finally, the counting constraints impose a restriction on the length of terms. By Algorithm 2.6, for any term t such that $\mathrm{CLS}^{\alpha}_{n}(t, t_1, \ldots, t_n)$ is implied by $\Phi_\mathbb{T} \wedge \theta_\mathbb{Z}$, Φ_Δ includes the counting constraint $\mathrm{CNT}^{\alpha}_{k,n}(t, t_1, \ldots, t_n)$, or equivalently, $\mathscr{N}(|t|) > n$, with $\mathscr{N}(|t|)$ as before. Since σ satisfies $\Phi_\mathbb{T} \wedge \theta_\mathbb{Z}$, it must have assigned different terms to $t, t_1, \ldots, t_n$, and thus their length necessarily satisfies $\mathscr{N}(|t|) > n$.

For (2.12) consider an arbitrary integer assignment σ_Δ, assigning $\bar{\mathbf{z}} := \bar{\mathbf{d}}$, such that $\Phi_\Delta(\bar{\mathbf{d}})$ holds. We have to show that there exists a term assignment $\sigma_\mathbb{T}$, assigning $\bar{\mathbf{x}} := \bar{\mathbf{t}}$ such that $|\bar{\mathbf{t}}| = \bar{\mathbf{d}}$ and $\Phi_\mathbb{T}(\bar{\mathbf{t}})$ and $\theta_\mathbb{Z}(\bar{\mathbf{t}})$ hold. In contrast with the proof of Theorem 2.3, it is not immediately obvious that such an assignment exists, because we no longer assume an infinite constant domain. Therefore we incrementally construct $\sigma_\mathbb{T}$, starting with terms of smallest length.

Let $G_\mathbb{T}$ be the DAG and $R\downarrow\uparrow$ the bidirectional closure for $\Phi_\mathbb{T}$ as constructed in Algorithm 2.1. We assume, as in the proof of Theorem 2.3, that all selectors have been eliminated and thus leaf nodes of $G_\mathbb{T}$ are labeled by either constants or variables. Let $G'_\mathbb{T}$ be identical to $G_\mathbb{T}$ except that all nodes corresponding to equivalence classes in $R\downarrow\uparrow$ have been merged. Since $\Phi_\mathbb{T}$ is equality-complete any two distinct vertices in $G'_\mathbb{T}$ must correspond to distinct terms.

To construct $\sigma_\mathbb{T}$, we order all term lengths according to the values assigned by σ_Δ:

$$\underbrace{|t^{(1)}_0| = \cdots = |t^{(1)}_{n_1}|}_{\textit{block } 1} < \underbrace{|t^{(2)}_0| = \cdots = |t^{(2)}_{n_2}|}_{\textit{block } 2} < \cdots\cdots < \underbrace{|t^{(k)}_0| = \cdots = |t^{(k)}_{n_k}|}_{\textit{block } k}$$

Note that any term t in $\Phi_\mathbb{T}$ appears in Φ_Δ and hence $|t|$ is assigned a length by σ_Δ and appears in the above sequence. Let l_i be the length of the terms in block i. For simplicity, we assume all terms in block i are of type α_i. Starting with the terms in the first block, that is, $t^{(1)}_1, \ldots, t^{(1)}_{n_1}$, we incrementally construct a satisfying assignment $\sigma_\mathbb{T}$ for $\Phi_\mathbb{T}$ such that $|\sigma_\mathbb{T}| = \sigma_\Delta$.

Obviously $t^{(1)}_i$ ($i \leq n_1$) is either a constant or a term variable as its length is the smallest, and we only need to consider $t^{(1)}_i$ which are term variables. By Algorithm 2.7 we know that $\mathrm{CNT}^{\alpha_1}_{k,n_1}(l_1)$ is in Φ_Δ. As σ_Δ satisfies Φ_Δ, there are at least n_1+1 different α_1-terms of length l_1. Therefore we can simply assign each $t^{(1)}_i$ (if it is a variable) a distinct term. Now we proceed to the $(i+1)^{th}$ block assuming that the terms in the i^{th} block have been assigned. At this time the values of all non-variable terms in the $(i+1)^{th}$ block have been fixed because variables (if any) in those terms have length less than l_i and those variables have been assigned by the i^{th} round. By the same argument as before, due to the presence of $\mathrm{CNT}^{\alpha_{i+1}}_{k,n_{i+1}}(l_{i+1})$ in Φ_Δ, we are able to assign each variable in the $(i+1)^{th}$ block a different tree of length l_{i+1}. The assignment in each round will not create any equality between terms in $G'_\mathbb{T}$ simply because terms of the same type and the same length are assigned to different values. By induction we can eventually construct a satisfying assignment $\sigma_\mathbb{T}$ for $\Phi_\mathbb{T}$ such that $|\sigma_\mathbb{T}| = \sigma_\Delta$.

Since Φ_Δ includes $\theta_\mathbb{Z}$, $\theta_\mathbb{Z}(\bar{\mathbf{d}})$ holds, and therefore $\sigma_\mathbb{T}$ also satisfies $\theta_\mathbb{Z}$, as required. □

Theorem 2.6 (Counting Constraints in $\mathrm{RB}_{\mathbb{Z}}$). $\mathrm{CNT}^{\mathrm{rb}}_n(x_1, x_2)$, $\mathrm{CNT}^{r}_n(x_1, x_2)$ and $\mathrm{CNT}^{b}_n(x_1, x_2)$ are expressible by quantifier-free Presburger formulas that can be computed in $O(n)$.

Proof. The proof follows closely the one for Theorem 2.4. We first consider three counting functions:

$$\begin{array}{lcll} f(n_1,n_2) & = & |S| & \textit{where} \quad S = \{\, t \mid |t|_{\max} = n_1 \wedge |t|_{\min} = n_2 \,\}\ , \\ f^r(n_1,n_2) & = & |S| & \textit{where} \quad S = \{\, t \mid |t|_{\max} = n_1 \wedge |t|_{\min} = n_2 \wedge \mathrm{Is}_{\mathrm{red}}(t) \,\}\ , \\ f^b(n_1,n_2) & = & |S| & \textit{where} \quad S = \{\, t \mid |t|_{\max} = n_1 \wedge |t|_{\min} = n_2 \wedge \mathrm{Is}_{\mathrm{black}}(t) \,\}\ , \end{array}$$

where $f(n_1, n_2)$ $(n_1, n_2 > 0)$ gives the number of distinct trees whose maximal black path has length n_1 and minimal black path has length n_2. Similarly, $f^r(n_1, n_2)$ (resp. $f^b(n_1, n_2)$) gives the corresponding number for trees with red (resp. black) root.

These three functions are computable as they can be defined recursively as follows.

$$f(1,1) = 2\ , \tag{2.78}$$

$$f(n_1,n_2) = f^r(n_1,n_2) + f^b(n_1,n_2) \quad \textit{for } n_1 \neq 1 \vee n_2 \neq 1\ , \tag{2.79}$$

$$f^r(0,n_2) = f^r(n_1,0) = 0\ , \tag{2.80}$$

$$f^r(1,1) = 1\ , \tag{2.81}$$

$$f^r(n_1,1) = 2\sum_{i \leq n_1} f^b(n_1,i) \qquad \textit{for } n_1 > 1\ , \tag{2.82}$$

$$f^r(n_1,n_2) = \Big\{ \Big(\sum_{n_2 < j \leq i < n_1} f^b(i,j) \Big) \cdot f^b(n_1,n_2) \quad + \tag{2.83}$$

$$\sum_{n_2 \leq i < n_1} f^b(i,n_2) \cdot \sum_{n_2 \leq j \leq n_1} f^b(n_1,j) \quad + \tag{2.84}$$

$$\sum_{n_2 < j \leq n_1} f^b(n_1,j) \cdot \sum_{n_2 \leq i \leq n_1} f^b(i,n_2) \quad + \tag{2.85}$$

$$f^b(n_1,n_2) \cdot \sum_{n_2 \leq j \leq i \leq n_1} f^b(i,j) \quad \Big\} \qquad \textit{for } n_2 > 1\ , \tag{2.86}$$

$$f^b(0,n_2) = f^b(n_1,0) = 0\ , \tag{2.87}$$

$$f^b(n_1,1) = 0\ , \tag{2.88}$$

$$f^b(n_1+1,2) = 2\sum_{i \leq n_1} f^b(n_1,i) + 2\sum_{i \leq n_1} f^r(n_1,i)\ , \tag{2.89}$$

$$f^b(n_1+1,n_2+1) = \Big\{ \sum_{(\natural,\sharp) \in \{(r,b),(r,r),(b,b),(b,r)\}}$$

$$\Big[\Big(\sum_{n_2 < j \leq i < n_1} f^{\natural}(i,j) \Big) \cdot f^{\sharp}(n_1,n_2) \quad + \tag{2.90}$$

$$\sum_{n_2 \leq i < n_1} f^{\natural}(i,n_2) \cdot \sum_{n_2 \leq j \leq n_1} f^{\sharp}(n_1,j) \quad + \tag{2.91}$$

$$\sum_{n_2<j\le n_1} f^{\natural}(n_1,j)\cdot \sum_{n_2\le i\le n_1} f^{\sharp}(i,n_2) \quad + \tag{2.92}$$

$$f^{\natural}(n_1,n_2)\cdot \sum_{n_2\le j\le i\le n_1} f^{\sharp}(i,j)\ \Big]\ \Big\} \qquad \text{for } n_2>1\ . \tag{2.93}$$

Note that $f(n_1,n_2)$ is defined to be the sum of $f^r(n_1,n_2)$ and $f^b(n_1,n_2)$ except when $n_1=n_2=1$. The reason is that in this case nil counts as a tree having both a maximal black path and a minimal black path of length 1. The recurrence relation for $f^r(n_1,n_2)$ is defined by (2.80)-(2.86). Formulas (2.80)-(2.82) take care of irregular basic cases. In particular, $f^r(n_1,1)$ $(n_1>1)$ is so defined as a tree with a minimal black path of length 1 must have nil as one child and a tree with black root as the other child. For $n_2>1$, $f^r(n_1,n_2)$ is defined as a sum of four terms (2.83)-(2.86). Term (2.83) corresponds to the cases that the left child contains neither a maximal black path nor a minimal black path. Term (2.84) corresponds to the cases that the left child contains a minimal black path but no maximal black path. Term (2.85) corresponds to the cases that the left child contains a maximal black path but no minimal black path. Term (2.86) corresponds to the cases that the left child contains both a maximal black path and a minimal black path. The recurrence relation for $f^b(n_1+1,n_2+1)$ is defined by (2.87)-(2.93). Similar as before, formulas (2.87)-(2.89) take care of basic but irregular cases. For $n_2>1$, $f^b(n_1+1,n_2+1)$ is obtained in a similar way as $f^r(n_1,n_2)$ (when $n_2>1$) but taking into account that each aforementioned term splits into four terms because there are four ways to color the root of the left child and the root of the right child. As a result, $f^b(n_1+1,n_2+1)$ $(n_2>1)$ contains 16 terms.

In the following we show that $\mathrm{CNT}_n(x_1,x_2)$ is expressible in PA. The expressibility of $\mathrm{CNT}^r_n(x_1,x_2)$ and $\mathrm{CNT}^b_n(x_1,x_2)$ can be similarly obtained. Since $\mathrm{CNT}_n(x_1,x_2)$ define the set $U=\{(x,y)\in\mathbb{N}^2 \mid f(x,y)>n\}$, we can express $\mathrm{CNT}_n(x_1,x_2)$ in PA if and only if we can finitely represent U in PA. First we note that f is non-decreasing. If $f(x,y)>n$ then $f(x+1,y)>n$ because every tree with at least two paths and measure (x,y) can grow into a tree of measure $(x+1,y)$ by appending black(nil, nil) to a leaf node on a maximal black path. The exception happens to nil which only has one path. However, we still have $f(2,1)=f(1,1)=2>1$. Also if $f(x,y)>n$ and $y<x$, then $f(x,y+1)>n$ because every tree with measure (x,y) can grow into a tree of measure $(x,y+1)$ by appending black(nil, nil) to leaf nodes on all minimal black paths. It follows that if $f(x,y)>n$, $x'\ge x$, $y'\ge y$, and $x'\ge y'$ then we have $f(x',y')>n$. Next we exploit this property to finitely represent U in PA.

Let $D=\{(x,y)\in\mathbb{N}^2 \mid x\ge y\}$ be the domain of legitimate pairs; for the length of a maximal black path must be greater than or equal to the length of a minimal black path. Let $<_c$, $\le_c\in D^2$ be the *partial covering orderings* such that

$$\begin{aligned}(x,y)<_c(x',y') \qquad &\text{iff} \qquad x\le x'\ \wedge\ y\le y'\ \wedge\ x+y<x'+y'\ ,\\ (x,y)\le_c(x',y') \qquad &\text{iff} \qquad (x,y)=(x',y')\ \vee\ (x,y)<_c(x',y')\ .\end{aligned}$$

We say that (x,y) is *covered* (resp. *strictly covered*) by (x',y') if $(x,y)\le_c(x',y')$ (resp. $(x,y)<_c(x',y')$).

The non-decreasing property implies that if $f(x,y) > n$ and $(x,y) \leq_c (x',y')$, then $f(x',y') > n$. It follows that for a fixed n, U is upper closed under $\leq_c$, or equivalently U is a *filter* of $\langle D, \leq_c \rangle$. We can present U in PA if we can find a finite base set $B \subset D$ generating U in the sense that

(a) $\forall (x,y) \in B\, f(x,y) > n$, and

(b) $\forall (x,y) \in U\, \exists (x',y') \in B\, (x',y') \leq_c (x,y)$.

In other words, U is the upper closure of B in $\langle D, \leq_c \rangle$. Below we construct such B.

Let $(x,y) = (x',y')$ denote $x = x' \wedge y = y'$. Let $<_l, \leq_l : \in D^2$ be the *contra-variant lexicographical linear orderings* such that

$$\begin{array}{lll} (x_1,y_1) <_l (x_2,y_2) & \text{iff} & x_1 < x_2 \ \vee\ (x_1 = x_2 \ \wedge\ y_2 < y_1)\ , \\ (x_1,y_1) \leq_l (x_2,y_2) & \text{iff} & (x,y) = (x',y') \ \vee\ (x_1,y_1) <_l (x_2,y_2)\ . \end{array}$$

It follows from (2.87)-(2.86) that the computation of $f^r(x,y)$ relies on the values of $f^b(x',y')$ where $(x',y') \leq_l (x,y)$, and the computation of $f^b(x,y)$ relies on the values of $f^b(x',y')$ and of $f^r(x',y')$ where $(x',y') <_l (x,y)$. Therefore, we can use dynamic programming to compute f^b and f^r inductively with respect to $<_l$. So obtained is f. Eventually we will find the first pair $(x_{\min}, y_{\max})$ (with respect to $<_l$) such that $f(x_{\min}, y_{\max}) > n$. In a similar way, for each positive $i < y_{\max}$, we can find the smallest $x^{(i)}_{\min}$ such that $f(x^{(i)}_{\min}, i) > n$. We claim that B is

$$\{\, (x_{\min}, y_{\max}) \,\} \ \cup\ \{\, (x^{(i)}_{\min}, i) \ \mid\ 0 < i < y_{\max} \,\}\ .$$

It suffices to show that for any (x,y), if $f(x,y) > n$ then there exists $(x',y') \in B$ such that $(x',y') \leq_c (x,y)$. By the definition of $<_l$ and the fact that $(x_{\min}, y_{\max})$ is the first pair (with respect to $<_l$) such that $f(x_{\min}, y_{\max}) > n$, we have $x \geq x_{\min}$. If $y \geq y_{\max}$, then $(x_{\min}, y_{\max}) \leq_c (x,y)$. So without loss of generality assume $0 < y < y_{\max}$. Since $x^{(y)}_{\min}$ is the smallest number such that $f(x^{(y)}_{\min}, y) > n$, we have $x \geq x^{(y)}_{\min}$ and hence $(x^{(y)}_{\min}, y) \leq_c (x,y)$. So if $f(x,y) > n$ then (x,y) covers some pairs in B and hence B is a finite base for U.

Therefore we define $\mathrm{CNT}_n(x_1,x_2)$ as

$$x_1 \geq x_2 > 0 \ \wedge\ \Big(\bigvee_{(i,j)\in B} (x_1,x_2) \geq_c (i,j) \Big)\ .$$

Clearly, $f(x_1,x_2)$ grows exponentially in terms of x_1 or x_2. Hence the size of B is $O(n)$, so is the size of $\mathrm{CNT}_n(x_1,x_2)$. Similarly, we can obtain $\mathrm{CNT}^r_n(x_1,x_2)$ and $\mathrm{CNT}^b_n(x_1,x_2)$ of size $O(n)$. □

Theorem 2.7 (RLCC in $\mathrm{RB}_{\mathbb{Z}}$). Φ_Δ obtained by Algorithm 2.9 is an RLCC for $\Phi_{\mathrm{RB}}/\theta_{\mathbb{Z}}$ and is expressible in a quantifier-free Presburger formula of size linear in the size of $\Phi_{\mathrm{RB}} \wedge \theta_{\mathbb{Z}}$.

Proof. It follows immediately from Algorithm 2.9 that Φ_Δ is expressible in a quantifier-free Presburger formula of size linear in the size of $\Phi_{\mathrm{RB}} \wedge \theta_{\mathbb{Z}}$. We are left to show the validity of (2.21) and (2.22). The validity of (2.21) is obvious as every rule in Algorithm 2.9 is sound. We establish (2.22) by showing that given a satisfying assignment σ_Δ for $\Phi_\Delta(\bar{\mathbf{z}})$, there is a satisfying assignment σ_{RB} for $\Phi_{\mathrm{RB}}(\bar{\mathbf{x}})$ such that $|\sigma_{\mathrm{RB}}| = \sigma_\Delta$, that is, $|[\![t(\bar{\mathbf{x}})]\!]| = [\![z]\!]$ for each corresponding $t(\bar{\mathbf{x}})$ and $z \in \bar{\mathbf{z}}$. We assume that $\Phi_{\mathrm{RB}}(\bar{\mathbf{x}})$ is consistent; otherwise (2.22) is trivially true. Note that for simplicity we already assumed that $\mathscr{L}^{\mathbb{Z}}_{\mathrm{RB}}$ does not have selectors.

It is easily seen that the contra-variant lexicographical order $<_l$ has *subterm property* (with respect to $\|\cdot\|$), that is, for $t_1, t_2 \in \mathbb{T}_{rb}$, we have

$$\begin{array}{llllll} \|t_1\| & <_l & \|\mathrm{black}(t_1,t_2)\| & \|t_2\| & <_l & \|\mathrm{black}(t_1,t_2)\| \ , \quad \textit{if} \quad \|\mathrm{black}(t_1,t_2)\| > 0 \ ; \\ \|t_1\| & \leq_l & \|\mathrm{red}(t_1,t_2)\| & \|t_2\| & \leq_l & \|\mathrm{red}(t_1,t_2)\| \ , \quad \textit{if} \quad \|\mathrm{red}(t_1,t_2)\| > 0 \ . \end{array}$$

Let σ_Δ be a satisfying assignment of Φ_Δ. We order all integer terms according to the measure as induced by σ_Δ as follows.

$$\underbrace{\|t_0^{(1)}\| = \cdots = \|t_{n_1}^{(1)}\|}_{\textit{block } 1} <_l \underbrace{\|t_0^{(2)}\| = \cdots = \|t_{n_2}^{(2)}\|}_{\textit{block } 2} <_l \cdots\cdots <_l \underbrace{\|t_0^{(k)}\| = \cdots = \|t_{n_k}^{(k)}\|}_{\textit{block } k}$$

We assume that no variable is asserted to be nil as such an variable can be removed by instantiation. Let $\mathscr{M}_i$ denote the measure of terms in the i-th block. For each $i > 0$, the i-th block contains n_i terms in which there are $n_i^{(b)}$ black-terms and $n_i^{(r)}$ red-terms. In general, $n_i = n_i^{(b)} + n_i^{(r)}$ except for the block containing nil. Without loss of generality we assume that the first block contains terms of measure $(0,0)$; that is, terms violating red-black tree property (4) (Definition 2.12). Let us begin to build a partial assignment σ_{RB} from the second block which has measure $(1,1)$. Only two distinct trees, namely nil and red(nil, nil) have measure $(1,1)$, and hence $\mathrm{CNT}^r_n(1,1)(x)$ will be false for any $n > 0$. Since Φ_Δ is satisfiable and contains $\mathrm{CNT}^r_n(1,1)(x)$, we have $n = 0$, i.e., there is at most one red-term in this block. If it is a variable, then it can be assigned red(nil, nil). Let us assume we have partially assigned all terms up to the i-th block. For the $(i+1)$-th block, we first consider all black-terms, i.e., constructor terms of the form $\mathrm{black}(t_1, t_2)$ and variables of black-type. Since both $\|t_1\| <_l \|\mathrm{black}(t_1,t_2)\|$ and $\|t_2\| <_l \|\mathrm{black}(t_1,t_2)\|$, t_1 and t_2 must have been assigned. So has been $\mathrm{black}(t_1,t_2)$. Due to the presence of $\mathrm{CNT}^b_{n_{i+1}^{(b)}}(\mathscr{M}_{i+1})$ in Φ_Δ, we are able to assign each such variable a black-tree of measure $\mathscr{M}_{i+1}$. Now let us consider the rest red-terms. For all constructor term of the form $\mathrm{red}(t_1,t_2)$, we know that t_1 and t_2 must be black-terms or nil (because otherwise $\mathscr{M}_{i+1} = (0,0)$). As before we also have $\|t_1\| \leq_l \|\mathrm{red}(t_1,t_2)\|$ and $\|t_2\| \leq_l \|\mathrm{red}(t_1,t_2)\|$. So t_1 and t_2 can only appear in the first i blocks or in this block. In the former case, they have been assigned. In the latter case, they have been assigned too because they are black-trees. So unassigned terms are only variables of red-type. As $\mathrm{CNT}^r_{n_{i+1}^{(r)}}(\mathscr{M}_{i+1})$ is present in Φ_Δ, we are able to assign each variable a distinct red-trees

of measure $\mathscr{M}_{i+1}$. By induction we can build a partial assignment up to the k-th block.

Now let us go back to the first block which contains terms violating property (4). Terms appearing in this block are constructor terms that may be variables or may contain variables appearing in other blocks of higher order. Let us denote $\bar{\mathbf{y}}$ those variables appearing in other blocks and by $\bar{\mathbf{x}}$ the rest of variables. $\bar{\mathbf{y}}$ have been assigned values, which have measure greater than (with respect to $<_l$) $(0,0)$. We assign $\bar{\mathbf{x}}$ distinct trees having only red internal nodes and make sure that for any $x_1, x_2 \in \bar{\mathbf{x}}$, the difference between the heights (the longest paths) of $[\![x_1]\!]$ and $[\![x_2]\!]$ is greater than the number of terms appearing so far, including those in the assignment. Since there are infinitely many trees violating property 4, this is obviously feasible. We now finished the construction of σ_{RB}.

It is clear that σ_{RB} satisfies all disequalities between terms from the 2-nd to k-th blocks. Let us consider disequalities of the form $x \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ where $x \in \bar{\mathbf{x}}$ is in the first block. $t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ can not be some $y \in \bar{\mathbf{y}}$ since it will have measure $(0,0)$, contradicting the fact it appears in other blocks. For the same reason we will not have disequalities $y \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ for $y \in \bar{\mathbf{y}}$. Now if $t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is some $x' \in \bar{\mathbf{x}}$, then obviously $[\![x]\!] \neq [\![x']\!]$ as they are assigned distinct trees. If $t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is a proper constructor terms containing some of $\bar{\mathbf{x}}$, then we still have $[\![x]\!] \neq [\![t(\bar{\mathbf{x}}, \bar{\mathbf{y}})]\!]$ as by the choice of values of $\bar{\mathbf{x}}$, $[\![x]\!]$ and $[\![t(\bar{\mathbf{x}}, \bar{\mathbf{y}})]\!]$ are trees of different height. All in all, σ_{RB} respects all disequalities in the first block, and hence σ_{RB} is a satisfying assignment such that $|\sigma_{\mathrm{RB}}| = \sigma_\Delta$. □

Theorem 2.9 (Soundness of Algorithm 2.13). All transformations in Algorithm 2.13 keep equivalence.

Proof. Recall that Algorithm 2.13 first calls Algorithm 2.11 to transform a formula into one in solved form, that is in the form

$$(\exists \bar{\mathbf{x}}) \Big[\bigwedge_i x_{p(i)} \neq t_i(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big] \ \wedge \ \bigwedge_k F_k y_{f(k)} \neq u_k(\bar{\mathbf{y}}) \ \wedge \ \bigwedge_l G_l y_{g(l)} = H_l y_{h(l)} \ . \quad (2.43)$$

It then eliminates the quantifiers by removing all literals containing $\bar{\mathbf{x}}$. It is straightforward to show that all transformations in Algorithm 2.11 preserve equivalence, and we omit the proof. It remains to show that (2.43) is equivalent to

$$\bigwedge_k F_k y_{f(k)} \neq u_k(\bar{\mathbf{y}}) \ \wedge \ \bigwedge_l G_l y_{g(l)} = H_l y_{h(l)} \ ,$$

that is, that

$$(\forall \bar{\mathbf{y}} : \mathbb{T})(\exists \bar{\mathbf{x}} : \mathbb{T}) \Big[\bigwedge_i x_{p(i)} \neq t_i(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \Big] \quad (2.44)$$

is valid, or that

$$\bigwedge_i x_{p(i)} \neq t_i(\bar{\mathbf{x}}, \bar{\mathbf{b}}) \ \wedge \ \bigwedge_j L_j b_{q(j)} \neq s_j(\bar{\mathbf{x}}, \bar{\mathbf{b}}) \tag{2.94}$$

is satisfiable for an arbitrary sequence $\bar{\mathbf{b}}$ of fixed TA-terms. For simplicity, we assume that all nonnullary constructors have the same arity. The general case can be proved via a minor modification.

Let $\bar{\mathbf{x}}$ be $x_1, \ldots, x_n$. Let ξ be the maximal length of terms not containing $\bar{\mathbf{x}}$ in (2.94) and let $\rho > \xi$ be such that there exist at least n distinct nonconstant terms $\bar{\mathbf{d}} = d_1, \ldots, d_n$ of length ρ. We claim that $\bar{\mathbf{d}}$ satisfies (2.94). Observe that, due to step (4) of Algorithm 2.11, none of the x_i are constrained to be constants, and thus all variables can be assigned nonconstant terms.

To show that $L_j b_{q(j)} \neq s_j(\bar{\mathbf{d}}, \bar{\mathbf{b}})$ holds, we assume that the constructor term s_j actually contains $\bar{\mathbf{x}}$. If not, the literal $L_j y_{q(j)} \neq s_j(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ can be moved out of the scope of $\exists \bar{\mathbf{x}}$. Since s_j is a constructor term, $|s_j(\bar{\mathbf{d}}, \bar{\mathbf{b}})| \geq \rho > \xi$. On the other hand, L_j is a selector sequence and thus $|L_j b_{q(j)}| \leq \xi$. Therefore, for all j, $L_j b_{q(j)} \neq s_j(\bar{\mathbf{d}}, \bar{\mathbf{b}})$.

To show that $d_{p(i)} \neq t_i(\bar{\mathbf{d}}, \bar{\mathbf{b}})$, consider the following four cases depending on the structure of t_i.

1. t_i does not contain any variable in $\bar{\mathbf{x}}$. Then $|t_i(\bar{\mathbf{b}})| \leq \xi$ while $|d_{p(i)}| = \rho > \xi$, and hence $d_{p(i)} \neq t_i(\bar{\mathbf{b}})$.

2. t_i properly contains a variable in $\bar{\mathbf{x}}$. Then $|d_{p(i)}| = \rho$ but $|t_i(\bar{\mathbf{d}}, \bar{\mathbf{b}})| > \rho$, and hence $d_{p(i)} \neq t_i(\bar{\mathbf{b}})$.

3. t_i is a constant c_i. Then $d_{p(i)} \neq c_i$ holds since none of terms in $\bar{\mathbf{d}}$ is a constant.

4. t_i is a variable in $\bar{\mathbf{x}}$. Then $t_i \equiv x_{p(i')}$ where $p(i') \neq p(i)$ (or $x_{p(i)} \neq x_{p(i)}$ simplifies to false). Since $p(i) \neq p(i')$, $d_{p(i)} \neq d_{p(i')}$ holds by the selection of $\bar{\mathbf{d}}$.

Theorem 2.10 (Complexity of Algorithm 2.13). Algorithm 2.13 removes a block of quantifiers in time $2^{O(n)}$.

Proof. First note that we need not be concerned with the increase of the matrix size by the substitution, since we can represent a conjunction of literals efficiently using the DAG representation (Definition 2.4), in which substitution can simply be done by rearranging edges in the graph. For example, consider the following sequence of formulas

$$x_1 = \alpha(x_2, x_2) \ , \qquad x_2 = \alpha(x_3, x_3) \ , \qquad \ldots \ , \qquad x_n = \alpha(x_{m+1}, x_{m+1}) \ .$$

Instead of generating a formula of size $2^{O(m)}$, the substitution only gives a linear "double-edged" path from x_1 to x_{m+1}. For details see [45].

It suffices to analyze each step of Algorithm 2.11. Let n be the size of the matrix. Step (1) (type completion) generates at most $2^{O(n)}$ disjuncts, as for a formula containing n selector terms, there are at most $2^{O(n)}$ combinations of tester literals. Step (2) (selector elimination) can be done in $O(n)$ as a

selector term can be transformed to a formula in the constructor language in linear size. Step (3) (decomposition, Algorithm 2.12) generates at most $2^{O(n)}$ disjuncts. In addition, it can increase the matrix size to $O(n^2)$, due to solving $\bar{\mathbf{x}}$ in terms of $\bar{\mathbf{y}}$ (step (3) of Algorithm 2.12); the total lengths of all paths in a tree of size n is bound by n^2. Note that the size increase comes from the increase of term occurrences and the number of distinct terms is still in $O(n)$. Step (4) (constant elimination) produces at most $2^{O(n)}$ disjuncts. All put together, steps (1)-(4) generate $2^{O(n)}$ disjuncts each of which has size $O(n^2)$ and contains $O(n)$ distinct terms. □

Theorem 2.11 (RLCC with Parameters in $\mathrm{TA}_{\mathbb{Z}}$**).** If $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is strongly solved in $\bar{\mathbf{x}}$ and cluster complete, then $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ computed by Algorithm 2.6 is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$.

Proof. To show that Φ_{Δ} computed by Algorithm 2.6 is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, we need to show, by Definition 2.18, the validity of

$$(\forall \bar{\mathbf{x}}, \bar{\mathbf{y}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \rightarrow (\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big(\Phi_{\Delta}(\bar{\mathbf{z}}, \bar{\mathbf{y}}) \wedge |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] , \tag{2.53}$$

$$(\forall \bar{\mathbf{y}} : \mathbb{T})(\forall \bar{\mathbf{z}} : \mathbb{Z}) \Big[\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \wedge \Phi_{\Delta}(\bar{\mathbf{z}}, \bar{\mathbf{y}}) \rightarrow (\exists \bar{\mathbf{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge |\bar{\mathbf{x}}| = \bar{\mathbf{z}} \Big) \Big] . \tag{2.54}$$

The validity of (2.53) can be shown by a similar argument as was given for (2.11) in the proof of Theorem 2.5.

To prove (2.54) consider arbitrary assignments σ_{Δ} and $\sigma^{(y)}_{\mathbb{T}}$, assigning $\bar{\mathbf{z}} := \bar{\mathbf{p}}$ and $\bar{\mathbf{y}} := \bar{\mathbf{s}}$ respectively, such that $\Phi_{\Delta}(\bar{\mathbf{p}}, \bar{\mathbf{s}})$ and $\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{s}})$ hold. We have to show that there exists a term assignment $\sigma^{(x)}_{\mathbb{T}}$, assigning $\bar{\mathbf{x}} := \bar{\mathbf{t}}$ such that $|\bar{\mathbf{t}}| = |\bar{\mathbf{p}}|$ and $\Phi_{\mathbb{T}}(\bar{\mathbf{t}}, \bar{\mathbf{s}})$ and $\theta_{\mathbb{Z}}(\bar{\mathbf{t}}, \bar{\mathbf{s}})$ hold. As in the proof of Theorem 2.5, we incrementally construct $\sigma^{(x)}_{\mathbb{T}}$, starting with terms of smallest length.

To construct $\sigma^{(x)}_{\mathbb{T}}$, we order all term lengths according to the values assigned by σ_{Δ}:

$$\underbrace{|u^{(1)}_0| = \cdots = |u^{(1)}_{n_1}|}_{\textit{block } 1} < \underbrace{|u^{(2)}_0| = \cdots = |u^{(2)}_{n_2}|}_{\textit{block } 2} < \cdots\cdots < \underbrace{|u^{(j)}_0| = \cdots = |u^{(j)}_{n_j}|}_{\textit{block } j}$$

Let l_i be the length of terms in the i^{th} block. For any TA-term t occurring in $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, $|\bar{\mathbf{t}}|$ appears in $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ and hence in the above sequence. In general a block can contain more than one cluster for each type. By Proposition 2.4, however, maximal clusters are mutually independent, and therefore, without loss of generality, we can assume each block i consists of only one maximal α_i-cluster.

Beginning with terms in the first block, namely $u^{(1)}_1, \ldots, u^{(1)}_{n_1}$, we incrementally construct a satisfying assignment for $\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{s}})$. We only need to consider $u^{(j)}_{n_j}$ which contains $\bar{\mathbf{x}}$. Obviously $u^{(1)}_i$ $(i \leq n_1)$ is either a constant or a variable in $\bar{\mathbf{x}}$ as its length is the smallest. By Algorithm 2.6 we know that $\mathrm{CNT}^{\alpha_1}_{k,n_1}(l_1)$ is in $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$. As $\bar{\mathbf{p}}$ satisfies $\Phi_{\Delta}(\bar{\mathbf{x}}, \bar{\mathbf{s}})$, there are at least $n_1 + 1$ different terms of length l_1. Therefore we can simply assign each $u^{(1)}_i$ (if it is a variable) a distinct α_1-term.

Now suppose that all variables in the i^{th} block have been assigned. Consider the $(i+1)^{th}$ block. At this time values of all non-variable terms in the $(i + 1)^{th}$ block have been determined, because $\bar{\mathbf{x}}$

only appear inside constructor terms, and hence have been assigned values by the i^{th} round. For example, suppose that $t(x)$ is a constructor term in the $(i{+}1)^{th}$ block. Since $|x| < |t(x)|$, x was assigned by the i^{th} round and so is the value of $t(x)$. Due to the presence of $\mathrm{CNT}^{\alpha_{i+1}}_{k,n_{i+1}}(l_{i+1})$ in $\Phi_\Delta(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, we are able to assign each variable in the $(i+1)^{th}$ block a different term of length l_{i+1}. The variable assignment in each round does not create any equality between terms in a block, i. e., it does not violate any disequalities.

Up to now the proof is essentially the same as the one for Theorem 2.5. The only problematic case is that a cluster may have selector terms containing $\bar{\mathbf{y}}$ as well as constructor terms containing $\bar{\mathbf{x}}$. For example, suppose that a cluster C in the i^{th} block contains both Ly and $\alpha_i(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_k(\bar{\mathbf{x}}, \bar{\mathbf{y}}))$ for $\mathrm{ar}(\alpha_i) = k$. $[\![Ly]\!]$ is fixed by $\sigma^{(y)}_{\mathbb{T}}$ and $[\![t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}})]\!], \ldots, [\![t_k(\bar{\mathbf{x}}, \bar{\mathbf{y}})]\!]$ have been determined before the i^{th} round and hence $[\![\alpha_i(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_k(\bar{\mathbf{x}}, \bar{\mathbf{y}}))]\!]$ is determined too. By Definition 2.19, however, the disequality $Ly \neq t(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is redundant, which can only be the case if for some j $(1 \leq j \leq k)$, $\mathsf{s}^{\alpha_i}_j Ly \neq t_j(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is implied by $\Phi_{\mathbb{T}}$. Since $|t_j(\bar{\mathbf{x}}, \bar{\mathbf{y}})| < |\alpha_i(t_j(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_k(\bar{\mathbf{x}}, \bar{\mathbf{y}}))|$, $t_j(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ has been assigned before the i^{th} round such that $[\![\mathsf{s}^{\alpha_i}_j Ly]\!] \neq [\![t_j(\bar{\mathbf{x}}, \bar{\mathbf{y}})]\!]$. As a consequence we have $[\![Ly]\!] \neq [\![\alpha_i(t_1(\bar{\mathbf{x}}, \bar{\mathbf{y}}), \ldots, t_k(\bar{\mathbf{x}}, \bar{\mathbf{y}}))]\!]$.

Since at each step we can build a satisfying partial assignment for $\bar{\mathbf{x}}$, by induction, we can eventually construct a satisfying assignment $\bar{\mathbf{t}}$ such that $\Phi_{\mathbb{T}}(\bar{\mathbf{t}}, \bar{\mathbf{s}})$ and $|\bar{\mathbf{t}}| = \bar{\mathbf{p}}$. It is clear that $\theta_{\mathbb{Z}}(\bar{\mathbf{t}}, \bar{\mathbf{s}})$ also holds as Φ_Δ implies $\theta_{\mathbb{Z}}$ (thanks to step (2) of Algorithm 2.6). □

Proposition 2.4 (Strongly Solved Form). Algorithm 2.14 produces a formula which is in strongly solved form and cluster complete.

Proof. The production of strong solved form is guaranteed by Algorithm 2.16. It suffices to show that Algorithm 2.16 preserves cluster completeness. Step (1a) of Algorithm 2.16 may generate new equalities as well as new disequality literals of the form $Ly \neq t'(\bar{\mathbf{x}}, \bar{\mathbf{y}})$, which in general destroys the mutual independence of clusters. Step (2) (Algorithm 2.15), however, is called after each run of step (1a) to restore cluster completeness. The termination argument has been given in the description of the algorithm. □

Theorem 2.12 (Soundness of Algorithm 2.17). All transformations in Algorithm 2.17 keep equivalence.

Proof. Clearly, Algorithm 2.14 preserves equivalence. So, it suffices to show the equivalence between (2.60) and (2.62), copied below,

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \Big[\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ , \tag{2.60}$$

$$\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \ \wedge \ (\exists \bar{\mathbf{u}} : \mathbb{Z}) \Big[\Phi_\Delta(\bar{\mathbf{u}}, \bar{\mathbf{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ . \tag{2.62}$$

By Proposition 2.4 $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is in strongly solved form and cluster complete, and hence by Theorem 2.11, $\Phi_\Delta(\bar{\mathbf{x}}, \bar{\mathbf{y}})$ is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$. By Proposition 2.1, $\Phi_\Delta(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ is an RLCC for $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}})/\bar{\mathbf{x}}/(\theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}))$.

To show that (2.60) implies (2.62), assume that for all $\bar{\mathbf{y}}$ and for all $\bar{\mathbf{z}}$ there exists $\bar{\mathbf{x}}$ such that $\Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ holds. Then by Definition 2.18, (2.53), there exists $\bar{\mathbf{u}} = |\bar{\mathbf{x}}|$ such that $\Phi_{\Delta}(\bar{\mathbf{u}}, \bar{\mathbf{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ holds, and hence (2.62) holds.

To show that (2.62) implies (2.60), assume that for all $\bar{\mathbf{y}}$ and for all $\bar{\mathbf{z}}$ there exists $\bar{\mathbf{u}}$ such that $\Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}}) \wedge \Phi_{\Delta}(\bar{\mathbf{u}}, \bar{\mathbf{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ holds. Then by Definition 2.18, (2.54), there exists $\bar{\mathbf{x}}$ such that $|\bar{\mathbf{x}}| = \bar{\mathbf{u}}$ and $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ holds, and hence, noting that $\Phi_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \equiv \Phi^{(1)}_{\mathbb{T}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \wedge \Phi^{(2)}_{\mathbb{T}}(\bar{\mathbf{y}})$, (2.60) holds, as required. □

Theorem 2.13 (Complexity of Algorithm 2.17). Algorithm 2.17 eliminates a block of quantifiers in time $2^{O(n^2 \lg n)}$.

Proof. We only need to analyze each step of Algorithm 2.14. As shown in the proof of Theorem 2.10, step (1) (basic normalization, Algorithm 2.11) generates $2^{O(n)}$ disjuncts each of which contains $O(n)$ distinct terms.

Step (2) (cluster completion, Algorithm 2.15) takes $2^{O(n \lg n)}$. It follows from the fact that an equality completion corresponds to a valid product of a partition of terms on syntactic equality and a partition of terms on length equality. For a set of size n the number of distinct partitions is called the Bell number, denoted by $B(n)$. An asymptotical expression for $B(n)$ is $\frac{1}{\sqrt{n}}\rho(n)^{n+\frac{1}{2}}e^{\rho(n)-n-1}$, where $\rho(n)$ is implicitly defined by $\rho(n)\lg\rho(n) = n$ [33]. Obviously $B(n)$ is bounded by $2^{O(n \lg n)}$, and so is the number of cluster completions.

Step (3) (Algorithm 2.16) is a bit more costly. At the beginning of Algorithm 2.16, the matrix of (2.44) induces at most n clusters containing variables in $\bar{\mathbf{x}}$. Let us assume that each rank r is assigned a number $n(r)$ in $[1, n]$ such that if $r_1 < r_2$, then $n(r_1) < n(r_2)$. The sum of all rank numbers is bounded by n^2. Each run of step (1)-(2) will reduce the sum at least by 1, and generates at most one new term appearing in disequalities. The total number of distinct terms after Algorithm 2.16 is bounded by $O(n^2)$. The number of cluster completions is therefore bounded by $2^{O(n^2 \lg n^2)} = 2^{O(n^2 \lg n)}$. This calculation has included the cost of Algorithm 2.15 and Algorithm 2.12, both of which could be called $O(n^2)$ times in the run of Algorithm 2.16. □

Chapter 3

Queues with Integers

01:38:44:460

> Tank : Okay, so what do you need? Besides a miracle.
> Neo : Guns. Lots of guns.
>
> —*The Matrix*.

Queues provide an important synchronization mechanism in modeling distributed protocols; they are the basis of many concurrent algorithms. In this chapter we extend the theory of queues with a length function that maps a queue to its size, resulting in a combined theory of queues and Presburger arithmetic. In order to handle the special properties of queue structure, we adapt the reduction technique presented in the previous chapter with new normalization procedures. We present decision procedures for quantifier-free theories and a quantifier elimination procedure for the first-order theory that can remove a block of existential quantifiers in one step.

3.1 Introduction.

Queues are widely used in many fields of computer science such as communication networks, job scheduling and simulation. They also provide an important synchronization mechanism in modeling distributed protocols and hence form the basis of many concurrent algorithms. In this chapter we extend the theory of queues with a length function which maps a queue to its size. The extended language has three sorts; atoms ($\mathcal{A}$), integers ($\mathbb{Z}$), and queues ($\mathcal{Q}$). As in the case of term algebras, the language is the set-theoretic union of the language of queues and the language of Presburger arithmetic. Formulas are formed from *atom*, *queue*, and *integer literals* using logical connectives and quantifications. The two theories are connected by the length function $|\cdot| : \mathcal{Q} \to \mathbb{N}$. It allows us to express semantics of string operations in the C language. For example, `strncmp` can be expressed in the existential theory of queues with prefix relation and integer arithmetic as follows.

$$
\begin{array}{ll}
 & \text{strncmp(const char } *s_1\text{, const char } *s_2\text{, size_t n)} : \\
 & res = 0 \ \wedge \ \exists q\,(\,|q| = n \ \wedge \ q \preceq s_1 \ \wedge \ q \preceq s_2\,) \\
\vee & res > 0 \ \wedge \ \Big[\,(s_1 \neq s_2 \ \wedge \ s_2 \preceq s_1 \ \wedge \ |s_2| \leq n) \ \vee \ \exists q \Big(\bigvee_{c<c'} (q \circ c' \preceq s_1 \ \wedge \ q \circ c \preceq s_2 \ \wedge \ |q| < n) \Big) \Big] \\
\vee & res < 0 \ \wedge \ \Big[\,(s_1 \neq s_2 \ \wedge \ s_1 \preceq s_2 \ \wedge \ |s_1| \leq n) \ \vee \ \exists q \Big(\bigvee_{c<c'} (q \circ c' \preceq s_2 \ \wedge \ q \circ c \preceq s_1 \ \wedge \ |q| < n) \Big) \Big] \ .
\end{array}
$$

With the full expressive power of Presburger arithmetic, we are able to express linear relations between sizes of queues. For example, in a queuing network with n input queues denoted by $q_i : (i < n)$, the property that the influx is bounded by B can be expressed as

$$\sum_{i=0}^{n-1} |q_i| < B \ .$$

We present decision procedures for quantifier-free theories of queues with integer arithmetic. We consider two kinds of quantifier-free theories, based on whether they include prefix relation or not. We also present a quantifier elimination procedure for the first-order theory of queues with integers. The elimination procedure removes a block of existential quantifiers in one step. In all developments, we assume that the atom domain is finite; the decision problems in an infinite domain are considerably easier.

Related Work and Comparison. Bjørner gave a decision procedure for the quantifier-free theory of queues with subsequence relations which consist of prefix, suffix and sub-queue relations [5]. Bjørner also discussed the integer combination for the case of infinite atom domain without the subsequence relations. The quantifier elimination and the complexity of the first-order theory of

queues were given by [48] and [50], respectively. By a standard encoding (in which a queue is represented as sets of natural numbers), the first order theory of queues with prefix relation can be interpreted by WS1S, and hence it is decidable. This theory also admits quantifier elimination [4]. Thomas studied theories of words with "equal length" predicate which can be viewed as special integer constraints [52].

Recently Klaedtke and Ruess showed the decidability of a fragment of $WS1S$ with cardinality constraints ($WS1S^{\mathsf{card}}$) and the undecidability of $WS1S^{\mathsf{card}}$ for certain fragments with second-order quantifier alternation [24]. By the standard encoding, the first-order theory of queues with prefix relation and integers can be interpreted in $WS1S^{\mathsf{card}}$. In particular, the quantifier-free fragment can be interpreted in the decidable fragment of $WS1S^{\mathsf{card}}$ which does not contain alternation of second-order quantifiers on variables occurring in cardinality constraints. Though interpretation in general renders elegant decidability results, it produces less efficient decision procedures in practice, especially if the host theory has high complexity (in this case even the existential $WS1S$ is non-elementary). Moreover, it is unlikely that any interpretation can put the full first-order theory of queues with integer arithmetic into a decidable fragment of $WS1S^{\mathsf{card}}$.

In Chapter 2 we gave decision procedures for the theory of term algebras with integer constraints. The method relies on a key normalization process to extract integer constraints from term constraints. The normalization partitions terms into stratified clusters such that

1. each cluster consists of pairwise unequal terms (trees) of the same length, and

2. disequalities between composite terms (proper trees) in a cluster are implied by disequalities in the clusters of lower ranks.

Property (2) allows the construction of a satisfying assignment in a bottom-up fashion, while providing integer constraints that precisely characterize the satisfiability of the clusters. Thus, (1) and (2) allow us to reduce the satisfiability of the original formula to the satisfiability of computable integer constraints. The decision procedures presented in this chapter rely on the same idea. But for queues, disequalities cannot be normalized into stratified clusters, because queues are not uniquely generated (they can grow at both ends). Consider, for example, the constraint

$$X \neq Y \ \wedge \ aX \neq Yb \ \wedge \ Xa \neq bY \ \wedge \ |X| = |Y| \ .$$

Clearly infinitely many assignments of the form $\{X = (ba)^n b, Y = a(ba)^n\}$ satisfy $X \neq Y$, but neither $aX \neq Yb$ nor $Xa \neq bY$. As a consequence, we cannot construct a satisfying assignment inductively. In this chapter we present new normalization procedures that allow the computation of a *cut length* L_t for all queue variables: below L_t all satisfying assignments can be enumerated, above L_t integer constraints can be computed that are equisatisfiable with the original formula.

Chapter Organization. Section 3.2 defines the language and structure of queues and presents some word properties. Section 3.3 describes a decision procedure for the quantifier-free theory of queues [5], the basis for our decision procedures. Section 3.4.1 introduces the theory of queues augmented with Presburger arithmetic and presents the technical machinery for the decision procedures. Section 3.4 presents the main contribution of this chapter: it adapts the technique presented in Section 2.4.2 to derive a decision procedure for the extended theory of queues. Section 3.5 refines the results in Section 3.4 for the extended theory of queues with prefix relation and integer arithmetic. Section 3.6 applies the technique to give a quantifier elimination procedure for the extended first-order theory of queues (without the prefix relation). Section 3.7 concludes with some ideas for future work. Most proofs are given in Section 3.8.

3.2 The Theory of Queues

Definition 3.1 (Queues). *The structure of queues* $\mathfrak{Q} : \langle \mathcal{Q}; \mathcal{A}, C, \mathcal{S} \rangle$ *consists of*

1. $\mathcal{A}$*: A finite set of atoms: a, b, c, . . . We use* $\epsilon_{\mathcal{A}}$ *to denote the "phantom atom" whose only purpose is to keep functions on queues total.*

2. $\mathcal{Q}$*: The domain of queues, which consists of sequences of atoms. We use* $\epsilon_{\mathcal{Q}}$ *to denote the empty queue.*

3. C*: Two constructors: the* left insertion $\mathrm{la} : \mathcal{A} \times \mathcal{Q} \to \mathcal{Q}$ *and the* right insertion $\mathrm{ra} : \mathcal{A} \times \mathcal{Q} \to \mathcal{Q}$ *such that for* $\alpha \in \mathcal{Q}$*,*

$$\mathrm{la}(\epsilon_{\mathcal{A}}, \alpha) = \alpha \ , \qquad \mathrm{ra}(\epsilon_{\mathcal{A}}, \alpha) = \alpha \ ,$$

and for $a \in \mathcal{A} \setminus \{\epsilon_{\mathcal{A}}\}$, $\langle s_1, \ldots, s_n \rangle \in \mathcal{Q}$*,*

$$\mathrm{la}(a, \epsilon_{\mathcal{Q}}) = \langle a \rangle \ , \qquad \mathrm{ra}(a, \epsilon_{\mathcal{Q}}) = \langle a \rangle \ ,$$
$$\mathrm{la}(a, \langle s_1, \ldots, s_n \rangle) = \langle a \ , s_1, \ldots, s_n \rangle \ , \qquad \mathrm{ra}(a, \langle s_1, \ldots, s_n \rangle) = \langle s_1, \ldots, s_n, a \rangle \ .$$

4. $\mathcal{S}$*: Four selectors: the* left head $\mathrm{lh} : \mathcal{Q} \to \mathcal{A}$*, the* left tail $\mathrm{lt} : \mathcal{Q} \to \mathcal{Q}$*, the* right head $\mathrm{rh} : \mathcal{Q} \to \mathcal{A}$*, and the* right tail $\mathrm{rt} : \mathcal{Q} \to \mathcal{Q}$ *such that*

$$\mathrm{lh}(\epsilon_{\mathcal{Q}}) = \epsilon_{\mathcal{A}} \ , \qquad \mathrm{lt}(\epsilon_{\mathcal{Q}}) = \epsilon_{\mathcal{Q}} \ ,$$
$$\mathrm{rh}(\epsilon_{\mathcal{Q}}) = \epsilon_{\mathcal{A}} \ , \qquad \mathrm{rt}(\epsilon_{\mathcal{Q}}) = \epsilon_{\mathcal{Q}} \ ,$$
$$\mathrm{lh}(\langle s_1 \rangle) = s_1 \ , \qquad \mathrm{lt}(\langle s_1 \rangle) = \epsilon_{\mathcal{Q}} \ ,$$
$$\mathrm{rh}(\langle s_1 \rangle) = s_1 \ , \qquad \mathrm{rt}(\langle s_1 \rangle) = \epsilon_{\mathcal{Q}} \ ,$$

and for $\langle s_1, \ldots, s_n \rangle \in \mathcal{Q}$ $(n > 1)$,

$$\mathrm{lh}(\langle s_1, \ldots, s_n \rangle) = s_1 \ , \qquad \mathrm{lt}(\langle s_1, \ldots, s_n \rangle) = \langle s_2, \ldots, s_n \rangle \ ,$$
$$\mathrm{rh}(\langle s_1, \ldots, s_n \rangle) = s_n \ , \qquad \mathrm{rt}(\langle s_1, \ldots, s_n \rangle) = \langle s_1, \ldots, s_{n-1} \rangle \ .$$

We use $\mathscr{L}_{\mathcal{Q}}$ *to denote the language of queues and* $\mathscr{L}_{\mathcal{A}}$ *the equational sub-language of* $\mathscr{L}_{\mathcal{Q}}$ *on* $\mathcal{A}$.

Queues are finite *words* constructed from *letters* in $\mathcal{A}$, i. e., $\mathcal{Q} = \mathcal{A}^*$. We assume $|\mathcal{A}| > 1$ as queue constraints trivially reduce to integer constraints if $\mathcal{A}$ is a singleton. Because of finiteness of $\mathcal{A}$, we assume only constant atoms appear in formulas (i. e., no occurrences of atom variables). For clarity, $X, Y, Z, \ldots$ are reserved for queue variables, $a, b, c, \ldots$ for constant atoms and $\alpha, \beta, \gamma, \ldots$ for constant queues. We use concatenation $\circ$ to express constructor operations. For example, $a \circ X \circ b$ stands for either $\mathrm{ra}(b, \mathrm{la}(a, X))$ or $\mathrm{la}(a, \mathrm{ra}(b, X))$. Often we even omit $\circ$ unless necessary for clarity.

The expressive power of the constructor language (the language without selectors) is the same as that of $\mathscr{L}_{\mathcal{Q}}$.

Proposition 3.1 (Elimination of Selectors). *For any* φ *in* $\mathscr{L}_{\mathcal{Q}}$*, one can effectively compute an equivalent* φ' *such that*

1. φ' *contains no selectors, and*
2. *if* φ *is quantifier-free, then* φ' *can be put into either* $\exists_1$ *or* $\forall_1$ *form.*

So in terms of satisfiability or validity, even in the quantifier-free fragment of $\mathscr{L}_{\mathcal{Q}}$, selectors are dispensable without compromising expressiveness. From now on we assume $\mathscr{L}_{\mathcal{Q}}$ is the constructor language except in Section 3.6 where selectors are used in quantifier elimination. In a constructor language, a queue variable can occur at most once in a term, and hence we can assume all terms of sort $\mathcal{Q}$ are in the form $\alpha X \beta$, where α, β are constant words and X is a queue variable.

We use "word", "letter" in semantic discussions and use "queue", "atom" to refer to their respective counterparts in the formal language. For a word α, $|\alpha|$ denotes the length of α; $\alpha[i]$ $(0 < i \leq |\alpha|)$ denotes the letter at position i; $\alpha[m..n]$ $(0 < m, n \leq |\alpha|)$ denotes the consecutive fragment from position m to position n; α^m denotes the word obtained by concatenating m copies of α. For $k < |\alpha|$, we define

$$\begin{aligned}
\alpha^* &= \{\alpha^m \mid m \geq 0\} \ , \\
\alpha^+ &= \{\alpha^m \mid m > 0\} \ , \\
\mathrm{ext}(\alpha, m, k) &= \alpha^m \alpha[1..k] \ , \\
\mathrm{orb}(\alpha, k) &= \bigcup_{m \geq 0} \mathrm{ext}(\alpha, m, k) \ , \\
\mathrm{orb}(\alpha) &= \bigcup_{k \geq 0} \mathrm{orb}(\alpha, k) \ .
\end{aligned}$$

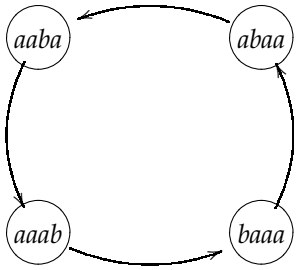


Figure 3-1: *aaba*, *abaa*, *aaab* and *baaa* are pairwise conjugate.

Here orb(α), called *orbit* of α, is the set of all words of the form $\alpha^*\alpha[1..i]$ ($i < |\alpha|$) and orb(α, k) is the subset of orb(α) consisting of words ending with $\alpha[1..k]$.

Example 3.1 (Orbits). *Let $\beta = aba$. Then* ext($\beta, 1, 2$) = *abaab,* orb(β) *are words in one of the following forms* $(aba)^*, (aba)^*a, (aba)^*ab$, *which are* orb($\beta, 0$), orb($\beta, 1$) *and* orb($\beta, 2$).

The equations in $\mathscr{L}_Q$ can express certain "circular properties" on words.

Definition 3.2 (Conjugacy [32]). *Two words α, β are* conjugate *if there exist words u, v ($v \neq \epsilon_Q$) such that $\alpha = uv$ and $\beta = vu$. In other words, α is obtained from β by circular shift, and vice versa. We say that α is k-*conjugate *with β if $|u| = k$.*

Example 3.2 (Conjugacy). *Figure 3-1 shows four words, aaba, abaa, aaab and baaa, which are pairwise conjugate.*

Proposition 3.2 (Conjugacy [32]). *Two words α and β are conjugate if and only if there exists γ such that $\alpha\gamma = \gamma\beta$. Moreover, α and β are k-conjugate if and only if for all γ, $\alpha\gamma = \gamma\beta$ if and only if $\gamma \in$ orb(α, k).*

This proposition says that if $\alpha = u_1u_2$, $\beta = u_2u_1$, then the solution set of $\alpha X = X\beta$ is $(u_1u_2)^*u_1$. As a consequence, we define $X \in$ orb(α, k) as "syntactic sugar" for $\alpha X = X\alpha[k+1..|\alpha|]\alpha[1..k]$; similarly $X \notin$ orb(α, k) for $\alpha X \neq X\alpha[k+1..|\alpha|]\alpha[1..k]$.

Definition 3.3 (Primitive Words). *A word β is* primitive *if $\beta \neq \alpha^n$ ($n \geq 1$) for any proper prefix α of β, and is* strongly primitive *if in addition $\beta \notin$ orb(α).*

Example 3.3 (Primitive Words). *Consider $\alpha \equiv aba$, $\beta \equiv abab$ and $\gamma \equiv abb$. It is clear that β is non-primitive, α is primitive but not strongly primitive and γ is strongly primitive.*

If β is non-primitive, then there exists α such that $\beta \in \alpha^*$. We call the shortest such α the *generator* of β, denoted by gen(β). It is easily seen that orb(β) = orb(gen(β)), i. e., every orbit is uniquely generated. Thus, without loss of generality, we always assume the occurrences of β in orb(β, k) to be primitive.

Proposition 3.3 (Exclusiveness of Orbit Literals [5, 48]). *Let α, β be two distinct primitive words and γ a word of length n. Then $\gamma \in \text{orb}(\alpha) \cap \text{orb}(\beta)$ implies $n < |\alpha| + |\beta| - 1$.*

This proposition says that $X \in \text{orb}(\alpha)$ and $X \in \text{orb}(\beta)$ (where $\alpha \not\equiv \beta$), are *mutually exclusive* except for a finite number of cases which can be enumerated by comparing two orbits of α and β coordinate-wise up to $|\alpha| + |\beta| - 2$. We have

Theorem 3.1 (Exclusiveness of Orbit Literals [5, 48]). *A conjunction of literals of the form*

$$\bigwedge_{0<i\leq n} X \in \text{orb}(\alpha_i) \ \wedge \ \bigwedge_{0<j\leq m} X \notin \text{orb}(\beta_j) \tag{3.1}$$

can be simplified to a formula in which at most one of $X \in \text{orb}(\alpha_i)$ appears, and if this happens, no $X \notin \text{orb}(\beta_j)$ occurs. In addition, if $n > 1$, (3.1) *simplifies to either* false *or a finite set of solutions.*

Example 3.4 (Exclusiveness of Orbit Literals). *$X \in \text{orb}(ab) \wedge X \in \text{orb}(aba)$ simplifies to $X \in \{a, b, aba\}$, and $X \in \text{orb}(ab) \wedge X \notin \text{orb}(aba)$ simplifies to $X \in \text{orb}(ab) \wedge X \notin \{a, b, aba\}$.*

3.3 Decision Procedure for $\text{Th}^{\forall}(\mathfrak{Q})$

The basis of the decision procedures for the combined theory is the decision procedure for the quantifier-free theory of queues, $\text{Th}^{\forall}(\mathfrak{Q})$ [5].

The key component of all decision procedures presented in this chapter is normalization, which includes elimination of certain forms of equalities, disequalities and prefix relations (when present). The common constituent of all normalization procedures is *equality elimination*.

Definition 3.4 (Solved Form in $\mathfrak{Q}$). *A set of equalities $\mathcal{E}$ is in* solved form *if every $E \in \mathcal{E}$ has the form $x = t(\bar{\mathbf{x}})$ where x neither occurs in $\bar{\mathbf{x}}$ nor in any other equations in $\mathcal{E}$.*

A set of solved equalities can be obtained by substitution; when a solution $X = t$ is obtained, other occurrences of X are replaced by t. Such substitution can detect certain inconsistencies, e.g., when $X = \alpha X \beta$ is generated with $\alpha \not\equiv \epsilon_Q$ or $\beta \not\equiv \epsilon_Q$. Obviously a set of equalities in solved form pose no restriction on the solution, and hence those equalities can be considered "virtually eliminated".

Definition 3.5 (Normal Form in $\mathfrak{Q}$). *A queue constraint Φ_Q is in* normal form *if*

1. *all equalities are in solved form,*
2. *for each queue variable X there exists at most one literal $X \in \text{orb}(\alpha, k)$, and*
3. *disequalities are in the form $X \neq \alpha$, or $\alpha X \neq Y\beta$ for $X \not\equiv Y$.*

The following algorithm, a simplified version of [5], reduces a set of equalities and inequalities to normal form.

Algorithm 3.1 (Normalization in $\mathfrak{Q}$, cf. [5]). *Input $\Phi_Q : \mathcal{E} \cup \mathcal{D}$ where $\mathcal{E}$, $\mathcal{D}$ are sets of equalities and disequalities, respectively.*

1. *Reduce literals of the forms*

$$\alpha X\beta = \beta' \ , \qquad \alpha X\beta \neq \beta' \ , \qquad \alpha X\beta = \alpha' Y\beta' \ , \qquad \alpha X\beta \neq \alpha' Y\beta' \ ,$$

where α, β, α', β' are constant queues and X, Y are queue variables, to

$$X = \beta \ , \qquad X \neq \beta \ , \qquad \alpha X = Y\beta \ , \qquad \alpha X \neq Y\beta \ .$$

by position-wise removing prefixes and suffixes. For example, $abXcd = abcYdd$ reduces to false *and $abXcd \neq abcYd$ to $Xc \neq cY$.*

2. *Eliminate equalities of the form $\alpha X = Y\beta$ where $X \not\equiv Y$, and either $\alpha \not\equiv \epsilon_Q$ or $\beta \not\equiv \epsilon_Q$.*

   (a) *For $|X| < |\beta|$, $\alpha X = Y\beta$ reduces to*

$$X = \beta[|\beta| - |X| + 1..|\beta|] \ \wedge \ Y = \alpha\beta[1..|\beta| - |X|] \ .$$

   (b) *For $|X| \geq |\beta|$, $\alpha X = Y\beta$ reduces to*

$$X = X'\beta \ \wedge \ Y = \alpha X' \ ,$$

   where X' is a fresh queue variable.

3. *Eliminate equalities of the form $\alpha X = X\beta$. By Proposition 3.2 if α, β are not conjugate, then $\alpha X = X\beta$ simplifies to* false. *If α, β are k-conjugate, $\alpha X = X\beta$ is replaced by $X \in \mathrm{orb}(\alpha, k)$.*

4. *Eliminate disequalities of the form $\alpha X \neq X\beta$. Again by Proposition 3.2, if α, β are not conjugate, $\alpha X \neq X\beta$ simplifies to* true. *If α, β are k-conjugate, $\alpha X \neq X\beta$ is replaced by $X \notin \mathrm{orb}(\alpha, k)$.*

Note that equalities are only *syntactically* eliminated in the normalization. Literals like $X \in \mathrm{orb}(\alpha, k)$ introduced in step (3) are implicit equalities, and they are the only kind equalities that can occur in a normal form. Theorem 3.1, however, ensures that a conjunction of more than one such equality is either inconsistent or a finite set of solutions can be computed, and that in the presence of $X \in \mathrm{orb}(\alpha, k)$, all occurrences $X \notin \mathrm{orb}(\alpha', k')$ (which are implicit disequalities introduced in step (4)) can be eliminated.

We claim that a constraint in normal form is satisfiable: a satisfiable assignment can be constructed incrementally by assigning each queue variable a queue with length distinct from all previously assigned terms.

Proposition 3.4 (Normal Form in $\mathfrak{Q}$). *A constraint in normal form is satisfiable.*

This proposition justifies the following algorithm.

Algorithm 3.2 (Decision Procedure for Queues [5]). *Input:* $\Phi \equiv \mathcal{E} \cup \mathcal{D}$.

1. *Transform* Φ *to* $\Phi' : \mathcal{E}' \cup \mathcal{D}'$ *which is in normal form.*

2. *If an inconsistency is discovered, return* FAIL*; otherwise, return* SUCCESS.

Note that an inconsistency may be due to a conflict in the atom domain. For example, a contradiction arises if Φ_Q requires at least $n+1$ distinct atoms in a domain of cardinality n. However, because of finiteness, any formula in $\mathscr{L}_\mathcal{A}$ is essentially ground by instantiation and hence the validity problem is readily decidable. Thus, for simplicity we omit discussions related to the atom domain in the above and the subsequent algorithms. Also note that in case of an infinite atom domain, a variable partition [22] serves the same purpose as instantiation, and of course the partition method is also applicable in the finite case.

3.4 Decision Procedures for $\mathrm{Th}^\forall(\mathfrak{Q}_\mathbb{Z})$

In this section we present decision procedures for $\mathrm{Th}^\forall(\mathfrak{Q}_\mathbb{Z})$, the quantifier-free theory of queues augmented with a length function and Presburger arithmetic. First, in Section 3.4.1 we describe the theory and outline the adaptation of the approach introduced in Section 2.4.2 for constructing decision procedures for this theory.

3.4.1 The Theory of Queues with Integers

Definition 3.6 (Queues with Integers). *The structure of queues with integers is* $\mathfrak{Q}_\mathbb{Z} : \langle \mathfrak{Q}, \mathrm{PA}; |\cdot| \rangle$ *where* $\mathfrak{Q}$ *is the structure of queues,* PA *is Presburger arithmetic, and* $|\cdot| : \mathcal{Q} \to \mathbb{N}$ *is the length function such that* $|X|$ *denotes the number of atoms in the queue* X.

We use subscripts $\mathcal{A}, \mathcal{Q}, \mathbb{Z}$ (or prefixes $\mathcal{A}$-, $\mathcal{Q}$-, PA-) to denote notions related to atom sort, queue sort and integer sort, respectively. For example, $\Phi_\mathcal{Q}$ denotes a queue formula and $\mathcal{V}_\mathcal{Q}$ denotes the collection of queue variables. As in Chapter 2, we use an integer term $|t(X)|$ in two ways; as the function value of $t(X)$ when $t(X)$ is in discussion, and as a purely syntactic integer variable (called *pseudo integer variable*). In the latter case, suppose $\Phi_\mathbb{Z}(\bar{\mathbf{X}})$ is given, then $\Phi_\mathbb{Z}(\bar{\mathbf{z}})$ is the formula obtained by substituting each pseudo integer variable $|X|$ ($X \in \bar{\mathbf{X}}$) for a real integer variable z ($z \in \bar{\mathbf{z}}$). $|\bar{\mathbf{X}}| = \bar{\mathbf{z}}$ denotes $\bigwedge_i |X_i| = z_i$. If σ is an assignment for $\mathcal{V}_\mathcal{Q}$, then $|\sigma|$ denotes the corresponding assignment for pseudo integer variables.

Similar as in term algebras, in a combined constraint $\Phi_\mathcal{Q} \wedge \Phi_\mathbb{Z}$, $\Phi_\mathbb{Z}$ restricts solutions to $\Phi_\mathcal{Q}$.

Example 3.5 (Hidden Length Constraints in $\mathfrak{Q}_{\mathbb{Z}}$). *The constraint*

$$\Phi_Q : Xba \neq abY \ \wedge \ Xab \neq baY \ \wedge \ Xaa \neq baY \ \wedge \ Xab \neq aaY$$

is not satisfiable with $\Phi_{\mathbb{Z}} : |X| = |Y| = 1$, in $\mathfrak{Q}_{\mathbb{Z}}$ with $\mathcal{A} = \{a, b\}$. It can be easily verified by enumerating all four combinations. On the other hand, both Φ_Q and $\Phi_{\mathbb{Z}}$ are obviously satisfiable in their respective domains.

Our combination strategy is exactly the same as that introduced in Section 2.4. We have the same notions of LCC, RLCC and the same high-level generic structure for combining decision procedures. For clarity, however, we rewrite the definitions and algorithms using the notation of queues.

Definition 3.7 (Length Constraint Completion (LCC) in $\mathfrak{Q}_{\mathbb{Z}}$). *A formula $\Phi_\Delta(\bar{\mathbf{X}})$ in $\mathscr{L}_{\mathbb{Z}}$ is a* length constraint completion *(LCC) for $\Phi_Q(\bar{\mathbf{X}})$ if the following formulae are valid:*

$$(\forall \bar{\mathbf{X}} : Q)\left[\Phi_Q(\bar{\mathbf{X}}) \ \rightarrow \ (\exists \bar{\mathbf{z}} : \mathbb{Z})\left(\Phi_\Delta(\bar{\mathbf{z}}) \ \wedge \ |\bar{\mathbf{X}}| = \bar{\mathbf{z}}\right)\right] , \tag{3.2}$$

$$(\forall \bar{\mathbf{z}} : \mathbb{Z})\left[\Phi_\Delta(\bar{\mathbf{z}}) \ \rightarrow \ (\exists \bar{\mathbf{X}} : Q)\left(\Phi_Q(\bar{\mathbf{X}}) \ \wedge \ |\bar{\mathbf{X}}| = \bar{\mathbf{z}}\right)\right] . \tag{3.3}$$

where $\Phi_\Delta(\bar{\mathbf{z}})$ is obtained from $\Phi_\Delta(\bar{\mathbf{X}})$ by substituting a fresh z for each pseudo integer variable of the form $|t(X)|$.

Example 3.6 (LCC in $\mathfrak{Q}_{\mathbb{Z}}$). *Consider Φ_Q in Example 3.5 and,*

$$\Phi_{\Delta+} : \text{true} \ , \qquad \Phi_{\Delta-} : |X| = |Y| = 2 \ , \qquad \Phi_\Delta : |X| = |Y| \wedge |X| \neq 1 \ .$$

$\Phi_{\Delta+}$ is not realizable by Φ_Q because the integer assignment $\sigma_\Delta : \{|X| = |Y| = 1\}$ can not be realized. On the other hand, $\Phi_{\Delta-}$ is not sound because it does not satisfy the queue assignment $\sigma_Q : \{X = \epsilon_Q, Y = \epsilon_Q\}$. Finally, Φ_Δ is both sound and realizable w.r.t. Φ_Q and hence is an LCC for Φ_Q.

Definition 3.8 (Relativized LCC (RLCC) in $\mathfrak{Q}_{\mathbb{Z}}$). *A formula $\Phi_\Delta(\bar{\mathbf{X}})$ is a* length constraint completion *for $\Phi_Q(\bar{\mathbf{X}})$ relativized to $\theta_{\mathbb{Z}}(\bar{\mathbf{X}})$, (in short, $\Phi_\Delta(\bar{\mathbf{X}})$ is an RLCC for $\Phi_Q(\bar{\mathbf{X}})/\theta_{\mathbb{Z}}(\bar{\mathbf{X}})$), if the following formulae are valid:*

$$(\forall \bar{\mathbf{X}} : Q)\left[\Phi_Q(\bar{\mathbf{X}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{X}}) \ \rightarrow \ (\exists \bar{\mathbf{z}} : \mathbb{Z})\left(\Phi_\Delta(\bar{\mathbf{z}}) \wedge |\bar{\mathbf{X}}| = \bar{\mathbf{z}}\right)\right] , \tag{3.4}$$

$$(\forall \bar{\mathbf{z}} : \mathbb{Z})\left[\Phi_\Delta(\bar{\mathbf{z}}) \ \rightarrow \ (\exists \bar{\mathbf{X}} : Q)\left(\Phi_Q(\bar{\mathbf{X}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{X}}) \wedge |\bar{\mathbf{X}}| = \bar{\mathbf{z}}\right)\right] . \tag{3.5}$$

Example 3.7 (RLCC in $\mathfrak{Q}_{\mathbb{Z}}$). *Revisiting Example 3.6, we partition Φ_Q into*

$$(\Phi_Q \ \wedge \ |X| \neq |Y|) \ \vee \ (\Phi_Q \ \wedge \ |X| = |Y|) \ .$$

The first disjunct simplifies to $|X| \neq |Y|$ as $|X| \neq |Y|$ implies Φ_Q. Now consider the second disjunct. It is clear that the RLCC for $\Phi_Q/(|X| = |Y|)$ is $|X| = |Y| \wedge |X| \neq 1$.

Similar to Algorithm 2.3, we have

Algorithm 3.3 (Generic Decision Procedure). *Input:* $\Phi_Q \wedge \Phi_{\mathbb{Z}}$.

1. *Return* FAIL *if* TA $\not\models_\exists \Phi_Q$.
2. *For each partition* $\Phi_Q^{(i)} \wedge \theta_{\mathbb{Z}}^{(i)}$ *of* Φ_Q*:*
   (a) *Compute an RLCC* $\Phi_\Delta^{(i)}$ *for* $\Phi_Q^{(i)}/\theta_{\mathbb{Z}}^{(i)}$.
   (b) *Return* SUCCESS *if* PA $\models_\exists \Phi_\Delta^{(i)} \wedge \Phi_{\mathbb{Z}}$.
3. *Return* FAIL.

3.4.2 Decision Procedure for $\mathrm{Th}^\forall(\mathfrak{Q}_{\mathbb{Z}})$

Recall that our strategy is to partition the search space for Φ_Q until the *RLCC* is easily obtained. The partition divides into a series of steps. When $|X|$ is known to be bounded by a constant l, we can instantiate X with a constant queue of length l. As $\mathcal{A}$ is finite, there are only finitely many such queues. When we make changes to Φ_Q by introducing new variables or instantiation, $\theta_{\mathbb{Z}}$ and $\Phi_{\mathbb{Z}}$ are updated accordingly. For example, in case X is replaced by $X'\alpha$, $|X|$ is replaced by $|X'| + |\alpha|$. To save notation, Φ_Q and $\theta_{\mathbb{Z}}$ always refer to the updated version for one of the partitions.

First we assume $\Phi_Q \wedge \theta_{\mathbb{Z}}$ satisfies the following condition.

Definition 3.9 (Equality Completeness). $\Phi_Q \wedge \theta_{\mathbb{Z}}$ *is* equality complete *if* $t_1 \neq t_2 \in \Phi_Q$, *then* $|t_1| = |t_2| \in \theta_{\mathbb{Z}}$.

This condition can be realized by the following algorithm.

Algorithm 3.4 (Equality Completion). *Input:* $\Phi_Q \wedge \theta_{\mathbb{Z}}$. *For each* $t_1 \neq t_2$,

1. *Add exactly one of* $|t_1| = |t_2|$ *and* $|t_1| \neq |t_2|$ *to* $\theta_{\mathbb{Z}}$.
2. *Remove* $t_1 \neq t_2$ *from* Φ_Q *if* $|t_1| \neq |t_2|$ *is in* $\theta_{\mathbb{Z}}$.

Definition 3.10 (Normal Form in $\mathfrak{Q}_{\mathbb{Z}}$). Φ_Q *is in* normal form *in* $\mathfrak{Q}_{\mathbb{Z}}$ *if* Φ_Q *satisfies Definition 3.5 and satisfies*

1. *no occurrence of disequalities like* $X \neq \alpha$;
2. *if* $\alpha X \neq Y\beta$ *occurs with either* $X \in \mathrm{orb}(\alpha', k)$ *or* $Y \in \mathrm{orb}(\beta', l)$, *then* $\alpha \equiv \epsilon_Q$;
3. $\alpha X \neq Y\beta$ *does not occur with both* $X \in \mathrm{orb}(\alpha', k)$ *and* $Y \in \mathrm{orb}(\beta', l)$.

Algorithm 3.5 (Normalization in $\mathfrak{Q}_{\mathbb{Z}}$). *Repeatedly execute the following sub-procedures until no update can be made.*

1. *Call Algorithm 3.1 to normalize* Φ_Q.

2. *For disequalities* $X \neq \alpha$, *we must have* $|X| = |\alpha|$ *due to equality completeness, and hence* X *can be removed by instantiation. For disequalities* $\alpha X \neq Y\beta$ *with* $|X| < |\beta|$ *or* $|Y| < |\alpha|$, *replace* X *and* Y *by instantiations. In the remaining steps we assume* $|X| \geq |\beta|$ *and* $|Y| \geq |\alpha|$ *are in* $\theta_\mathbb{Z}$.

3. *Consider each subconstraint of the form*

$$\alpha X \neq Y\beta \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ , \tag{3.6}$$

which asserts that X *is of the form* $(\alpha')^*\alpha'[1..k]$ *and similar for* Y. *If* β *is not a prefix of* X *or* α *is not a prefix of* Y, $\alpha X \neq Y\beta$ *simplifies to* true. *Otherwise* (3.6) *can be replaced by*

$$X' \neq Y' \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ \wedge\ X = X'\beta \ \wedge\ Y = \alpha Y'$$

which can be further reduced to

$$\begin{aligned} &X' \neq Y' \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \\ &\qquad \wedge\ X' \in \mathrm{orb}(\alpha', k') \ \wedge\ Y' \in \mathrm{orb}(\beta'', l') \ \wedge\ |X| = |\beta| + |X'| \ \wedge\ |Y| = |\alpha| + |Y'| \ , \end{aligned} \tag{3.7}$$

where

$$k' = \Big(k + |\alpha'| - (|\beta| \bmod |\alpha'|)\Big) \bmod |\alpha'| \ , \tag{3.8}$$

$$\beta'' = \beta'[(|\alpha| \bmod |\beta'|) + 1..|\beta'|] \circ \beta'[1..(|\alpha| \bmod |\beta'|)] \ , \tag{3.9}$$

$$l' = \Big(l + |\beta'| - (|\alpha| \bmod |\beta'|)\Big) \bmod |\beta'| \ . \tag{3.10}$$

If $\alpha' = \beta''$, *then* $k' = l'$, *because* $|X'| = |Y'|$. *Thus* (3.7) *is* false *and so is* (3.6). *If* $\alpha' \neq \beta''$, *then there are only finitely many cases that* $X' = Y'$ *which can be computed and excluded.*

4. *Consider each subconstraint of the form*

$$\alpha X \neq Y\beta \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ . \tag{3.11}$$

Guess a word α'' *such that* $|\alpha''| = |\alpha|$ *and set* $Y = \alpha''Y'$. *For* $\alpha \not\equiv \alpha''$, *replace* $\alpha X \neq Y\beta$ *by* $Y = \alpha''Y'$, *otherwise, replace* $\alpha X \neq Y\beta$ *by* $Y = \alpha Y' \ \wedge\ X \neq Y'\beta$.

5. *Consider each subconstraint of the form*

$$\alpha X \neq Y\beta \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ . \tag{3.12}$$

If α *is not a prefix of* Y *(which has the form* $(\beta')^*\beta'[1..l]$*), then* (3.12) *simplifies to* true. *Otherwise*

(3.12) *can be replaced by*

$$X \neq Y'\beta \ \wedge \ Y \in \mathrm{orb}(\beta', l) \ \wedge \ Y = \alpha Y' \ ,$$

which can be further simplified to

$$X \neq Y'\beta \ \wedge \ Y \in \mathrm{orb}(\beta', l) \ \wedge \ Y' \in \mathrm{orb}(\beta'', l') \ \wedge \ |Y| = |\alpha| + |Y'| \tag{3.13}$$

with β'' and l' the same as in (3.9) *and* (3.10) *respectively.*

Proposition 3.5 (Normal Form in $\mathfrak{Q}_{\mathbb{Z}}$). *Algorithm 3.5 preserves equivalence and terminates with Φ_Q in the normal form as given by Definition 3.10.*

Algorithm 3.6 (Computation of $\Phi_{\Delta+}$ in $\mathfrak{Q}_{\mathbb{Z}}$). *Input: $\Phi_Q \wedge \theta_{\mathbb{Z}}$. Initially set $\Phi_{\Delta+} = \emptyset$. Add to $\Phi_{\Delta+}$:*

1. $|t_1| = |t_2|$, *if $t_1 \neq t_2$ or $t_1 = t_2$;*
2. $|X| + |\alpha| = |\alpha X| = |X\alpha|$, *if αX or $X\alpha$ occurs;*
3. $|X| \equiv k(\bmod |\alpha|)$, *if $X \in \mathrm{orb}(\alpha, k)$.*

Φ_Q can be satisfied by sufficiently long queues: there exists a *cut length* δ such that if $\mathfrak{Q} \models_{\exists} \Phi_Q$, then for any solution $(l_i)_n$ (i. e., $l_0, \ldots, l_n$) for $\Phi_{\Delta+}$ such that $l_i \geq \delta$, there exists a solution $(\alpha_i)_n$ for Φ_Q such that $|\alpha_i| = l_i$. Let $C_\Phi(\delta)$ denote

$$\bigwedge_{X \in \mathcal{V}_Q(\Phi_Q)} |X| \geq \delta \ .$$

It is clear that $\Phi_{\Delta+} \wedge C_\Phi(\delta) \wedge \theta_{\mathbb{Z}}$ is an RLCC for $\Phi_Q/(\theta_{\mathbb{Z}} \wedge C_\Phi(\delta))$. It is not true, however, that δ is the smallest $max\{(\mu_i)_n\}$ such that

$$\mathfrak{Q}_{\mathbb{Z}} \models_{\exists} \Phi_Q \ \wedge \ \bigwedge_{0<i\leq n} |X_i| = \mu_i \ ,$$

where $(X_i)_n$ enumerate $\mathcal{V}_Q(\Phi_Q)$.

Example 3.8 (Anomaly). *Revisiting Example 3.5 we see an anomaly where $\{X := \epsilon_Q, Y := \epsilon_Q\}$ is a solution for Φ_Q (with $|X| = |Y| = 0$), while there exists no solution for Φ_Q such that $|X| = |Y| = 1$.*

To avoid such anomalies we separate the search for a satisfying assignment into two cases. We compute a cut length $L_t \geq \delta$ and enumerate all assignments σ with $|[\![X]\!]\sigma| < L_t$, while for $|[\![X]\!]\sigma| \geq L_t$ the satisfiability of the queue constraints is reduced to the satisfiability of integer constraints in the same way as in Chapter 2.

The computation of L_t is based on the observation that an assignment σ is satisfying if every $[\![X]\!]\sigma$ includes a unique "marker" at the same, fixed, position. Such a marker can be constructed by concatenating a "shortest unused prefix" and a unique identifier for each queue variable and proper constant queue.

Let PRE_Φ denote the set of all words α such that αX is a proper term in Φ_Q. Let $|\mathrm{PRE}_\Phi|$ be the cardinality of PRE_Φ and let

$$\sup(\mathrm{PRE}_\Phi) = \max\{\,|\alpha| \mid \alpha \in \mathrm{PRE}_\Phi\,\} \ .$$

Definition 3.11 (Delimiters in $\mathfrak{Q}$). *A word q is called a* delimiter *of Φ_Q in $\mathscr{L}_{\mathfrak{Q}}$ if q is strongly primitive and $q \notin \mathrm{orb}(\alpha)$ for any $\alpha \in \mathrm{PRE}_\Phi$.*

A delimiter always exists for a finite PRE_Φ. In particular any strongly primitive word longer than $\sup(\mathrm{PRE}_\Phi)$ is a delimiter. In general, the length of a delimiter can be much shorter, because for a fixed k, there are at most $|\mathrm{PRE}_\Phi|$ distinct prefixes of words in PRE_Φ having length k, while there are $2^{O(k)}$ strongly primitive words of length k provided $\mathscr{L}_{\mathfrak{Q}}$ has more than two atoms.

Now let d_p denote an arbitrary shortest delimiter (there can be more than one) and let $L_p = |d_p|$. Let L_c be the smallest number of letters necessary to create a unique identifying word d_c, called a *color*, for each queue variable and constant queue properly occurring in Φ_Q. We define a *tag* as a concatenation of d_p and a color d_c. We say that a word α is *tagged* by q, if q is a tag and is a prefix of α. We claim that $L_c + L_p = L_t \geq \delta$.

Example 3.9 (Computation of L_t). *Consider again Φ_Q in Example 3.5. Here $\mathrm{PRE}_\Phi = \{ab, ba, aa\}$; a shortest delimiter is aab, and thus $L_p = 3$. Φ_Q includes two queue variables, requiring one letter to identify them with two letters in the alphabet. Thus, we need four letters to construct a unique identifying word, resulting in $L_t = 4$.*

Theorem 3.2 (RLCC in $\mathfrak{Q}_{\mathbb{Z}}$). *$\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L_t)$ is an RLCC for $\Phi_Q/(\theta_{\mathbb{Z}} \wedge C_\Phi(L_t))$.*

Definition 3.12 (Length Configurations in $\mathfrak{Q}_{\mathbb{Z}}$). *A* length configuration *for Φ_Q (in $\mathfrak{Q}_{\mathbb{Z}}$) is a conjunction*

$$\bigwedge_{X \in \mathcal{V}_Q(\Phi_Q)} A_X \ , \tag{3.14}$$

where A_X is either $|X| = i$ (for some $i < L_t$) or $|X| \geq L_t$.

Let C be the set of all configurations. Clearly C creates a finite partition of the search space that includes $C_\Phi(L_t)$. A partial assignment ∂ is *compatible* with a configuration C if for any variable X, $[\![X]\!]\partial$ is defined if and only if $|X| = i$ (for some $i < L_t$) occurs in C. The empty assignment is vacuously a satisfying partial assignment, the only one compatible with $C_\Phi(L_t)$. It can be shown

that a C-compatible substitution can only reduce PRE_Φ and hence L_t. So $\Phi_{\Delta+} \wedge C \wedge \theta_{\mathbb{Z}}$ (all updated by variable instantiation) is an RLCC for $\Phi_Q/(\theta_{\mathbb{Z}} \wedge C)$ (also the updated version after by variable instantiation). As a consequence of Theorem 3.2, we have

Algorithm 3.7 (Decision Procedures for $\mathfrak{Q}_{\mathbb{Z}}$). *Input: $\Phi_Q \wedge \theta_{\mathbb{Z}} \wedge \Phi_{\mathbb{Z}}$ where $\Phi_Q \wedge \theta_{\mathbb{Z}}$ denotes one of the partitions.*

1. *For each $C \in \mathcal{C}$,*
   (a) *Guess a satisfying ∂ compatible with C and update C, $\Phi_{\Delta+}$, $\theta_{\mathbb{Z}}$ and $\Phi_{\mathbb{Z}}$ accordingly.*
   (b) *If succeed, return* SUCCESS *if* $\text{PA} \models_\exists C \wedge \Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge \Phi_{\mathbb{Z}}$.
2. *Return* FAIL.

In fact we can encode configurations more efficiently because $\Phi_{\Delta+}$ induces a partition on queue variables. Let Φ_Q and $\Phi_{\Delta+}$ be as stated in Algorithm 3.6. We say that two queue variables X, Y in $\mathcal{V}(\Phi_Q)$ are *connected*, written $X \sim_{\mathbb{Z}} Y$, if $|X| = |Y| + m$ is implied by $\Phi_{\Delta+}$. We write $X \leq_{\mathbb{Z}} Y$ if $m \geq 0$. It is easily seen that connectedness relation $\sim_{\mathbb{Z}}$ is an equivalence relation. A *block* (w.r.t. Φ_Q) is an equivalence class induced by $\sim_{\mathbb{Z}}$. Variables in a block B can be linearly ordered by $\leq_{\mathbb{Z}}$. We call a minimum in B with respect to $\leq_{\mathbb{Z}}$ the *base variable* of B, written X_B. Let $X_0, \ldots, X_n$ be an increasing enumeration of variables in B. We have $X_0 \equiv X_B$, $|X_i| = |X_B| + l_i$ where $i \leq n$ and $l_i \leq l_j$ for $i \leq j \leq n$. It is obvious that we only need to consider configurations on base variables.

Example 3.10 (Blocks). *Consider Φ_Q be*

$$Xab \neq bY \ \wedge \ abaX \neq Zb \ \wedge \ W \in \text{orb}(aba, 1) \ .$$

Then $\Phi_{\Delta+}$ implies $|Y| = |X| + 1$ and $|Z| = |X| + 2$, which induces two blocks $B_1 : \{X, Y, Z\}$ and $B_2 : \{W\}$. We have $X_{B_1} = X$, $X_{B_2} = W$, and X, Y, Z are in the increasing order with respect to $\leq_{\mathbb{Z}}$. It suffices to only consider configurations on X and W.

Theorem 3.3 (Complexity of $\text{Th}^\forall(\mathfrak{Q}_{\mathbb{Z}})$). *The decision problem for $\text{Th}^\forall(\mathfrak{Q}_{\mathbb{Z}})$ is NP-complete.*

Proof. Clearly $\text{Th}^\forall(\mathfrak{Q}_{\mathbb{Z}})$ is NP-hard as it is a super theory of $\text{Th}^\forall(\mathfrak{Q})$ and $\text{Th}^\forall(\mathbb{Z})$, which are both NP-complete. Algorithm 3.6 computes $\Phi_{\Delta+}$ in $O(n)$. As $|\text{PRE}_\Phi|$ is bounded by $O(n)$, L_t can be computed in $O(n)$ too. By the nondeterministic nature of our algorithms, we can show that each branch of computation in the normalization procedures and in Algorithm 3.7 is in P. Therefore $\text{Th}^\forall(\mathfrak{Q}_{\mathbb{Z}})$ is NP-complete. □

3.5 Decision Procedure for $\text{Th}^\forall(\mathfrak{Q}^+_{\mathbb{Z}})$

In this section we add the prefix relation to the theory of queues, and construct a decision procedure for this theory of queues with integers.

Definition 3.13 (Queues with Prefix). *The structure of queues with prefix relation is* $\mathfrak{Q}^{+} : \langle \mathfrak{Q}; \prec \rangle$ *where* $\mathfrak{Q}$ *is as in Definition 3.1 and* $\prec : Q \times Q \to \mathbb{B}$ *such that* $\langle s_0, \ldots, s_n \rangle \prec \langle t_1, \ldots, t_m \rangle$ *if and only if* $m < n$ *and* $s_i = t_i$ *for any* $i \leq m$. *Let* $\mathscr{L}_{Q^+}$ *denote the language of* $\mathfrak{Q}^{+}$.

Definition 3.14 (Queues with Prefix and Integers). *The structure of queues with prefix relation and integers is* $\mathfrak{Q}^{+}_{\mathbb{Z}} : \langle \mathfrak{Q}^{+}, \mathrm{PA}; |\cdot| \rangle$. *Let* $\mathscr{L}^{\mathbb{Z}}_{Q^+}$ *denote the language of* $\mathfrak{Q}^{+}_{\mathbb{Z}}$.

The basic idea is simple; finite words are partially ordered by the prefix relation into a tree structure. Intuitively we want to order all queue variables in a given formula into a tree (assuming ϵ_Q be the common root) where each path is induced by a chain of positive $\prec$-literals in the form

$$X_0\beta_0 \prec X_1 \ , \quad X_1\beta_1 \prec X_2 \ , \quad \ldots \ , \quad X_n\beta_n \prec X_{n+1} \ . \tag{3.15}$$

In this way we are able to build a solution inductively from the root. However, the normalization procedure needs considerable modification to deal with combinations of literals. Eventually we are looking for a normal form satisfying the following conditions.

Definition 3.15 (Normal Form in $\mathfrak{Q}^{+}_{\mathbb{Z}}$). Φ_Q *is in* normal form *(in* $\mathfrak{Q}^{+}_{\mathbb{Z}}$*) if* Φ_Q *satisfies Definition 3.5 and the following conditions:*

1. *All literals contain two variables.*
2. *Negative* $\prec$*-literals are either in the form* $\alpha X \not\prec Y$ *or in the form* $X \not\prec \alpha Y$ *(for* $X \not\equiv Y$*).*
3. *Positive* $\prec$*-literals in* Φ_Q *are in the form* $X\beta \prec Y$ *(for* $X \not\equiv Y$*) and they partially order queue variables into a tree structure in which each path is induced by a chain of the form* (3.15)*.*
4. *For a chain like* (3.15)
   (a) *if* $X_i \in \mathrm{orb}(\alpha, k)$ *occurs, then* $i = 0$*, and*
   (b) *no literals of the form* $X_i \not\prec X_j$ $(i < j \leq n+1)$ *occurs.*
5. *For each of the following combinations,* α *must be* ϵ_Q*.*
   (a) $\alpha X \neq Y\beta$ *occurs with* $Y \in \mathrm{orb}(\beta', l)$*.*
   (b) $\alpha X \not\prec Y$ *occurs with* $Y \in \mathrm{orb}(\beta', l)$*.*
   (c) $X \not\prec \alpha Y$ *occurs with* $X \in \mathrm{orb}(\beta', l)$*.*
6. *None of* $\alpha X \neq Y\beta$, $\alpha X \not\prec Y$ *and* $X \not\prec \alpha Y$ *occurs with both* $X \in \mathrm{orb}(\alpha', k)$ *and* $Y \in \mathrm{orb}(\beta', l)$*.*
7. *If* $\alpha X \neq Y\beta$ *appears, then* X *and* Y *do not occur in the same chain.*
8. $X \not\prec \alpha Y$ *does not occur with* $Y\beta \prec X$*.*

9. *There does not exist a literal $\alpha X \not\prec Y$ such that*

   (a) *X appears in a positive orbit literal while Y does not appear in any positive orbit literal.*

   (b) *Y appears in a chain whose head X' appears in a positive orbit literal, and $X'\beta \prec Y$ is induced by the chain.*

   (c) *$\theta_{\mathbb{Z}}$ does not contain $|\alpha| + |X| \geq |X'| + |\beta| + |d_p|$ were d_p is the delimiter in $\mathscr{L}_{\mathbb{Q}^+}$ defined in the sequel.*

For a chain of the form (3.15), we call X_0 the *head* of the chain. We say that the chain is *pure* if X_0 does not appear in positive orbit literals. Note that with respect to =-literals only, condition (2) in the above definition is in fact weaker than condition (2) in Definition 3.10 as the former requires α be ϵ_Q only when Y appears in an orbit literal while the latter in addition requires α be ϵ_Q when X appears in an orbit literal. This is a technical choice to avoid generating literals of the form $Y = \alpha Y'$ as in step (3.11) in Algorithm 3.5.

We divide this normalization into several steps. First we define *prefix completeness*.

Definition 3.16 (Prefix Completeness). $\Phi_Q \wedge \theta_{\mathbb{Z}}$ *is* prefix complete *if the following two conditions are satisfied.*

1. *if $t_1 \not\prec t_2 \in \Phi_Q$, then $|t_1| < |t_2| \in \theta_{\mathbb{Z}}$.*

2. *if $t_1 \prec t \in \Phi_Q$ and $t_2 \prec t \in \Phi_Q$, then exactly one of $t_1 = t_2$, $t_1 \prec t_2$ and $t_2 \prec t_1$ is in Φ_Q.*

$\Phi_Q \wedge \theta_{\mathbb{Z}}$ can be made prefix complete by the following algorithm

Algorithm 3.8 (Prefix Completion). *Input: $\Phi_Q \wedge \theta_{\mathbb{Z}}$.*

1. *For each $t_1 \prec t$ and $t_2 \prec t$ in Φ_Q, add exactly one of $t_1 = t_2$, $t_1 \prec t_2$ and $t_2 \prec t_1$ to Φ_Q.*

2. *For each $t_1 \not\prec t_2$,*

   (a) *Add exactly one of $|t_1| = |t_2|$, $|t_1| > |t_2|$ and $|t_1| < |t_2|$ to $\theta_{\mathbb{Z}}$.*

   (b) *Remove $t_1 \not\prec t_2$ from Φ_Q if $|t_1| > |t_2|$ or $|t_1| = |t_2|$ is in $\theta_{\mathbb{Z}}$.*

Step (2) is justified by the fact that prefixes of a term are linearly ordered by $\prec$. In general suppose we have $\bigwedge_{0<i\leq n} t_i \prec t$, we guess an *order completion* of t_i:

$$t_{p(1)} \lhd t_{p(2)} \lhd \cdots \lhd t_{p(n)} \prec t \ ,$$

where p is permutation function on $[1, n]$ and $\lhd$ stands for either $=$ or $\prec$.

We define a partial order with respect to a given constraint $\Phi : \Phi_Q \wedge \theta_{\mathbb{Z}}$ as follows.

$$X <_\Phi Y \quad \text{iff} \quad (\exists \alpha, \alpha', \beta, \beta' \in Q)\ \alpha X \beta \prec \alpha' Y \beta' \in \Phi_Q \ .$$

Obviously, $<_\Phi$ is a transitive relation, and for simplicity, we also use $<_\Phi$ to denote the corresponding transitive closure.

Condition (1) of Definition 3.15 can be satisfied with the following algorithm.

Algorithm 3.9 (Elimination of Single Variable Literals).

1. *For literals of the forms $X \neq \alpha$, $X\beta \prec \alpha$, $X\beta \not\prec \alpha$, we instantiate X because by equality completeness or prefix completeness, $|X|$ is bounded.*

2. *$\alpha \prec X\beta$. Instantiate X if we guess $|X| \leq |\alpha|$. Otherwise rewrite $\alpha \prec X\beta$ to $X = \alpha X' \wedge |X'| > 0$.*

3. *$\alpha \not\prec X\beta$. Instantiate X if we guess $|X| < |\alpha|$. Otherwise guess an $\alpha' \not\equiv \alpha$ such that $|\alpha'| = |\alpha|$, and rewrite $\alpha \not\prec X\beta$ to $X = \alpha' X'$.*

Condition (2) of Definition 3.15 can be satisfied with the following algorithm.

Algorithm 3.10 (Simplification of Negative $\prec$ Literals).

1. *β-Elimination. By peeling off constant prefixes or suffixes coordinate-wise, we can assume negative $\prec$-literals are of the form*

$$\alpha X\beta \not\prec Y\beta' \ , \tag{3.16}$$

$$X\beta \not\prec \alpha' Y\beta' \ . \tag{3.17}$$

Rewrite (3.16)-(3.17)*, respectively, to*

$$\alpha X\beta \not\prec Y \ \vee \bigvee_{\beta'' \not\prec \beta' \ \wedge \ |\beta''| < |\beta'|} \alpha X\beta = Y\beta'' \ , \tag{3.18}$$

$$X\beta \not\prec \alpha' Y \ \vee \bigvee_{\beta'' \not\prec \beta' \ \wedge \ |\beta''| < |\beta'|} X\beta = \alpha' Y\beta'' \ . \tag{3.19}$$

Now we can assume that negative $\prec$-literals are in the forms

$$\alpha X\beta \not\prec Y \ , \tag{3.20}$$

$$X\beta \not\prec \alpha' Y \ . \tag{3.21}$$

Continue to rewrite (3.20)-(3.21)*, respectively, to*

$$(\alpha X \not\prec Y \ \wedge \ |\alpha X| < |Y|) \ \vee \bigvee_{\beta' \not\equiv \beta \ \wedge \ |\beta'| \leq |\beta|} \alpha X\beta' \prec Y \ , \tag{3.22}$$

$$(X \not\prec Y \ \wedge \ |X| < |Y|) \ \vee \bigvee_{\beta' \not\equiv \beta \ \wedge \ |\beta'| \leq |\beta|} X\beta' \prec Y \ . \tag{3.23}$$

2. *Self-loop Elimination.*

   (a) *Rewrite* $\alpha X \not\prec X$ *to* true.

   (b) *Rewrite* $X \not\prec \alpha X$ *to*

$$\bigwedge_{k<|\alpha|} X \notin \mathrm{orb}(\alpha, k) \ .$$

In order to satisfy condition (3) of Definition 3.15, we need first to put positive $\prec$ literals into the form of $X\beta \prec Y$. Intuitively this can be achieved by the following two simple algorithms.

Algorithm 3.11 (Upward Alignment). *Input:* $\alpha X\beta \prec Y$.

Case $|Y| < |\alpha|$. *Instantiate* Y.

Case $|Y| \geq |\alpha|$. *Rewrite* $\alpha X\beta \prec Y$ *to* $X\beta \prec Y' \ \wedge \ Y = \alpha Y'$.

Algorithm 3.12 (Downward Alignment). *Input:* $X\beta \prec \alpha Y$.

Case $|X| < |\alpha|$. *Instantiate* X.

Case $|X| \geq |\alpha|$. *Rewrite* $X\beta \prec \alpha Y$ *to* $X'\beta \prec Y \ \wedge \ X = \alpha X'$.

However, we cannot simply call Algorithms 3.11 and 3.12 to get all positive $\prec$-literals into the form of $X\alpha \prec Y$. For example, consider the formula

$$aX \prec Y \ \wedge \ X \prec Y \ . \tag{3.24}$$

Applying Algorithm 3.11 to (3.24) we will obtain (one of choices)

$$X \prec Y' \ \wedge \ X \prec aY' \ . \tag{3.25}$$

Then applying Algorithm 3.12 to (3.25) we will obtain (one of choices)

$$aX' \prec Y' \ \wedge \ X' \prec Y' \ , \tag{3.26}$$

which is equivalent to (3.24) up to variable renaming.

To guarantee progress, we need the following sub-conditions.

1. Positive $\prec$-literals are of the form

$$\alpha X\beta \prec Y \quad (X \not\equiv Y) \ , \tag{3.27}$$

$$X\beta \prec \alpha' Y \quad (X \not\equiv Y) \ . \tag{3.28}$$

2. There exists no variable X such that $X <_{\Phi} X$.

3. For any X, Y, if $X <_{\Phi} Y$, there exists at most one path from X to Y. More precisely, if

$$\begin{aligned}&\alpha_0 X\beta_0 \prec \alpha'_1 X_1 \beta'_1 \ , \quad \alpha_1 X_1 \beta_1 \prec \alpha'_2 X_2 \beta'_2 \ , \quad \ldots \ , \quad \alpha_{n-1} X_{n-1} \beta_{n-1} \prec \alpha'_n Y \beta'_n \ ,\\ &\gamma_0 X\eta_0 \prec \gamma'_1 X'_1 \eta'_1 \ , \quad \gamma_1 X'_1 \eta_1 \prec \gamma'_2 X'_2 \eta'_2 \ , \quad \ldots \ , \quad \gamma_{m-1} X'_{m-1} \eta_{m-1} \prec \gamma'_m Y \eta'_m \ ,\end{aligned}$$

then we have $m = n$, $\alpha_i \equiv \gamma_i$, $\alpha'_i \equiv \gamma'_i$, $\beta_i \equiv \eta_i$, $\beta'_i \equiv \eta'_i$ and $X_i \equiv X'_i$ for $0 < i \leq m-1$.

Subcondition (1) can be satisfied with the following algorithm.

Algorithm 3.13 (Simplification of Positive $\prec$ Literals).

1. *By peeling off constant prefixes or suffixes coordinate-wise, we can assume $\prec$-literals are of the form*

$$\alpha X\beta \prec Y\beta' \ , \tag{3.29}$$

$$X\beta \prec \alpha' Y\beta' \ . \tag{3.30}$$

Rewrite (3.29)-(3.30)*, respectively, to*

$$\alpha X\beta \prec Y \ \vee \bigvee_{i<|\beta'|} \alpha X\beta = Y\beta'[1..i] \ , \tag{3.31}$$

$$X\beta \prec \alpha' Y \ \vee \bigvee_{i<|\beta'|} X\beta = \alpha' Y\beta'[1..i] \ . \tag{3.32}$$

2. *Self-loop Elimination.*

   (a) *Case $\alpha X\beta \prec X$. Simplify $\alpha X\beta \prec X$ to* false.

   (b) *Case $X\beta \prec \alpha X$.*

      i. *If $|\alpha| \leq |\beta|$, then $X\beta \prec \alpha X$ simplifies to* false.

      ii. *Otherwise, it simplifies to*

$$\bigvee_{\{\tilde{\beta} \,|\, |\tilde{\beta}|=|\alpha|-|\beta|\}} X\beta\tilde{\beta} = \alpha X \ .$$

Sub-condition (2) can be satisfied by the following algorithm.

Algorithm 3.14 (Loop Elimination). *Repeat until there is no chain of the form*

$$\alpha_0 X\beta_0 \prec \alpha'_1 X_1 \beta'_1 \ , \quad \alpha_1 X_1 \beta_1 \prec \alpha'_2 X_2 \beta'_2 \ , \ \ldots \ , \quad \alpha_{n-1} X_{n-1} \beta_{n-1} \prec \alpha'_n X \beta'_n \ .$$

Choose a minimal chain of the above form (i. e., no subchain of it has this form). Consider two cases.

1. *The length of the chain is* 1. *It must be the case that* $X\beta \prec \alpha X$, *which is rewritten by step* (2) *in Algorithm 3.13 to a disjunction of equalities accordingly.*

2. *The length of the chain is greater than* 1. *There are four combinations for the first two positive* $\prec$*-literals.*

$$\begin{array}{lll} (1) & \alpha_1 X\beta_1 \prec Y \ , & \alpha_2 Y\beta_2 \prec Z \ ; \\ (2) & \alpha_1 X\beta_1 \prec Y \ , & Y\beta_2 \prec \alpha_2' Z \ ; \\ (2) & X\beta_1 \prec \alpha_1' Y \ , & \alpha_2 Y\beta_2 \prec Z \ ; \\ (4) & X\beta_1 \prec \alpha_1' Y \ , & Y\beta_2 \prec \alpha_2' Z \ . \end{array}$$

By Algorithm 3.11 we can rewrite them respectively to

$$\begin{array}{lll} (1) & X\beta_1 \prec Y' \ , & \alpha_2\alpha_1 Y'\beta_2 \prec Z \ ; \\ (2) & X\beta_1 \prec Y' \ , & \alpha_1 Y'\beta_2 \prec \alpha_2' Z \ ; \\ (3) & X\beta_1 \prec \alpha_1' Y \ , & \alpha_2 Y\beta_2 \prec Z \ ; \\ (4) & X\beta_1 \prec \alpha_1' Y \ , & Y\beta_2 \prec \alpha_2' Z \ . \end{array}$$

Continue with Y *or* Y' *in the chain. Eventually we have a left aligned tower of words sandwiched between* $X\beta$ *and* αX, *By step* (2) *in Algorithm 3.13* $X\beta \prec \alpha X$ *either is* false *or is equivalent to* $X \in \mathrm{orb}(\alpha, k)$ *for some* $k < |\alpha|$. *All other variables appearing in the chain must be in* $\mathrm{orb}(\alpha)$. *Literals of the form* $t_1 \prec t_2$ *in the chain then can be replaced by*

$$|t_1| < |t_2| \ \wedge \ t_1 \in \mathrm{orb}(\alpha) \ \wedge \ t_2 \in \mathrm{orb}(\alpha) \ .$$

The chain is so eliminated.

Sub-condition (3) can be satisfied by the following algorithm.

Algorithm 3.15 (Multi-path Elimination). *Repeat until there is no variable* Y *such that there is more than one literal of the form* $X\beta \prec \alpha Y$ *or of the form* $\alpha X \prec Y$. *Pick such a* Y *(maximal respect to* $<_\Phi$*).*

1. *For all literals of the form* $X\beta \prec \alpha Y$, *consider two cases:*

   (a) $X \not\equiv Y$.

      i. $|X| < |\alpha|$. *Then* X *is replaced by instantiation.*

      ii. $|X| \geq |\alpha|$. *Then* $X\beta \prec \alpha Y$ *is replaced by*

$$X'\beta \prec Y \ \wedge \ X = \alpha X' \ .$$

   (b) $X \equiv Y$. *It must be the case that* $|\beta| < |\alpha|$. *So* $X\beta \prec \alpha Y$ *becomes*

$$\bigvee_{\{\tilde{\beta} \,|\, |\tilde{\beta}| = |\alpha| - |\beta|\}} X\beta\tilde{\beta} = \alpha' X \ .$$

2. *Now we can assume that Y only exists in $\prec$-literals of the form $t \prec Y$. Consider all chains of the following forms*

$$\begin{aligned} & t_1^{(1)} \prec t_2^{(1)} \prec \cdots \prec t_{n_1}^{(1)} \prec Y \ , \\ & \cdots\cdots\cdots \\ & t_1^{(m)} \prec t_2^{(m)} \prec \cdots \prec t_{n_m}^{(m)} \prec Y \ . \end{aligned}$$

We replace these chains by (guessing) an order completion for terms $t_{n_i}^{(i)}$ and Y.

Now condition (3) of Definition 3.15 can be satisfied by the following algorithm under the assumption that all sub-conditions hold.

Algorithm 3.16 (Alignment of Positive $\prec$ Literals). *Repeat until there are no positive $\prec$-literals of the form $\alpha X\beta \prec Y$ ($\alpha \not\equiv \epsilon_Q$) or of the form $X\beta \prec \alpha Y$ ($\alpha \not\equiv \epsilon_Q$).*

Case $\alpha X\beta \prec Y$. Call Algorithm 3.11.

Case $X\beta \prec \alpha Y$. Call Algorithm 3.12.

Condition (4) of Definition 3.15 can be satisfied by the following algorithm.

Algorithm 3.17 (Chain Normalization). *Repeat for each chain of the form* (3.15)*.*

1. *Simplify $X_i \not\prec X_j$ ($j > i$) to* false*.*

2. *If $\{X_i \in \mathrm{orb}(\alpha, k)\}$ ($i > 0$) occurs in* (3.15)*, then for each $j < i$, guess $X_j \in \mathrm{orb}(\alpha, k_j)$ for some $k_j < |\alpha|$ such that $X_j\alpha_j \in \mathrm{orb}(\alpha)$. If there is no such an arrangement, return* false*. Otherwise, replace $X_j\beta_j \prec X_{j+1}$ by $|X_j| + |\beta_j| < |X_{j+1}|$ for $j < i$.*

Conditions (5) and (6) of Definition 3.15 can be realized by the following algorithm.

Algorithm 3.18 (Alignment with Presence of Orbit Literals). *In the following we assume $|Y| \geq |\alpha|$ and $|X| \geq \max(|\alpha|, |\beta|)$. Otherwise, variables can be removed by instantiation.*

1. *For each subconstraint of the form*

$$\alpha X \neq Y\beta \ \wedge \ X \in \mathrm{orb}(\alpha', k) \ \wedge \ Y \in \mathrm{orb}(\beta', l) \ , \tag{3.6}$$

do the same transformation as in step (3) *in Algorithm 3.5.*

2. *Consider each subconstraint of the form*

$$\alpha X \not\prec Y \ \wedge \ X \in \mathrm{orb}(\alpha', k) \ \wedge \ Y \in \mathrm{orb}(\beta', l) \ , \tag{3.33}$$

which asserts that X is of the form $(\alpha')^*\alpha'[1..k]$ and similar for Y. If α is not a prefix of Y, then $\alpha X \not\prec Y$ simplifies to true. *Otherwise* (3.33) *can be replaced by*

$$X \not\prec Y' \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ \wedge\ Y = \alpha Y' \ ,$$

which can be further reduced to

$$X \not\prec Y' \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ \wedge\ Y' \in \mathrm{orb}(\beta'', l') \ \wedge\ |Y| = |\alpha| + |Y'| \tag{3.34}$$

with β'' and l' the same as in (3.9) *and* (3.10)*, respectively. If $\alpha' = \beta''$, then* (3.34) *is* false *(because $|X'| < |Y'|$) and so is* (3.33)*. If $\alpha' \neq \beta''$, then there are only finitely many cases that $X' \prec Y'$ which can be computed and excluded.*

3. *Consider each subconstraint of the form*

$$X \not\prec \alpha Y \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ , \tag{3.35}$$

which asserts that X is of the form $(\alpha')^*\alpha'[1..k]$ and similar for Y. If α is not a prefix of X, then $X \not\prec \alpha Y$ simplifies to true. *Otherwise* (3.35) *can be replaced by*

$$X' \not\prec Y \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ \wedge\ X = \alpha X' \ ,$$

which can be further reduced to

$$X' \not\prec Y \ \wedge\ X \in \mathrm{orb}(\alpha', k) \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ \wedge\ X' \in \mathrm{orb}(\alpha'', k') \ \wedge\ |X| = |\alpha| + |X'| \ , \tag{3.36}$$

where

$$k' = \Big(l + |\alpha'| - (|\alpha| \bmod |\alpha'|)\Big) \bmod |\alpha'| \ , \tag{3.37}$$

$$\alpha'' = \alpha'[(|\alpha| \bmod |\alpha'|) + 1..|\alpha'|] \circ \alpha'[1..(|\alpha| \bmod |\alpha'|)] \ . \tag{3.38}$$

If $\alpha'' = \beta'$, then (3.36) *is* false *(because $|X'| < |Y|$) and so is* (3.35)*. If $\alpha'' \neq \beta'$, then there are only finitely many cases that $X' \prec Y$ which can be computed and excluded.*

4. *For each subconstraint of the form*

$$\alpha X \neq Y\beta \ \wedge\ Y \in \mathrm{orb}(\beta', l) \ , \tag{3.12}$$

do the same transformation as in step (5) *in Algorithm 3.5.*

5. *Consider each subconstraint of the form*

$$\alpha X \not\approx Y \ \wedge \ Y \in \mathrm{orb}(\beta', l) \ , \tag{3.39}$$

which asserts Y is of the form $(\beta')^*\beta'[1..l]$. If α is not a prefix of Y, then $\alpha X \not\approx Y$ simplifies to true. *Otherwise* (3.39) *can be replaced by*

$$X \not\approx Y' \ \wedge \ Y \in \mathrm{orb}(\beta', l) \ \wedge \ Y = \alpha Y' \ ,$$

which can be further reduced to

$$X \not\approx Y' \ \wedge \ Y \in \mathrm{orb}(\beta', l) \ \wedge \ Y' \in \mathrm{orb}(\beta'', l') \ \wedge \ |Y| = |\alpha| + |Y'| \tag{3.40}$$

with β'' and l' the same as in (3.9) *and* (3.10)*, respectively.*

6. *Consider each subconstraint of the form*

$$X \not\approx \alpha Y \ \wedge \ X \in \mathrm{orb}(\alpha', k) \ , \tag{3.41}$$

which asserts that X is of the form $(\alpha')^*\alpha'[1..k]$. If α is not a prefix of X, then $X \not\approx \alpha Y$ simplifies to true. *Otherwise* (3.41) *can be replaced by*

$$X' \not\approx Y \ \wedge \ X \in \mathrm{orb}(\alpha', k) \ \wedge \ X = \alpha X' \ ,$$

which can be further reduced to

$$X' \not\approx Y \ \wedge \ X \in \mathrm{orb}(\alpha', k) \ \wedge \ X' \in \mathrm{orb}(\alpha'', k') \ \wedge \ |X| = |\alpha| + |X'| \tag{3.42}$$

with k' and α'' the same as in (3.37) *and* (3.38)*, respectively.*

Condition (7) of Definition 3.15 is obtained by the following algorithm.

Algorithm 3.19 (Elimination of Chains with Disequalities). *Repeat until there exist no literals $\alpha X \neq Y\beta$ such that X and Y appear in the same chain.*

1. *$X\beta' \prec Y$ is induced by a chain of the form*

$$X\beta \prec X_0 \ , \quad X_0\beta_0 \prec X_1 \ , \quad \ldots \ , \quad X_n\beta_n \prec Y \ . \tag{3.43}$$

   (a) *If $|\alpha| - |\beta| - |\beta'| \leq 0$, then return* false.

(b) Guess a sequence $\beta'', \beta'_0, \ldots, \beta'_n$ *such that* $|\beta''| > 0$, $|\beta'_i| > 0$ $(i \leq n)$, *and*

$$|\beta''| + |\beta_0| + |\beta'_0| + \cdots + |\beta_n| + |\beta'_n| = |\alpha| - |\beta| - |\beta'| \ .$$

(c) Remove (3.43) *and add*

$$\begin{aligned} X_0 &= X\beta'\beta'' \ , \\ X_{i+1} &= X_i\beta_i\beta'_i \ , \quad (i < n) \\ Y &= X_n\beta_n\beta'_n \ . \end{aligned}$$

2. $Y\beta' \prec X$ *is induced by a chain. Similar to the previous case.*

Condition (8) of Definition 3.15 is obtained by the following algorithm.

Algorithm 3.20 (Elimination of Chains with Negative $\prec$-Literals). *Repeat until* $X \not\prec \alpha Y$ *does not occur when* $Y\beta \prec X$ *is induced by a chain*

$$Y\beta \prec X_0, \quad X_0\beta_0 \prec X_1, \quad \ldots, \quad X_n\beta_n \prec X \ . \tag{3.44}$$

1. *If* $|\alpha| - |\beta| - 1 \leq 0$, *return* false.

2. *Guess a sequence* $\beta', \beta'_0, \ldots, \beta'_n$ *such that* $|\beta'| > 0$, $|\beta'_i| > 0$ $(i \leq n)$, *and*

$$|\beta'| + |\beta_0| + |\beta'_0| + \cdots + |\beta_n| + |\beta'_n| = |\alpha| - |\beta| - 1 \ .$$

3. *Remove* (3.44) *and add*

$$\begin{aligned} X_0 &= Y\beta\beta' \ , \\ X_{i+1} &= X_i\beta_i\beta'_i \ , \quad (i < n) \\ X &= X_n\beta_n\beta'_n \ . \end{aligned}$$

To obtain condition (9) of Definition 3.15, we need to redefine the notations of *delimiter* and *tag*. Let SUF_Φ denote the set of all words β such that $X\beta$ is a proper term in Φ_Q. Let $|\mathrm{SUF}_\Phi|$ be the cardinality of SUF_Φ and let

$$\sup(\mathrm{SUF}_\Phi) = \max\{\, |\alpha| \mid \alpha \in \mathrm{SUF}_\Phi \,\} \ .$$

Definition 3.17 (Delimiters in $\mathfrak{Q}^+$). *A word* q *is called a* delimiter *of* Φ_Q *in* $\mathscr{L}_{\mathfrak{Q}^+}$ *if neither* q *is a subsequence in* $\mathrm{orb}(\alpha)$ *for any* $\alpha \in \mathrm{PRE}_\Phi$ *nor* q *is a subsequence of any* $\beta \in \mathrm{SUF}_\Phi$.

A delimiter always exists for finite SUF_Φ and PRE_Φ. In particular any strongly primitive word longer than $\sup(\mathrm{PRE}_\Phi)$ and $\sup(\mathrm{SUF}_\Phi)$ is a delimiter. In general, the length of a delimiter can be much shorter. Let $n = max\{|\mathrm{PRE}_\Phi|, |\mathrm{SUF}_\Phi|\}$ and $m = max\{\sup(\mathrm{PRE}_\Phi), \sup(\mathrm{SUF}_\Phi)\}$. For a fixed k, there are at most $O(mn)$ words of length k that cannot be delimiter, while there are $2^{O(k)}$ strongly primitive words of length k in a language having more than two atoms.

Recall d_p is a shortest delimiter. In the following d_p always refers to the delimiter for the *current* Φ_Q. Condition (9) of Definition 3.15 is obtained by the following algorithm.

Algorithm 3.21 (Elimination of Negative $\prec$-Literals with Chains). *Choose one of the following two branches.*

1. *For each instance $\alpha X \not\prec Y \wedge X'\beta \prec Y$ that violates conditions* (9) *in Definition 3.15, add to $\theta_{\mathbb{Z}}$*

$$|\alpha| + |X| \geq |X'| + |\beta| + |d_p| \ .$$

   Then return.

2. *Split on an instance $\alpha X \not\prec Y \wedge X'\beta \prec Y$ which violates conditions* (9) *and choose one of the following three choices.*

   (a) *Add $|\alpha| + |X| = |X'| + |\beta|$ to $\theta_{\mathbb{Z}}$ and replace $\alpha X \not\prec Y$ by $\alpha X \neq X'\beta$.*

   (b) *Add $|\alpha| + |X| < |X'| + |\beta|$ to $\theta_{\mathbb{Z}}$ and replace $\alpha X \not\prec Y$ by $\alpha X \not\prec X'\beta$.*

   (c) *Guess β' of length less than $|d_p|$ and replace $\alpha X \not\prec Y$ by $X'\beta\beta' \prec Y \ \wedge \ \alpha X \neq X'\beta\beta'$.*

Put all together we have

Algorithm 3.22 (Normalization in $\mathfrak{Q}^{+}_{\mathbb{Z}}$). *Input: $\Phi_Q \wedge \theta_{\mathbb{Z}}$. Repeatedly call the following algorithms in order until no update to $\Phi_Q \wedge \theta_{\mathbb{Z}}$ can be made.*

1. *Algorithm 3.1 (Normalization in $\mathfrak{Q}$).*
2. *Algorithm 3.4 (Equality Completion).*
3. *Algorithm 3.8 (Prefix Completion).*
4. *Algorithm 3.9 (Elimination of Single Variable Literals).*
5. *Algorithm 3.10 (Simplification of Negative $\prec$-Literals).*
6. *Algorithm 3.13 (Simplification of Positive $\prec$-Literals).*
7. *Algorithm 3.14 ($\prec$-Loop Elimination).*
8. *Algorithm 3.15 ($\prec$-Multipath Elimination).*

9. *Algorithm 3.16 (Alignment of Positive $\prec$-Literals).*
10. *Algorithm 3.17 (Basic Chain Normalization).*
11. *Algorithm 3.18 (Alignment of Negative $\prec$-Literals).*
12. *Algorithm 3.19 (Elimination of Chains with Disequalities).*
13. *Algorithm 3.20 (Elimination of Chains with Negative $\prec$-Literals).*
14. *Algorithm 3.21 (Elimination of Negative $\prec$-Literals with Chains).*

Proposition 3.6 (Normalization in $\mathfrak{Q}^+_{\mathbb{Z}}$). *Algorithm 3.22 preserves equivalence and terminates with Φ_Q in the normal form as given by Definition 3.15.*

Proof. It is easily seen that all transformations in Algorithm 3.22 are sound. We are left to show the termination of Algorithm 3.22. Examining all branches in the algorithm, we see that each branch never generates more $\prec$-literals. Only newly generated equalities and disequalities can destroy properties established by the previous round. Equalities and disequalities are generated at the expense of $\prec$-literals. So each branch eventually stops generating equalities and disequalities. After that each sub-procedure needs to run at most once to obtain the normal form. □

From now on, let us assume $\Phi_Q \wedge \theta_{\mathbb{Z}}$ is in the normal form as in Definition 3.15.

Algorithm 3.23 (Computation of $\Phi_{\Delta+}$ in $\mathfrak{Q}^+_{\mathbb{Z}}$). *Input: $\Phi_Q \wedge \theta_{\mathbb{Z}}$ in normal form. Initially set $\Phi_{\Delta+} := \emptyset$. Add to $\Phi_{\Delta+}$:*

1. $|t_1| = |t_2|$, *if* $t_1 \neq t_2$ *or* $t_1 = t_2$;
2. $|t_1| < |t_2|$, *if* $t_1 \not\prec t_2$ *or* $t_1 \prec t_2$;
3. $|X| + |\alpha| = |\alpha X| = |X\alpha|$, *if* αX *or* $X\alpha$ *occurs;*
4. $|X| \equiv k(\bmod |\alpha|)$, *if* $X \in \mathrm{orb}(\alpha, k)$.

For any variable X, let $\inf(X)$ denote the term t such that $t \prec X$ occurs Φ_Q. Since Φ_Q is in normal form, there exists at most one such t. If there is no such t in (3.15), let $\inf(X)$ be ϵ_Q. We redefine a *tag* as a concatenation of d_p, a color d_c and again d_p. For technical reasons, we require

1. $|d_p| > |d_c|$,
2. $d_c[1] = d_p[1]$ and $d_c[1]$ is not a part of the encoding, and
3. no d_c is equal to any word in SUF_Φ.

The latter two conditions mean $|d_c|$ may be greater than the shortest length for an encoding. Let $L^+_t = 2|d_p| + |d_c|$, the length of a tag. Define $C^+_\Phi(\delta)$ as

$$\bigwedge_{X \in \mathcal{V}_Q(\Phi_Q)} |X| \geq |\inf(X)| + \delta \ .$$

Similar to Theorem 3.2, we show that $|\inf(X)| + L^+_t$ is greater than or equal to the relative cut length of X.

Theorem 3.4 (RLCC in $\mathfrak{Q}^+_{\mathbb{Z}}$). *$\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L^+_t)$ is an RLCC for $\Phi_Q/(\theta_{\mathbb{Z}} \wedge C_\Phi(L^+_t))$.*

Definition 3.18 (Length Configurations in $\mathfrak{Q}^+_{\mathbb{Z}}$). *A* length configuration *(in $\mathfrak{Q}^+_{\mathbb{Z}}$) is either $C_\Phi(L^+_t)$ or $|X| = |\inf(X)| + i$ $(i < L^+_t)$ for some $X \in \mathcal{V}_Q$.*

The set of all configurations, $\mathcal{C}$, *covers* the search space, but does not partition the search space as before. If we take $C_\Phi(L^+_t)$, then by Theorem 3.4 we are done. However, there exists one complication. In the case of $\mathfrak{Q}_{\mathbb{Z}}$, for a length configuration C, the updated version of $\Phi_{\Delta+} \wedge C \wedge \theta_{\mathbb{Z}}$ (after instantiating variables) is an RLCC for the updated version of $\Phi_Q/(\theta_{\mathbb{Z}} \wedge C)$, because instantiation can only reduce PRE_Φ and hence L_t. The case of $\mathfrak{Q}^+_{\mathbb{Z}}$ is complicated as an instantiation in general destroys the normal form by introducing longer β which occurs in terms $Y\beta$. It may also enlarge SUF_Φ and hence L^+_t. Nevertheless, progress can still be made. Suppose $\theta_{\mathbb{Z}}$ contains another

$$C : |X| = |\inf(X)| + i \quad (i < L^+_t) \ .$$

By substitution, we can remove X. If $\inf(X) \equiv \epsilon_Q$, such a substitution is just an instantiation. For example, consider $\mathfrak{Q}^+_{\mathbb{Z}}$ with $\mathcal{A} = \{a, b\}$. $Xa \prec Y \wedge |Y| = |Xa| + 1$ is replaced by either $Y = Xaa$ or $Y = Yab$. Thus we obtain a constraint which has fewer variables than the original. However, a rerun of the normalization is needed for the updated constraints. We summarize the procedure as follows.

Algorithm 3.24 (Decision Procedure for $\mathfrak{Q}^+_{\mathbb{Z}}$). *Input: $\Phi_Q \wedge \theta_{\mathbb{Z}} \wedge \Phi_{\mathbb{Z}}$ where $\theta_{\mathbb{Z}} \equiv \emptyset$.*

1. *If $\Phi_Q \wedge \theta_{\mathbb{Z}} \wedge \Phi_{\mathbb{Z}}$ contains no variables, then*
   (a) *return* SUCCESS *if it is satisfiable;*
   (b) *return* FAIL *otherwise.*
2. *Normalize $\Phi_Q \wedge \theta_{\mathbb{Z}}$ by Algorithm 3.22.*
3. *Return* SUCCESS *if* $\mathrm{PA} \models_\exists C_\Phi(L^+_t) \wedge \Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge \Phi_{\mathbb{Z}}$.
4. *Guess a $C \in \mathcal{C} \backslash \{C_\Phi(L^+_t)\}$ and a substitution compatible with C, and update Φ_Q, $\theta_{\mathbb{Z}}$ and $\Phi_{\mathbb{Z}}$ accordingly.*
5. *Call itself using updated Φ_Q, $\theta_{\mathbb{Z}}$ and $\Phi_{\mathbb{Z}}$.*

Theorem 3.5 (Complexity of $\mathrm{Th}^\forall(\mathfrak{Q}^+_\mathbb{Z})$). *The decision problem for $\mathrm{Th}^\forall(\mathfrak{Q}^+_\mathbb{Z})$ is NP-complete.*

Proof. Clearly $\mathrm{Th}^\forall(\mathfrak{Q}^+_\mathbb{Z})$ is NP-hard. Algorithm 3.23 computes $\Phi_{\Delta+}$ in $O(n)$. The number of subsequences in PRE_Φ or SUF_Φ is bounded by $O(n^2)$. So L^+_t can be computed in $O(n^2)$. By the nondeterministic nature of our algorithms, we can show that each branch of computation in the normalization procedures and in Algorithm 3.24 is in P. Therefore $\mathrm{Th}^\forall(\mathfrak{Q}^+_\mathbb{Z})$ is NP-complete. □

By symmetry, with minor modifications, this algorithm also applies to the theory of queues with suffix relation.

3.6 Decision Procedure for $\mathrm{Th}(\mathfrak{Q}_\mathbb{Z})$

In this section we present a quantifier elimination for the first-order theory of queues with integers, $\mathrm{Th}(\mathfrak{Q}_\mathbb{Z})$. The procedure removes a block of quantifiers of the same type in a single step.

Since the extended theory has three sorts, namely, atoms, integers, and queues, we need to show elimination of quantifiers of all three sorts. The elimination procedure consists of the following three sub-procedures.

Elimination of Quantifiers on Atom Variables We assume formulae with quantifiers on atom variables are in the form

$$(\exists \bar{\mathbf{x}} : \mathcal{A}) \Big[\ \Phi_\mathcal{A}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{X}}) \ \wedge\ \Phi_Q(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{X}}) \ \wedge\ \Phi_\mathbb{Z}(\bar{\mathbf{u}}, \bar{\mathbf{X}}) \ \Big] \ , \tag{3.45}$$

where $\bar{\mathbf{X}} \subseteq \mathcal{V}_Q$, $\bar{\mathbf{x}}, \bar{\mathbf{y}} \subseteq \mathcal{V}_\mathcal{A}$ and $\bar{\mathbf{u}} \subseteq \mathcal{V}_\mathbb{Z}$. Note that $\bar{\mathbf{X}}$ occur in $\Phi_\mathcal{A}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{X}})$ by selector functions and occur in $\Phi_\mathbb{Z}(\bar{\mathbf{u}}, \bar{\mathbf{X}})$ by the length function. As the atom domain is finite, all such quantifiers can be eliminated by instantiation, i. e., (3.45) is equivalent to

$$\bigvee_{\bar{\mathbf{a}} \in \mathcal{A}^*} \Phi_\mathcal{A}(\bar{\mathbf{a}}, \bar{\mathbf{y}}, \bar{\mathbf{X}}) \ \wedge\ \Phi_Q(\bar{\mathbf{a}}, \bar{\mathbf{y}}, \bar{\mathbf{X}}) \ \wedge\ \Phi_\mathbb{Z}(\bar{\mathbf{u}}, \bar{\mathbf{X}}) \ .$$

Elimination of Quantifiers on Integer Variables We assume formulas with quantifiers on integer variables are in the form

$$(\exists \bar{\mathbf{u}} : \mathbb{Z}) \Big[\ \Phi_Q(\bar{\mathbf{X}}) \ \wedge\ \Phi_\mathbb{Z}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{X}}) \ \Big] \ , \tag{3.46}$$

where $\bar{\mathbf{X}} \subseteq \mathcal{V}_Q$ and $\bar{\mathbf{v}}, \bar{\mathbf{u}} \subseteq \mathcal{V}_\mathbb{Z}$. Since $\Phi_Q(\bar{\mathbf{X}})$ does not contain $\bar{\mathbf{u}}$, we can move them out of the scope of $(\exists \bar{\mathbf{u}})$, and then obtain

$$\Phi_Q(\bar{\mathbf{X}}) \ \wedge\ (\exists \bar{\mathbf{u}} : \mathbb{Z})\, \Phi_\mathbb{Z}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{X}}) \ . \tag{3.47}$$

Since in $\Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{X}})$, $\bar{\mathbf{X}}$ occur as pseudo integer variables, $(\exists \bar{\mathbf{u}} : \mathbb{Z})\Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{X}})$ is essentially a Presburger formula and we can proceed to remove the quantifier using Cooper's method [12]. In fact we can defer the elimination until all other types of quantifiers are removed.

Elimination of Quantifiers on Queue Variables We assume formulas with quantifiers on queue variables are in the form

$$(\exists \bar{\mathbf{X}} : Q)\left[\ \Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \ \wedge\ \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{X}}, \bar{\mathbf{Y}})\ \right], \tag{3.48}$$

where $\bar{\mathbf{X}}, \bar{\mathbf{Y}} \subseteq \mathcal{V}_Q$, $\bar{\mathbf{u}} \subseteq \mathcal{V}_{\mathbb{Z}}$, and $\Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{X}}, \bar{\mathbf{Y}})$ can be an arbitrary Presburger formula (not necessarily quantifier-free). By Proposition 3.1, we can assume $\bar{\mathbf{X}}$ does not occur in selectors. Though elimination of selectors in general adds more existential quantifiers of sort queue or atom, the newly added quantifiers will be removed together with the original ones.

We can assume $s_0(s_1(\ldots(s_n(Y)\ldots))$ is *proper*, i. e., $|Y| > n$ is in $\Phi_{\mathbb{Z}}$. We need to extend the notion of RLCC to deal with parameters.

Definition 3.19 (RLCC with Parameters in $\mathfrak{Q}_{\mathbb{Z}}$). *Consider* $(\exists \bar{\mathbf{X}}{:}Q)\,[\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}})]$, *where* $\bar{\mathbf{Y}}$ *are parameters. Let* $\Phi_Q^{(2)}(\bar{\mathbf{Y}})$ *be the maximum subset of* $\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}})$ *not containing* $\bar{\mathbf{X}}$ *and* $\Phi_Q^{(1)}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) := \Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \setminus \Phi_Q^{(2)}(\bar{\mathbf{Y}})$. *A formula* $\Phi_\Delta(\bar{\mathbf{X}}, \bar{\mathbf{Y}})$ *is an RLCC in* $\bar{\mathbf{X}}$ *for* $\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}})$ *relativized to* $\theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}})$, *(in short,* $\Phi_\Delta(\bar{\mathbf{X}}, \bar{\mathbf{Y}})$ *is an RLCC for* $\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}})/\bar{\mathbf{X}}/\theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}})$*), if the following hold:*

$$(\forall \bar{\mathbf{X}}, \bar{\mathbf{Y}} : Q)\left[\ \Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \ \wedge\ \theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \ \rightarrow\ (\exists \bar{\mathbf{z}} : \mathbb{Z})\left(\Phi_\Delta(\bar{\mathbf{z}}, \bar{\mathbf{Y}}) \ \wedge\ |\bar{\mathbf{X}}| = \bar{\mathbf{z}}\right)\ \right], \tag{3.49}$$

$$(\forall \bar{\mathbf{Y}} : Q)(\forall \bar{\mathbf{z}} : \mathbb{Z})\left[\ \Phi_Q^{(2)}(\bar{\mathbf{Y}}) \ \wedge\ \Phi_\Delta(\bar{\mathbf{z}}, \bar{\mathbf{Y}}) \ \rightarrow\ (\exists \bar{\mathbf{X}} : Q)\left(\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \ \wedge\ \theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \ \wedge\ |\bar{\mathbf{X}}| = \bar{\mathbf{z}}\right)\ \right]. \tag{3.50}$$

We also need to update the notion of normal form for parameters.

Definition 3.20 (Normal Form with Parameters in $\mathfrak{Q}_{\mathbb{Z}}$). Φ_Q *is in* normal form *in* $\mathfrak{Q}_{\mathbb{Z}}$ *(with parameters) if* Φ_Q *satisfies Definition 3.10 and the following condition: if* $\alpha X \beta \neq t(Y)$ *(where* Y *is a parameter) appears in* Φ_Q*, then* $\alpha \equiv \beta \equiv \epsilon_Q$ *and* X *does not occur in literals of the form* $X \in \mathrm{orb}(\alpha, k)$.

3.6.1 Normalization in $\mathfrak{Q}_{\mathbb{Z}}$ with Parameters $\bar{\mathbf{Y}}$

Algorithm 3.5 has taken care of literals without parameters and we do not need to touch literals built up only by parameters. Thus, we are left with literals of the form $t(X) = t'(Y)$ or $t(X) \neq t'(Y)$.

Algorithm 3.25 (Normalization with Parameters in $\mathfrak{Q}_{\mathbb{Z}}$).

1. *Call Algorithm 3.5 to process literals without parameters.*

2. *Pick a literal and consider four cases:*

(a) $\alpha X = Y\beta$: *if* $|X| < |\beta|$ *we remove* X *by an instantiation; otherwise we replace* $\alpha X = Y\beta$ *by*

$$X = \mathrm{lt}^{|\alpha|}(Y) \circ \beta \ \wedge \ \bigwedge_{i=1}^{|\alpha|} \alpha[i] = \mathrm{lh}(\mathrm{lt}^{i-1}(Y)) \ ,$$

where lt^n *denotes* n *applications of* lt.

(b) $X\alpha = \beta Y$: *Similar to (a), if* $|X| < |\beta|$ *we remove* X *by an instantiation; otherwise we replace* $X\alpha = \beta Y$ *by*

$$X = \beta \circ \mathrm{rt}^{|\alpha|}(Y) \ \wedge \ \bigwedge_{i=1}^{|\alpha|} \alpha[i] = \mathrm{rh}(\mathrm{rt}^{i-1}(Y)) \ ,$$

where rt^n *denotes* n *applications of* rt.

(c) $\alpha X \neq Y\beta$: *negation of (a).*

(d) $X\alpha \neq \beta Y$: *negation of (b).*

In all cases, if Y *has been decomposed, i. e., newly generated selector terms contain* Y*, we express* Y *in concatenation notation and use it to replace other occurrences. For example, if* $\mathrm{lh}(\mathrm{lt}^{i-1}(Y))$ *occurs, we rewrite other occurrences of* Y *by*

$$\mathrm{lh}(Y) \circ \cdots \circ \mathrm{lh}(\mathrm{lt}^{j}(Y)) \circ \cdots \circ \mathrm{lh}(\mathrm{lt}^{i-1}(Y)) \circ \mathrm{lt}^{i}(Y) \ . \tag{3.51}$$

3. *Replace conjunctions of the form*

$$X \in \mathrm{orb}(\alpha, k) \ \wedge \ \bigwedge_{i} X \neq t_i(Y_i) \ ,$$

where $t_i(Y_i)$*'s are terms containing parameters, by*

$$X \in \mathrm{orb}(\alpha, k) \ \wedge \ \bigwedge_{i} t_i(Y_i) \notin \mathrm{orb}(\alpha, k) \ .$$

The equivalence of this transformation is easily seen under the side condition $|X| = |t_i(Y_i)|$.

We treat Q-terms of the form $t(Y)$ as distinct variables. Let L_c,L_p and L_t be as defined in Section 3.4 and we obtain $C_\Phi(L_t)$ and $\Phi_{\Delta+}(\bar{\mathbf{X}}, \bar{\mathbf{Y}})$ accordingly.

Theorem 3.6 (RLCC with Parameters in $\mathfrak{Q}_\mathbb{Z}$). *$\Phi_{\Delta+}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge \theta_\mathbb{Z}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge C_\Phi(L_t)$ is an RLCC for $\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}})/\bar{\mathbf{X}}/(\theta_\mathbb{Z}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge C_\Phi(L_t))$.*

Algorithm 3.26 (Elimination of Queue Quantifiers).

1. *Guess and add a* $C \in \mathcal{C}$ *to* (3.48).

2. *Instantiate each X such that* $|X| = i$ $(i < L_t)$ *occurs in C, and update C,* Φ_Q, $\theta_{\mathbb{Z}}$ *and* $\Phi_{\mathbb{Z}}$ *accordingly.*

3. *Rewrite* (3.48) *to*

$$(\exists \bar{\mathbf{X}} : Q)\Big[\, \Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \;\wedge\; C \;\wedge\; \theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \;\wedge\; \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{X}}, \bar{\mathbf{Y}}) \,\Big] \;, \tag{3.52}$$

which is equivalent to

$$\Phi_Q^{(2)}(\bar{\mathbf{Y}}) \;\wedge\; (\exists \bar{\mathbf{v}} : \mathbb{Z})\Big[\, \Phi_{\Delta+}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \;\wedge\; C \;\wedge\; \theta_{\mathbb{Z}}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \;\wedge\; \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{Y}}) \,\Big] \;. \tag{3.53}$$

Theorem 3.7 (Soundness of Algorithm 3.26). *Algorithm 3.26 terminates and preserves equivalence.*

3.7 Conclusion

We presented decision procedures for the theory of queues with integer constraints. Our method combines the extraction of integer constraints from queue constraints, and in case of the quantified theory, with a reduction of quantifiers on queue variables to quantifiers on integer variables.

We plan to extend our results to theories with more expressive power, e.g., the theory of concatenation [35] and the theory of queues with subsequence relations (including subqueue, prefix and suffix relation) [5]. We also plan to investigate decidability of the first-order theory of queues with integers and prefix or suffix relation. Note that for the first-order theory we cannot obtain decidability in a language with both prefix and suffix relations, nor in a language with prefix relation (resp. suffix relation) and the right insertion constructor (resp. the left insertion constructor), because all extensions are sufficiently expressive to interpret the first-order theory of arrays, which is well-known to be undecidable.

3.8 Proofs

In this section we present the proofs not included in the main text of this chapter. For ease of reference we restate the propositions and theorems.

Proposition 3.1 (Elimination of Selectors). For any φ in $\mathscr{L}_Q$, one can effectively compute an equivalent φ' such that

1. φ' contains no selectors, and
2. if φ is quantifier-free, then φ' can be put into either $\exists_1$ or $\forall_1$ form.

Proof. Below we use constructors to define selectors lh and lt in the $\exists_1$-fragment.

$$\begin{aligned}
\mathrm{lh}(X) = y &\quad\leftrightarrow\quad (\exists Y \in Q)(X = yY) \ , \\
\mathrm{lt}(X) = Y &\quad\leftrightarrow\quad (\exists x \in \mathcal{A})(X = xY) \ , \\
\mathrm{lh}(X) \neq y &\quad\leftrightarrow\quad (\exists Y \in Q)(\exists y' \in \mathcal{A})\left(X = y'Y \ \wedge \ y' \neq y\right) \ , \\
\mathrm{lt}(X) \neq Y &\quad\leftrightarrow\quad (\exists Y' \in Q)(\exists y \in \mathcal{A})\left(X = yY' \ \wedge \ Y' \neq Y\right) \ .
\end{aligned}$$

We can also define them in the $\forall_1$-fragment as follows.

$$\begin{aligned}
\mathrm{lh}(X) = y &\quad\leftrightarrow\quad (\forall Y \in Q)(\forall y' \in \mathcal{A})\left(y' \neq y \ \rightarrow \ X \neq y'Y\right) \ , \\
\mathrm{lt}(X) = Y &\quad\leftrightarrow\quad (\forall Y' \in Q)(\forall y \in \mathcal{A})\left(Y \neq Y' \ \rightarrow \ X \neq yY'\right) \ , \\
\mathrm{lh}(X) \neq y &\quad\leftrightarrow\quad (\forall Y \in Q)(X \neq yY) \ , \\
\mathrm{lt}(X) \neq Y &\quad\leftrightarrow\quad (\forall x \in \mathcal{A})(X \neq xY) \ .
\end{aligned}$$

Similar definitions for rh and rt. □

Proposition 3.4 (Normal Form in $\mathfrak{Q}$). A constraint in normal form is satisfiable.

Proof. Let Φ_Q be a constraint in normal form and $\mathcal{V}_Q(\Phi_Q) = \{X_0, \ldots, X_n\}$. We build a satisfying assignment for Φ_Q as follows. Beginning with X_0, set $[\![X_0]\!]$ to an arbitrary word if neither $X_0 \in \mathrm{orb}(\alpha, k)$ nor $X_0 \notin \mathrm{orb}(\alpha', k')$ occurs. If $X_0 \in \mathrm{orb}(\alpha, k)$ is present, set $[\![X_0]\!]$ to a word in $\mathrm{orb}(\alpha, k)$. If $\bigwedge_{0<i\leq n} X_0 \notin \mathrm{orb}(\alpha_i, k_i)$ is present, set $[\![X_0]\!]$ to a word in $\bigcap_{0<i\leq n}(\sim \mathrm{orb}(\alpha_i, k_i))$ which is infinite. The partial assignment $[\![X_0]\!]$ is obviously satisfiable. Assuming $X_0, \ldots, X_i$ have been assigned such that $[\![X_0, \ldots, X_i]\!]$ is a satisfying partial assignment. Let $\mathcal{T}(X_0, \ldots, X_i)$ denote the set of all terms in Φ_Q built up using $X_0, \ldots, X_i$ and $[\![\mathcal{T}(X_0, \ldots, X_i)]\!]$ the corresponding set of assigned values. Obviously, $[\![\mathcal{T}(X_0, \ldots, X_i)]\!]$ is finite. Choose $[\![X_{i+1}]\!]$ to be a word which is longer than any words in $[\![\mathcal{T}(X_0, \ldots, X_i)]\!]$. This can always be done even if $X_{i+1} \in \mathrm{orb}(\alpha, k)$ or $X_{i+1} \notin \mathrm{orb}(\alpha', k')$ occurs, because both $\mathrm{orb}(\alpha, k)$ and $\bigcap_{0<i\leq n}(\sim \mathrm{orb}(\alpha_i, k_i))$ are infinite. It follows that any word in $[\![\mathcal{T}(X_{i+1})]\!]$

is longer than words in $[\![\mathcal{T}(X_0, \ldots, X_i)]\!]$, and so disequalities between terms in $\mathcal{T}(X_0, \ldots, X_i, X_{i+1})$ are respected, that is, $[\![X_0, \ldots, X_{i+1}]\!]$ is a satisfying partial assignment of Φ_Q. By induction, Φ_Q is satisfiable. □

Proposition 3.5 (Normal Form in $\mathfrak{Q}_{\mathbb{Z}}$). Algorithm 3.5 preserves equivalence and terminates with Φ_Q in the normal form as given by Definition 3.10.

Proof. Soundness is clear from the description of the algorithm. We now show termination. The only problematic case is step (4) where we generate $Y = \alpha'' Y'$. If there exists a disequality $Y \neq Z\gamma$, then the substitution will produce $\alpha'' Y' \neq Z\gamma$. However, there are no literals of the form $Y' \in \text{orb}(\beta', l')$. Otherwise, since Y' is fresh, there must be literals of the form $Y \in \text{orb}(\beta, l)$. Then we should have had the first case (step (3)) as X already appears in $X \in \text{orb}(\alpha', k)$. Therefore, $\alpha'' Y' \neq Z\gamma$ can only trigger the third case (step (5)) when there exists a literal of the form $Z \in \text{orb}(\gamma', m)$. To be in line with the notation in the algorithm description, let us rename α'', Y', Z, γ, γ' and m respectively, by α, X, Y, β, β' and l. Step (5) transforms $\alpha X \neq Y\beta \wedge Y \in \text{orb}(\beta', l)$ to

$$X \neq Y'\beta \ \wedge \ Y' \in \text{orb}(\beta'', l') \ \wedge \ Y \in \text{orb}(\beta', l) \ \wedge \ |Y| = |\alpha| + |Y'| \ , \tag{3.13}$$

where β'' and l' are the same as in (3.9) and (3.10) respectively. Here we introduce a new variable Y' but no new term with a non-empty prefix. Termination follows. □

Proposition 3.7 (Tags in $\alpha\beta$). *If β is tagged by q, then $\alpha\beta$ is not, where $\alpha \in \text{PRE}_\Phi$.*

Proof. Let $q = d_p d_c$ where d_p is the delimiter and d_c is a color. If $\alpha\beta$ is tagged by q, then $|d_q| > |\alpha|$ since d_q can not be a prefix of α by the definition. This implies that $d_p = \alpha \circ \beta[1..|d_p| - |\alpha|]$. On the other hand, we have $d_p = \beta[1..|d_p|]$ as β is tagged by q. Put together, $\alpha \circ \beta[1..|d_p| - |\alpha|] = \beta[1..|d_p|]$ implies

$$d_p[1..|d_p| - |\alpha|] = d_p[|\alpha| + 1..|d_p|] \ ,$$

which means that d_p is not strongly primitive, contradicting the definition of delimiter. □

Theorem 3.2 (RLCC in $\mathfrak{Q}_{\mathbb{Z}}$). $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L_t)$ is an RLCC for $\Phi_Q / (\theta_{\mathbb{Z}} \wedge C_\Phi(L_t))$.

Proof. We need to show the validity of (3.4) and (3.5) with

$$\theta_{\mathbb{Z}}(\bar{\mathbf{X}}) := \theta_{\mathbb{Z}}(\bar{\mathbf{X}}) \ \wedge \ C_\Phi(L_t) \ ,$$
$$\Phi_\Delta := \Phi_{\Delta+} \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{X}}) \ \wedge \ C_\Phi(L_t) \ .$$

The soundness is immediate. Now let $\sigma_{\mathbb{Z}} = \{|X_i| := l_i \mid i < n\}$ be a satisfying assignment for $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{X}}) \wedge C_\Phi(L_t)$. We need to find a solution σ_Q for Φ_Q such that $|[\![X_i]\!]\sigma_Q| = l_i$ for $i < n$. Since

Φ_Q contains no equalities, it suffices to show σ_Q satisfying literals of the following forms

$$\alpha X \neq Y\beta \ , \tag{3.54}$$

$$X \in \text{orb}(\alpha, k) \ , \tag{3.55}$$

$$X \notin \text{orb}(\alpha, k) \ . \tag{3.56}$$

Let us assign each distinct queue variable or proper constant queue a unique color. Let c_X denote the color assigned to variable X and c_α the color assigned to α. For each variable X, we define $[\![X]\!]\sigma_Q$ as follows.

1. If $X \in \text{orb}(\alpha, k)$ occurs in Φ_Q, then

$$[\![X]\!]\sigma_Q := \text{ext}(\alpha, m, k) \ , \tag{3.57}$$

where $m * |\alpha| + k = [\![|X|]\!]\sigma_{\mathbb{Z}}$. This definition is sound as

(a) $\Phi_{\Delta+}$ contains $|X| \equiv k(\bmod |\alpha|)$ (Algorithm 3.6), and

(b) X appears in at most one literal of the form $X \in \text{orb}(\alpha, k)$ (Algorithm 3.1).

2. If $X \in \text{orb}(\alpha, k)$ does not occur in Φ_Q, then

$$[\![X]\!]\sigma_Q := d_p \circ c_X \circ p_X \ , \tag{3.58}$$

where p_X denotes an arbitrary padding to make $[\![X]\!]\sigma_Q$ has length $[\![|X|]\!]\sigma_{\mathbb{Z}}$.

Clearly by the definition $|[\![X]\!]\sigma_Q| = [\![|X|]\!]\sigma_{\mathbb{Z}}$. So we are left to show that σ_Q is satisfying. For notation simplicity, we drop σ_Q and use $[\![t]\!]$ to denote the value assigned to term t under σ_Q. We first analyze four cases to show literals of the form $\alpha X \neq Y\beta$ are satisfied.

1. $\alpha X \neq Y\beta$ occurs with both $X \in \text{orb}(\alpha', k)$ and $Y \in \text{orb}(\beta', l)$. This can not happen as step (3) of Algorithm 3.5 would have replaced X, Y by instantiation.

2. $\alpha X \neq Y\beta$ occurs with $X \in \text{orb}(\alpha', k)$ but not with $Y \in \text{orb}(\beta', l)$. Then $\alpha \equiv \epsilon_Q$ (thanks to Algorithm 3.5) and Y is assigned a tag as in (3.58). Thus, $[\![X]\!] \neq [\![Y\beta]\!]$ as $|d_p| \leq |[\![X]\!]|$ and $d_p \notin \text{orb}(\alpha')$.

3. $\alpha X \neq Y\beta$ occurs with $Y \in \text{orb}(\beta', l)$ but not with $X \in \text{orb}(\alpha', k)$. Then $\alpha \equiv \epsilon_Q$ (again thanks to Algorithm 3.5) and X is assigned a tag as in (3.58). Thus, $[\![X]\!] \neq [\![Y\beta]\!]$ as $|d_p| \leq |[\![Y]\!]|$ and $d_p \notin \text{orb}(\beta')$.

4. $\alpha X \neq Y\beta$ occurs with neither $X \in \text{orb}(\alpha', k)$ nor $Y \in \text{orb}(\beta', l)$. Then both $[\![X]\!]$ and $[\![Y]\!]$ begin with a tag. If $\alpha \equiv \epsilon_Q$, then $[\![X]\!] \neq [\![Y\beta]\!]$ because $c_X \neq c_Y$. If $\alpha \not\equiv \epsilon_Q$, then by Proposition 3.7, $[\![\alpha X]\!]$ is not tagged while $[\![Y\beta]\!]$ is tagged by $d_p \circ c_Y$. Hence $[\![\alpha X]\!] \neq [\![Y\beta]\!]$.

For literals of the form $X \in \text{orb}(\alpha, k)$, $[\![X]\!]$ must be defined according to (3.57), and hence $X \in \text{orb}(\alpha, k)$ is clearly satisfied. For literals of the form $X \notin \text{orb}(\alpha, k)$, $[\![X]\!]$ must be defined according to (3.58) as by Algorithm 3.1, X does not appear in literals of the form $X \in \text{orb}(\alpha, k)$. Since $d_p \notin \text{orb}(\alpha)$, we have $[\![X]\!] \notin \text{orb}(\alpha, k)$.

Thus, we can find a solution σ_Q for Φ_Q such that $|[\![X_i]\!]| = l_i$ for $i < n$, provided that $\sigma_{\mathbb{Z}} = \{|X_i| : l_i \mid i < n\}$ satisfy $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L_t)$. Therefore, $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L_t)$ is realizable and hence it is an RLCC for $\Phi_Q/(\theta_{\mathbb{Z}} \wedge C_\Phi(L_t))$. □

Theorem 3.4 (RLCC in $\mathfrak{Q}^+_{\mathbb{Z}}$). $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L^+_t)$ is an RLCC for $\Phi_Q/(\theta_{\mathbb{Z}} \wedge C_\Phi(L^+_t))$.

Proof. As in the proof of Theorem 3.2, soundness is immediate. Now let $\sigma_{\mathbb{Z}} = \{|X_i| := l_i \mid i < n\}$ be a satisfying assignment for $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{X}}) \wedge C_\Phi(L^+_t)$. We need to find a solution σ_Q for Φ_Q such that $|[\![X_i]\!]\sigma_Q| = l_i$ for $i < n$. It suffices to show that σ_Q satisfies literals like (3.54)-(3.56) and literals of the forms

$$X\beta \prec Y \ , \tag{3.59}$$

$$\alpha X \not\prec Y \ , \tag{3.60}$$

$$X \not\prec \alpha Y \ . \tag{3.61}$$

Recall Φ_Q is in normal form and literals of the form $X\alpha \prec Y$ are arranged in a tree structure (assuming ϵ_Q is the common root). We build an satisfying assignment incrementally. For a variable X, if $\inf(X) \equiv \epsilon_Q$, i.e. X is in the head of a chain (ignoring the root ϵ_Q), then we assign X as in the proof of Theorem 3.2. Here we require the padding p_X be in the form a^* where $a \neq d_p[1]$. If $\inf(X) \not\equiv \epsilon_Q$ and $[\![\inf(X)]\!]$ is determined, we assign

$$[\![X]\!] := [\![\inf(X)]\!] \circ d_p \circ c_X \circ d_p \circ p_X \ .$$

Let us denote this assignment by σ. We need the following propositions for the rest of the proof.

Proposition 3.8 (Tags in Chains). *Given a chain of the form*

$$X_0\beta_0 \prec X_1 \ , \quad X_1\beta_1 \prec X_2 \ , \quad \ldots \ , \quad X_n\beta_n \prec X_{n+1} \ . \tag{3.15}$$

1. *If X_0 does not occur in any positive orbit literals, then $[\![X_i]\!]$ ($i \leq n+1$) contains exactly $i+1$ non-overlapping tags, namely $t_{X_0}, \ldots, t_{X_i}$ where for $j \leq i$, t_{X_j} is the tag for X_j beginning after $[\![\inf(X_j)]\!]$.*
2. *If X_0 occur in any positive orbit literals, then $[\![X_i]\!]$ contains exactly i unique and non-overlapping tags, namely $t_{X_1}, \ldots, t_{X_i}$.*

Proof. Let us first assume that X_0 does not occur in any positive orbit literals. Suppose there exists a tag t_X in $[\![X_i]\!]$ such that t_X does not coincide with any of $t_{X_0}, \ldots, t_{X_i}$. Let t_X be $d^1_p \circ c_X \circ d^2_p$ where

$d_p^1 \equiv d_p^2 \equiv d_p$ with the superscripts denoting the different occurrences. It is clear that d_p^1 must appear in a fragment of the form

$$d_p^{(1)} \circ c_{X_j} \circ d_p^{(2)} \circ p_{X_j} \circ \beta_j \ , \tag{3.62}$$

for $j < i$ or in the fragment

$$d_p^{(1)} \circ c_{X_i} \circ d_p^{(2)} \circ p_{X_j} \ , \tag{3.63}$$

where $d_p^{(1)} \circ c_{X_j} \circ d_p^{(2)}$ is t_{X_j} and $d_p^{(1)} \equiv d_p^{(2)} \equiv d_p$ (again with the superscripts denoting the different occurrences). Let us first consider overlapping of the form (3.62) for $j < i$.

1. d_p^1 cannot begin properly in $d_p^{(1)}$, i. e., $d_p^1[1] = d_p^{(1)}[i]$ for $i > 1$, as otherwise d_p is not strongly primitive.
2. Similarly, d_p^1 cannot begin properly in $d_p^{(2)}$.
3. d_p^1 can not begin in c_{X_j} because $|d_p| > |c_{X_j}|$ and hence d_p^1 overlaps with $d_p^{(2)}$, again implying d_p is not strongly primitive.
4. d_p^1 cannot begin in β_j. Suppose otherwise, then either d_p^1 overlaps with the beginning delimiter of $[\![X_{j+1}]\!]$ or d_p^1 is a subsequence of β_j. Both lead to contradictions.
5. d_p^1 cannot begin in p_{X_j} as by definition p_{X_j} only contains a letter a and $a \neq d_p[1]$.
6. d_p^1 cannot be in the same position as $d_p^{(1)}$ because then t_X coincides with t_{X_j}.
7. The only possibility left is that d_p^1 occurs in the same position as $d_p^{(2)}$. Consider two cases.
   (a) If $p_{X_j} \not\equiv \epsilon_Q$, then p_{X_j} is not a prefix of any color d_c. because $p_{X_j}[1] \neq d_c[1]$. Therefore, t_X is an invalid tag.
   (b) Suppose $p_{X_j} \equiv \epsilon_Q$. If $|c_X| = |\beta_j|$, then the overlapping cannot happen because $c_X \neq \beta_j$. If $|c_X| < |\beta_j|$, then d_p^2 overlaps with the beginning delimiter of X_{j+1}, a contradiction. If $|c_X| > |\beta_j|$, then d_p^2 still overlaps with the beginning delimiter of X_{j+1} because for any color d_c, $|d_p| > |d_c|$ and hence $|c_X| < |\beta_j| + |d_p|$.

The argument for overlapping of the form (3.63) is similar, except that we do not need to worry about cases involving β_j.

Now suppose that X_0 does not appear in positive orbit literals. We only need to show that d_p^1 does not appear in $[\![X_0\beta_0]\!]$.

1. Suppose d_p^1 begins in β_0. Then either d_p^1 overlaps with the beginning delimiter of $[\![X_1]\!]$ or d_p^1 is a subsequence of β_0. Both lead to contradictions. For the same reason d_p^2 cannot begin in β_0 either.

2. Suppose d_p^1 begins in $[\![X_0]\!] \in \text{orb}(\alpha, k)$. Then d_p^1 cannot end in $[\![X_0]\!] \in \text{orb}(\alpha, k)$, since otherwise d_p^1 is a subsequence in $\text{orb}(\alpha, k)$. So d_p^1 must end in β_0 or otherwise it overlaps with the beginning delimiter of $[\![X_1]\!]$. Because $|c_X| < |d_p|$, d_p^2 has to begin in the beginning delimiter of $[\![X_1]\!]$, a contradiction.

Note that we in fact showed that $[\![X_i\beta_i]\!]$ $(i < n+1)$ contains i non-overlapping tags (in case that $X_0 \in \text{orb}(\alpha, k)$ occurs) and $i+1$ non-overlapping tags otherwise. □

Proposition 3.9 (Tags in αX). *For any variable X occurring in $X \in \text{orb}(\alpha', k')$, $[\![\alpha X]\!]$ does not contain any tags.*

Proof. Suppose there is a tag $d_p^1 \circ t_X \circ d_p^2$ occurring in $[\![\alpha X]\!]$. d_p^1 cannot be a subsequence of α because otherwise d_p^1 is a subsequence in $\text{orb}(\alpha)$. This forces d_p^2 to be a subsequence of $[\![X]\!]$, a contradiction, because d_p^2 is a subsequence of $\text{orb}(\alpha')$. □

Let us resume the proof. We consider all combinations for literals of the forms (3.54)-(3.56) and (3.59)-(3.61).

1. $X\beta < Y$. Literals of this type form a tree structure. In addition, If X appears in a positive orbit literal, then X should be the head of a chain (ignoring the root ϵ_Q). That also means Y cannot appear in a positive orbit literal. So by the inductive definition, we have $[\![X\beta]\!] \prec [\![Y]\!]$.

2. $X \in \text{orb}(\alpha, k)$. Then X must be the head of a chain (which may be just itself). In addition, neither there is an occurrence of $X \in \text{orb}(\alpha', k')$ ($k \not\equiv k'$ or $\alpha \not\equiv \alpha'$) nor there is an occurrence of $X \notin \text{orb}(\alpha'', k'')$. So by definition we have $[\![X]\!] \in \text{orb}(\alpha, k)$.

3. $X \notin \text{orb}(\alpha, k)$. Then there is no occurrence of $X \in \text{orb}(\alpha', k')$. So $[\![X]\!]$ begins with $d_p \circ c_X \circ d_p$. As d_p cannot be a subsequence in $\text{orb}(\alpha)$, we have $[\![X]\!] \notin \text{orb}(\alpha, k)$.

4. $\alpha X \neq Y\beta$. There are four cases.

   (a) $\alpha X \neq Y\beta$ occurs with both $X \in \text{orb}(\alpha', k)$ and $Y \in \text{orb}(\beta', l)$. This could not happen as step (1) of Algorithm 3.18 would have replaced X, Y by instantiation.

   (b) $\alpha X \neq Y\beta$ occurs with $X \in \text{orb}(\alpha', k)$ but not with $Y \in \text{orb}(\beta', l)$. Then Y must be in a chain whose head variable is not itself. By Proposition 3.8 $[\![Y\beta]\!]$ contains the tag t_Y while by Proposition 3.9 $[\![\alpha X]\!]$ does not contain any tags. So $[\![\alpha X]\!] \neq [\![Y\beta]\!]$.

   (c) $\alpha X \neq Y\beta$ occurs with $Y \in \text{orb}(\beta', l)$ but not with $X \in \text{orb}(\alpha', k)$. Then by the second part of Proposition 3.8, $[\![Y\beta]\!]$ does not contain any tags while $[\![\alpha X]\!]$ contains the tag t_X, and so $[\![\alpha X]\!] \neq [\![Y\beta]\!]$.

   (d) $\alpha X \neq Y\beta$ occurs with neither $X \in \text{orb}(\alpha', k)$ nor $Y \in \text{orb}(\beta', l)$. Then by condition 7 in Definition 3.15, X and Y are not in the same chain. Therefore, $[\![\alpha X]\!]$ contains the tag t_X while $[\![Y\beta]\!]$ does not. So $[\![\alpha X]\!] \neq [\![Y\beta]\!]$.

5. $X \not\prec \alpha Y$.

   (a) $X \not\prec \alpha Y$ occurs with both $X \in \text{orb}(\alpha', k)$ and $Y \in \text{orb}(\beta', l)$. This could not happen as step (3) of Algorithm 3.18 would have replaced X, Y by instantiation.

   (b) $X \not\prec \alpha Y$ occurs with $X \in \text{orb}(\alpha', k)$ but not with $Y \in \text{orb}(\beta', l)$. Then by condition (5c) in Definition 3.15, $\alpha \equiv \epsilon_Q$. By condition (4b) in Definition 3.15, $X\beta \prec Y$ is not implied by any chain. Neither is $Y\beta \prec X$ due to prefix completeness. Hence X and Y are not in the same chain. If Y is in a pure chain, then it begins with d_p, which then is a prefix of $[\![X]\!] \in \text{orb}(\alpha', k)$, a contradiction. Suppose Y appear in an impure chain with the head X' such that $X'\beta \prec Y$ is induced by the chain. Let $X'\beta \prec X''$ be the first literal in the chain. By condition (9) in Definition 3.15, we have $|X| \geq |X'| + |\beta| + |d_p|$ in $\theta_{\mathbb{Z}}$. So if $[\![X]\!] \prec [\![Y]\!]$, then $[\![X]\!]$ has to contain d_p because $t_{X''} \equiv d_p \circ c_{X''} \circ d_p$ and $[\![X'\beta]\!] \circ t_{X''}$ is a prefix of $[\![Y]\!]$. We have a contradiction as $[\![X]\!] \in \text{orb}(\alpha', k)$. Therefore $[\![X]\!] \not\prec [\![Y]\!]$.

   (c) $X \not\prec \alpha Y$ occurs with $Y \in \text{orb}(\beta', l)$ but not with $X \in \text{orb}(\alpha', k)$. Then $[\![X]\!]$ contains the tag t_X while by Proposition 3.9 $[\![\alpha Y]\!]$ does not. So $[\![X]\!] \not\prec [\![\alpha Y]\!]$.

   (d) $X \not\prec \alpha Y$ occurs with neither $X \in \text{orb}(\alpha', k)$ nor $Y \in \text{orb}(\beta', l)$. Suppose that X and Y are not in the same chain. Then $[\![X]\!]$ contains the tag t_X while $[\![\alpha Y]\!]$ does not. So $[\![X]\!] \not\prec [\![\alpha Y]\!]$. Suppose that X and Y are in the same chain. $Y\beta \prec X$ cannot be implied by the chain due to condition (8) in Definition 3.15. Then it must be the case $X\beta \prec Y$ is implied and $\alpha \not\equiv \epsilon_Q$. It follows that $[\![\alpha X\beta]\!] \prec [\![\alpha Y]\!]$ and hence $[\![X]\!] \prec [\![\alpha X\beta]\!]$. Then $[\![X]\!] \in \text{orb}(\alpha, k)$ which cannot have d_p as a prefix. But $[\![Y]\!]$ begins with d_p, a contradiction.

6. $\alpha X \not\prec Y$.

   (a) $\alpha X \not\prec Y$ occurs with both $X \in \text{orb}(\alpha', k)$ and $Y \in \text{orb}(\beta', l)$. This could not happen as step (2) of Algorithm 3.18 would have replaced X, Y by instantiation.

   (b) $\alpha X \not\prec Y$ occurs with $X \in \text{orb}(\alpha', k)$ but not with $Y \in \text{orb}(\beta', l)$. Suppose $[\![\alpha X]\!] \prec [\![Y]\!]$. If Y is in a pure chain with head X', then it begins with $d_p \circ c_{X'} \circ d_p$. If $|d_p| \leq |\alpha|$, then d_p is a prefix of α, a contradiction. If $|d_p| > |\alpha|$, then the second occurrence of d_p must be a subsequence of $[\![X]\!] \in \text{orb}(\alpha', k)$, contradicting the choice of d_p. So Y must appear in an impure chain with the head X' such that $X'\beta \prec Y$ is induced by the chain. Let $X'\beta \prec X''$ be the first literal in the chain. By condition (9) in Definition 3.15, we have $|X| + |\alpha| \geq |X'| + |\beta| + |d_p|$ in $\theta_{\mathbb{Z}}$. Since $[\![\alpha X]\!] \prec [\![Y]\!]$, $[\![X]\!]$ has to contain d_p because $|[\![X]\!]| > |d_p|$, $t_{X''} \equiv d_p \circ c_{X''} \circ d_p$ and $[\![X'\beta]\!] \circ t_{X''}$ is a prefix of $[\![Y]\!]$. We have a contradiction as $[\![X]\!] \in \text{orb}(\alpha', k)$. Therefore $[\![\alpha X]\!] \not\prec [\![Y]\!]$.

   (c) $\alpha X \not\prec Y$ occurs with $Y \in \text{orb}(\beta', l)$ but not with $X \in \text{orb}(\alpha', k)$. Then $[\![\alpha X]\!]$ contains tag t_X while $[\![Y]\!]$ does not. Hence $[\![\alpha X]\!] \not\prec [\![Y]\!]$.

(d) $\alpha X \not\prec Y$ occurs with neither $X \in \mathrm{orb}(\alpha', k)$ nor $Y \in \mathrm{orb}(\beta', l)$. Suppose that X and Y are not in the same chain. Then $[\![\alpha X]\!]$ contains the tag t_X while $[\![Y]\!]$ does not. Hence $[\![\alpha X]\!] \not\prec [\![Y]\!]$. Suppose that X and Y are in the same chain. The chain cannot imply $Y\beta \prec X$ due to order completeness. Then it must be the case $X\beta \prec Y$ is implied and $\alpha \not\equiv \epsilon_{\boldsymbol{Q}}$. Suppose that $[\![\alpha X]\!] \prec [\![Y]\!]$. Then we have three cases: (i) $[\![\alpha X]\!] = [\![X\beta]\!]$, (ii) $[\![\alpha X]\!] \prec [\![X\beta]\!]$ or (iii) $[\![X\beta]\!] \prec [\![\alpha X]\!]$. In all cases, we have $[\![X]\!] \in \mathrm{orb}(\alpha, k)$. But the assignment requires $[\![X]\!]$ contain tag t_X and hence d_p, a contradiction.

Thus, we can find a solution $(X_i)_n$ for $\Phi_{\boldsymbol{Q}}$ for each corresponding integer solution for $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L_t^+)$. That means $\Phi_{\Delta+} \wedge \theta_{\mathbb{Z}} \wedge C_\Phi(L_t^+)$ is an RLCC for $\Phi_{\boldsymbol{Q}}/(\theta_{\mathbb{Z}} \wedge C_\Phi(L_t^+))$. □

Theorem 3.6 (RLCC with Parameters in $\mathfrak{Q}_{\mathbb{Z}}$). $\Phi_{\Delta+}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge C_\Phi(L_t)$ is an RLCC for $\Phi_{\boldsymbol{Q}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}})/\bar{\mathbf{X}}/(\theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge C_\Phi(L_t))$.

Proof. Similar to the proof of Theorem 3.2. Now we view parameters $\bar{\mathbf{Y}}$ as arbitrary but fixed constant queues. So $\boldsymbol{Q}$-terms like $t(Y)$ are fixed too. For every variable X, we can assign a unique color c_X such that for any $t(Y)$,

$$c_X \neq t(Y)[L_p + 1 \,..\, L_p + L_c] \ . \tag{3.64}$$

This can be done because we treat syntactically different $t(Y)$ as distinct variables and L_c is long enough to be able to encode all $\boldsymbol{Q}$-variables and proper constant queues. We are left to show the satisfiability of

$$X \neq t(Y) \ .$$

By Algorithm 3.25, literals of the form $X \in \mathrm{orb}(\alpha, k)$ do not occur. So $[\![X]\!]$ is defined by (3.58). It follows directly from (3.64) that $[\![X]\!] \neq t(Y)$ holds. □

Theorem 3.7 (Soundness of Algorithm 3.26). Algorithm 3.26 terminates and preserves equivalence.

Proof. The termination of Algorithm 3.26 is straightforward. Steps (1)-(2) are obviously sound. The correctness then reduces to the equivalence between (3.52) and (3.53).

$$(\exists \bar{\mathbf{X}} : \boldsymbol{Q}) \Big[\Phi_{\boldsymbol{Q}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \ \wedge \ C \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{X}}, \bar{\mathbf{Y}}) \Big] \ , \tag{3.52}$$

$$\Phi^{(2)}_{\boldsymbol{Q}}(\bar{\mathbf{Y}}) \ \wedge \ (\exists \bar{\mathbf{v}} : \mathbb{Z}) \Big[\Phi_{\Delta+}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \ \wedge \ C \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{Y}}) \Big] \ . \tag{3.53}$$

Note that after step (2), for the variables left in $\bar{\mathbf{X}}$, we have $|X| \geq L_t$ in C and so we can assume C is $C_\Phi(L_t)$. By Theorem 3.6, $\Phi_{\Delta+}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \wedge \theta_{\mathbb{Z}}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \wedge C_\Phi(L_t)$ is an RLCC for $\Phi_{\boldsymbol{Q}}(\bar{\mathbf{X}}, \bar{\mathbf{Y}})/\bar{\mathbf{X}}/(\theta_{\mathbb{Z}}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \wedge C_\Phi(L_t))$.

By Theorem 3.6, "(3.52) $\Rightarrow$ (3.53)" follows from (3.49) and "(3.53) $\Rightarrow$ (3.52)" follows from (3.50) noting that $\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \equiv \Phi_Q^{(1)}(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \wedge \Phi_Q^{(2)}(\bar{\mathbf{Y}})$, and (by the additive property of RLCC)

$$\Phi_{\Delta+}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \ \wedge \ C_\Phi(L_t) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{Y}})$$

is an RLCC for

$$\Phi_Q(\bar{\mathbf{X}}, \bar{\mathbf{Y}})/\bar{\mathbf{X}}/(\theta_{\mathbb{Z}}(\bar{\mathbf{v}}, \bar{\mathbf{Y}}) \ \wedge \ C_\Phi(L_t) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\mathbf{u}}, \bar{\mathbf{v}}, \bar{\mathbf{Y}})) \ . \qquad \square$$

Chapter 4

Decidability of Knuth-Bendix Order

01:39:04:310

> Trinity : No one has ever done anything like this.
> Neo : That's why it's going to work.
>
> —*The Matrix*.

Two kinds of orderings are widely used in term rewriting and theorem proving, namely recursive path ordering *(RPO) and* Knuth-Bendix ordering *(KBO). They provide powerful tools to prove the termination of rewriting systems. They are also applied in ordered resolution to prune the search space without compromising refutational completeness. Solving ordering constraints is therefore essential to the successful application of ordered rewriting and ordered resolution. Besides the needs for decision procedures for quantifier-free theories, situations arise in constrained deduction where the truth value of quantified formulas must be decided. Unfortunately, the full first-order theory of lexicographic path ordering (LPO), the most popular form of RPO, is undecidable. This leaves an open question whether the first-order theory of KBO is decidable. In this chapter, we give a positive answer to this question using the very reduction technique presented in the previous chapters.*

4.1 Introduction

Two kinds of orderings are widely used in term rewriting and theorem proving. One is *recursive path ordering* (RPO) which is based on syntactic precedence [14]. The other is *Knuth-Bendix ordering* (KBO) which is of hybrid nature; it relies on numerical values assigned to symbols as well as syntactic precedence [25]. In ordered term rewriting, a strategy built on ordering constraints can dynamically orient an equation, at the time of instantiation, even if the equation is not uniformly orientable. This provides a powerful tool to prove the termination of rewriting systems [10]. In ordered resolution and paramodulation, ordering constraints are used to select maximal literals to perform resolution. It also serves as enabling conditions for inference rules and such conditions can be inherited from previous inferences at each deduction step. This helps to prune redundancy of the search space without compromising refutational completeness [43].

Solving ordering constraints is therefore essential to the successful application of ordered rewriting and ordered resolution. The decision procedures for quantifier-free constraints of both types of orderings have been well-studied [7, 21, 41, 38, 42, 26, 27]. However, situations arise where we need to decide the truth values of quantified formulas on those orderings, especially in the $\forall^*\exists^*$ fragment. Examples include checking the soundness of simplification rules in constrained deduction. Consider a "total simplification scheme" given in [23, 11].

$$\frac{s \;\rightarrow\; t \mid c}{s[v]_p \;\rightarrow\; t \mid (c \wedge c' \wedge s|_p = u)} \quad (u \;\rightarrow\; v \mid c') \;\;,$$

where $s|_p$ denotes the subterm occurring at position p in s and $s[v]_p$ denotes the term obtained from s by substituting v for $s|_p$, states that $s \rightarrow t \mid c$ is simplified to $s[v]_p \rightarrow t \mid (c \wedge c' \wedge s|_p = u)$ by $u \rightarrow v \mid c'$ provided *for all* assignments for variables in s which satisfies c, *there exists* an assignment for variables in u which satisfies c' and $s|_p = u$. The soundness of this rule is formally expressed as

$$\mathrm{TA} \models \forall \mathcal{V}(s) \exists \mathcal{V}(u) \Big[\; c \;\rightarrow\; (c' \;\wedge\; s|_p = u) \;\Big] \;\;,$$

which necessarily involves quantifier alternation. To determine the soundness of such simplification rules, we need to be able to reason in the $\forall^*\exists^*$ fragment.

Unfortunately, the full first-order theory of lexicographic path ordering (LPO), the most popular form of RPO, is undecidable [56, 11] except for the special case where the language only has unary functions and the precedence order is total [37]. Until now it has been an open question whether the first-order theory of Knuth-Bendix order is decidable (RTA open problem ♯99). Here we answer this question affirmatively by showing that an extended theory of term algebras with Knuth-Bendix order admits quantifier elimination.

The basic framework is the combination of term algebras with Presburger arithmetic. The combination is more tightly coupled than $\mathrm{TA}_{\mathbb{Z}}$ presented in Chapter 2: not only do we have a

weight function mapping terms to integers, but we also have various *boundary functions* mapping integers to terms. In addition, the Knuth-Bendix order is expanded in two directions. First, the order is decomposed into three disjoint suborders depending on which of three conditions is used in the definition. Secondly, all orders (including the suborders) are extended to gap orders, which assert the least number of distinct objects between two terms. Moreover, as Knuth-Bendix order is recursively defined on a lexicographic extension of itself, gap orders are extended to tuples of terms. Thus we actually establish the decidability of a richer theory.

Related Work and Comparison. The decidability of the theory of RPO has been well-studied. Comon proves the decidability of the quantifier-free theory of total lexicographic path ordering (LPO, a variant of RPO) [7]. A similar result holds for RPO [21]. Nieuwenhuis establishes the NP-completeness for the quantifier-free theory of LPO [41]. Narendran, Rusinowitch and Verma obtain a similar result for RPO [38]. A more efficient algorithm for the quantifier-free theory of RPO is given by Nieuwenhuis and Rivero [42]. Comon and Treinen show the undecidability of the first-order theory of LPO and the undecidability of the first-order theory of RPO in case of partial precedence [56, 11]. The decidability of the first-order theory of RPO (LPO) in case of unary signature and total precedence is due to Narendran and Rusinowitch [37]. The decidability of the first-order theory of RPO in case of total precedence remains open.

Recently some partial decidability results for the theory of KBO have been obtained. Korovin and Voronkov show the decidability of the quantifier-free theory of term algebras with KBO [26]. They later improve the algorithm and shows that the quantifier-free theory of KBO is NP-complete [27]. Analogous to [37], they also show the decidability of the first-order theory of KBO in the case where all functions are unary [28].

In this chapter, we show the general decidability result for an extended theory of KBO with arbitrary function symbols and weight functions.

Chapter Organization. Section 4.2 introduces the theory of term algebras with Knuth-Bendix ordering and presents the technical machinery for eliminating quantifiers. Section 4.3 presents the main contribution of this chapter: it expands the elimination procedure in [60] for the extended theory of KBO and proves its correctness. Section 4.4 explains how to adapt the elimination procedure to the special case where the language contains a unary function of weight 0. Section 4.5 concludes with some ideas for future work. Most proofs are given in Section 4.6. They are supported by integer predicates and functions defined in Section 4.7 and reductions of gap order literals listed in Section 4.8.

4.2 Term Algebras with Knuth-Bendix Order

In this section we introduce the theory of term algebras with KBO and present the technical machinery needed in the quantifier elimination procedure.

Let Σ be a finite signature in the constructor language (i. e., $\Sigma = C$ in Definition 2.1) and $\mathsf{w} : \Sigma \to \mathbb{N}$ a weight function. We expand $\mathrm{dom}(\mathsf{w})$ to the ground term domain $\mathbb{T}$ by recursively defining $\mathsf{w}(\alpha(t_1, \ldots, t_k)) = \mathsf{w}(\alpha) + \sum_{i=1}^{k} \mathsf{w}(t_i)$. Let $\prec^\Sigma$ be a linear precedence order on symbols in Σ. We enumerate all symbols in the decreasing $\prec^\Sigma$-order such that $\alpha_1 \succ^\Sigma \alpha_2 \succ^\Sigma \cdots \succ^\Sigma \alpha_{|\Sigma|}$.

Definition 4.1 (Knuth-Bendix Order [25]). *A Knuth-Bendix order (KBO) $\prec^{\mathrm{kb}}$ (parameterized with a weight function w and a precedence order $\prec^\Sigma$) is defined recursively on $\mathbb{T}$ such that for $u, v \in \mathbb{T}$, $u \prec^{\mathrm{kb}} v$ if and only if one of the following conditions holds:*

1. $\mathsf{w}(u) < \mathsf{w}(v)$;
2. $\mathsf{w}(u) = \mathsf{w}(v)$ *and* $\mathrm{type}(u) \prec^\Sigma \mathrm{type}(v)$;
3. $\mathsf{w}(u) = \mathsf{w}(v)$, $u \equiv \alpha(u_1, \ldots, u_k)$, $v \equiv \alpha(v_1, \ldots, v_k)$ *and*

$$(\exists i)\Big[\, 0 < i \leq k \;\wedge\; u_i \prec^{\mathrm{kb}} v_i \;\wedge \forall j(0 < j < i \to u_j = v_j) \,\Big] \,. \tag{4.1}$$

The KBO $\prec^{\mathrm{kb}}$ is a well-founded total order on $\mathbb{T}$ [25, 2]. To guarantee well-foundedness, two *compatibility conditions* for w and $\prec^\Sigma$ are required:

(i) $\mathsf{w}(a) > 0$ for any constant a, and

(ii) a unary function of weight 0, if present, should be the maximum in $\prec^\Sigma$.

Let us denote by $\bot$ the smallest term with respect to $\prec^{\mathrm{kb}}$. It follows from (i) and (ii) that $\bot$ must be a constant and so it can be determined when w and $\prec^\Sigma$ are given. By (ii) if a unary function of weight 0 exists, it must be unique. For presentation simplicity, we assume that $\mathsf{w}(\alpha_1) > 0$. However, the existence of such function in fact considerably simplifies our decision procedure. We defer the discussion to Section 4.4.

Definition 4.2 (Term Algebras with Knuth-Bendix Order). *The structure of term algebras with KBO is* $\mathrm{TA}_{\mathrm{kb}} = \langle \mathrm{TA}; \prec^{\mathrm{kb}} \rangle$. *Let $\mathscr{L}_{\mathrm{kb}}$ denote the language of* $\mathrm{TA}_{\mathrm{kb}}$.

In the rest of this chapter, we assume our formal language does not have constructors (except constants). This does not compromise expressiveness as constructors can be defined by selectors and testers. Moreover, this simplification reduces chances of confusion by separating semantic objects (ground constructor terms) in $\mathbb{T}$ from syntactic objects (selector terms) in $\mathscr{L}_{\mathrm{kb}}$.

4.2.1 Proof Plan

We shall show the decidability of $\mathrm{Th}(\mathrm{TA}_{\mathrm{kb}})$ by quantifier elimination. The procedure relies on the following two ideas: *solved form* and *depth reduction*.

1. *Solved Form.* A quantifier-free formula $\varphi(x, \bar{\mathbf{y}})$ is *solved* in x if it is in the form

$$\bigwedge_{i \le m} u_i \prec^{\mathrm{kb}} x \ \wedge \ \bigwedge_{j \le n} x \prec^{\mathrm{kb}} v_j \ \wedge \ \varphi'(\bar{\mathbf{y}}) \ , \tag{4.2}$$

   where x does not appear in u_i, v_i and φ'. It is not hard to argue that $(\exists x)\, \varphi(x, \bar{\mathbf{y}})$ simplifies to

$$\bigwedge_{i \le m, j \le n} u_i \prec_2^{\mathrm{kb}} v_j \ \wedge \ \varphi'(\bar{\mathbf{y}}) \ , \tag{4.3}$$

   where $\prec_n^{\mathrm{kb}}$, called *gap order*, is an extension of $\prec^{\mathrm{kb}}$ such that $x \prec_n^{\mathrm{kb}} y$ states there is an increasing chain from x to y with at least $n-1$ elements in between [16, page 196]. It is clear that the elimination of $\exists x$, the transformation from (4.2) to (4.3), becomes straightforward once the matrix $\varphi(x, \bar{\mathbf{y}})$ is solved in x, or equivalently, $\mathrm{depth}^{\varphi}(x) = 0$. That leads us to the notion of *depth reduction*.

2. *Depth Reduction.* Let us first consider the simple case where x is α-typed for a proper constructor α and all occurrences of x have depth greater than 0. By introducing new variables $x_1, \ldots, x_{\mathrm{ar}(\alpha)}$ (called the *descendants* of x) to represent x, we can rewrite $\exists x \varphi(x, \bar{\mathbf{y}})$ to

$$\exists x_1, \ldots, \exists x_{\mathrm{ar}(\alpha)}\, \varphi'(x_1, \ldots, x_{\mathrm{ar}(\alpha)}, \bar{\mathbf{y}}) \ , \tag{4.4}$$

   where $\varphi'(x_1, \ldots, x_{\mathrm{ar}(\alpha)}, \bar{\mathbf{y}})$ is obtained from $\varphi(x, \bar{\mathbf{y}})$ by substituting x_i for $\mathsf{s}_i^{\alpha} x$ $(0 < i \le \mathrm{ar}(\alpha))$. It is clear that $\mathrm{depth}^{\varphi'}(x_i) < \mathrm{depth}^{\varphi}(x)$. If all occurrences of x have the same depth, then by repeating the process we can generate a formula solved in $\bar{\mathbf{x}}^*$ where $\bar{\mathbf{x}}^*$ are descendants of x. A difficulty arises when not all occurrences of x have equal depth. So eventually we meet the situation where some occurrences of x have depth 0 and some do not. Here we have to represent all occurrences of x of depth 0 in terms of $\mathsf{s}_1^{\alpha}(x), \ldots, \mathsf{s}_{\mathrm{ar}(\alpha)}^{\alpha}(x)$. This amounts to reducing literals of the form $x \prec_n^{\mathrm{kb}} t$ and literals of the form $t \prec_n^{\mathrm{kb}} x$ to quantifier-free formulas using $\mathsf{s}_1^{\alpha}(x), \ldots, \mathsf{s}_{\mathrm{ar}(\alpha)}^{\alpha}(x)$. After that we can introduce new variables and do quantifier manipulation just as in the simple case to bring $\exists x \varphi(x, \bar{\mathbf{y}})$ into the form of (4.4). Therefore by the depth reduction of x, we actually mean reducing the depths of the descendants of x, and this essentially depends on the reduction of $x \prec_n^{\mathrm{kb}} t$ and $t \prec_n^{\mathrm{kb}} x$. In order to carry out the reduction we need to extend the language extensively as follows.

   (a) We decompose $\prec^{\mathrm{kb}}$ into three disjoint suborders $\prec^{\mathsf{w}}$, $\prec^{\mathsf{p}}$ and $\prec^{\mathsf{l}}$, each of which is also extended to gap orders.

(b) We introduce Presburger arithmetic explicitly in order to define *counting constraints* to count how many distinct terms there are at certain weight, and define *boundary functions* to *delineate* gap orders.

(c) The reduction of literals like $x \prec_n^{\mathrm{kb}} t$ or $t \prec_n^{\mathrm{kb}} x$ eventually comes down to resolving relations between two terms of the same weight and of the same type. So we need to extend all aforementioned notions to tuples of terms of the same total weight.

In the rest of this section we define these extensions.

4.2.2 Decomposition of Knuth-Bendix Order

Definition 4.3 (Decomposition of KBO). *A Knuth-Bendix order $\prec^{\mathrm{kb}}$ can be decomposed into three* disjoint *orders, a* weight order $\prec^{\mathsf{w}}$*, a* precedence order $\prec^{\mathsf{p}}$*, and a* lexicographic order $\prec^{\mathsf{l}}$*, as follows:*

$$
\begin{aligned}
u \prec^{\mathsf{w}} v &\;\leftrightarrow\; \mathsf{w}(u) < \mathsf{w}(v) \;, \\
u \prec^{\mathsf{p}} v &\;\leftrightarrow\; \mathsf{w}(u) = \mathsf{w}(v) \;\wedge\; \mathrm{type}(u) \prec^{\Sigma} \mathrm{type}(v) \;, \\
u \prec^{\mathsf{l}} v &\;\leftrightarrow\; \mathsf{w}(u) = \mathsf{w}(v) \;\wedge\; \mathrm{type}(u) = \mathrm{type}(v) \;\wedge\; u \prec^{\mathrm{kb}} v \;.
\end{aligned}
$$

It is clear that $u \prec^{\mathrm{kb}} v$ is equivalent to $u \prec^{\mathsf{w}} v \vee u \prec^{\mathsf{p}} v \vee u \prec^{\mathsf{l}} v$. We write $u \prec^{\mathsf{pl}} v$ for $u \prec^{\mathsf{p}} v \vee u \prec^{\mathsf{l}} v$ and $u \preceq^{\sharp} v$ for $u \prec^{\sharp} v \vee u = v$ ($\sharp \in \{\mathsf{kb}, \mathsf{w}, \mathsf{p}, \mathsf{l}, \mathsf{pl}\}$). We say that w has the highest priority, followed by p and l, and define the maximum and minimum of a set of suborders accordingly. For example, $\max\{\mathsf{w}, \mathsf{p}, \mathsf{l}\} = \mathsf{w}$ and $\min\{\mathsf{w}, \mathsf{p}, \mathsf{l}\} = \mathsf{l}$.

4.2.3 Gap Orders

To express formulas of the form $\exists x(u \prec^{\sharp} x \prec^{\sharp} v)$ ($\sharp \in \{\mathsf{kb}, \mathsf{w}, \mathsf{p}, \mathsf{l}, \mathsf{pl}\}$), in a quantifier-free language we need to extend all aforementioned orders to *gap orders*.

Definition 4.4 (Gap Orders). *Define $\prec_n^{\mathrm{kb}}$ ($n \geq 0$) such that $\prec_0^{\mathrm{kb}}$ is $=$ and for $n > 0$*

$$
u \prec_n^{\mathrm{kb}} v \;\leftrightarrow\; (\exists u_1, \ldots, \exists u_n)\Big[u \;\prec^{\mathrm{kb}}\; u_1 \;\prec^{\mathrm{kb}}\; \cdots \;\prec^{\mathrm{kb}}\; u_n \;\preceq^{\mathrm{kb}}\; v\Big] \;.
$$

Let $\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}, \mathsf{pl}\}$. Define two orders $\prec_n^{\sharp}$ and $\leqq_n^{\sharp}$ ($n \geq 0$) such that $\prec_0^{\sharp}$ is $\preceq^{\sharp}$ and $\leqq_0^{\sharp}$ is $=$, and for $n > 0$

$$
\begin{aligned}
u \prec_n^{\sharp} v &\;\leftrightarrow\; u \;\prec_n^{\mathrm{kb}}\; v \;\wedge\; u \prec^{\sharp} v \;, \\
u \leqq_n^{\sharp} v &\;\leftrightarrow\; u \;\prec_n^{\sharp}\; v \;\wedge\; \neg\,(u \prec_{n+1}^{\sharp} v) \;.
\end{aligned}
$$

Note that $\prec_1^{\sharp}$ is just $\prec^{\sharp}$, and for $n \geq 0$, we have

$$
u \prec_n^{\sharp} v \;\leftrightarrow\; u \prec_{n+1}^{\sharp} v \;\vee\; u \leqq_n^{\sharp} v \;.
$$

A gap order $u \prec_n^\sharp v$ ($n > 0$) states that "u is less than v w.r.t. $\prec^\sharp$, and there are *at least* $n-1$ elements in between." Similarly, $u \leqq_n^\sharp v$ ($n > 0$) states that "u is less than v w.r.t. $\prec^\sharp$, and there are *exactly* $n-1$ elements in between". We call $\prec_n^\sharp$ and $\leqq_n^\sharp$ *stretchable* gap orders and *rigid* gap orders, respectively. Accordingly, $u \prec_n^\sharp v$ are *stretchable* gap order literals and $u \leqq_n^\sharp v$ are *rigid* gap order literals.

Example 4.1 (Elimination in Linear Order). *The formula* $\exists x(u \prec^\mathsf{l} x \prec^\mathsf{l} v)$ *reduces to* $u \prec_2^\mathsf{l} v$ *if* u, v *do not contain* x.

4.2.4 Boundary Functions

Consider the formula $u \leqq_1^\mathsf{w} v$. Intuitively it states "$\mathsf{w}(u) < \mathsf{w}(v)$ and there are no terms z such that $u \prec^\mathrm{kb} z \prec^\mathrm{kb} v$, that is, u is the largest term of weight $\mathsf{w}(u)$ and v is the smallest term of weight $\mathsf{w}(v)$". To express this we introduce *boundary functions*.

Definition 4.5 (Boundary Functions). *Let* $n, p > 0$. *The following functions are called boundary functions:*

1. $0^\mathsf{w} : \mathbb{N} \to \mathbb{T}$ *such that* $0^\mathsf{w}(n)$ *is the smallest term (w.r.t.* $\prec^\mathrm{kb}$*) of weight* n,
2. $0^\mathsf{p} : \mathbb{N}^2 \to \mathbb{T}$ *such that* $0^\mathsf{p}(n,p)$ *is the smallest term (w.r.t.* $\prec^\mathrm{kb}$*) of weight* n *and type* α_p,
3. $1^\mathsf{w} : \mathbb{N} \to \mathbb{T}$ *such that* $1^\mathsf{w}(n)$ *is the largest term (w.r.t.* $\prec^\mathrm{kb}$*) of weight* n,
4. $1^\mathsf{p} : \mathbb{N}^2 \to \mathbb{T}$ *such that* $1^\mathsf{p}(n,p)$ *is the largest term (w.r.t.* $\prec^\mathrm{kb}$*) of weight* n *and type* α_p,

where, for all of the above, $f(n) = \bot$ *and* $f(n,p) = \bot$, *if no such term exists.*

We write $0^\sharp_{(\ldots)}$ for $0^\sharp(\ldots)$ and $1^\sharp_{(\ldots)}$ for $1^\sharp(\ldots)$. We call $0^\sharp_{(\ldots)}$ *lower boundary functions* and $1^\sharp_{(\ldots)}$ *upper boundary functions*. TA-terms having one of these functions as root symbol are called *boundary terms*, distinguishing them from ordinary TA-terms. Correspondingly, we have *lower boundary terms* and *upper boundary terms*. A literal of the form $u \star v$, where $\star$ is either a term equality or a gap order, is *open* if both u and v are ordinary TA-terms, *closed* if both u and v are boundary terms, and *half-open* otherwise. Although by definition boundary terms are syntactic objects, with little chance of confusion, we also call the corresponding semantic terms in $\mathbb{T}$ (ground) boundary terms.

4.2.5 Extension of Knuth-Bendix Order

We build the extension on $\mathrm{TA}_\mathbb{Z}$ with the length function $|\cdot|$ being replaced by the weight function $(.)^\mathsf{w}$. As before we use *integer occurrence* for a TA-term appearing inside the weight function, to distinguish it from the normal *term occurrence*. We use $\mathrm{depth}^\varphi(x)$ to denote the maximum depth of term occurrences of x in φ.

Example 4.2 (Extension of Knuth-Bendix Order). *The formula*

$$(\exists x : \mathbb{T})\left[\ \mathsf{0}^{\mathsf{w}}_{(x^{\mathsf{w}})} \prec^{\mathsf{pl}} x \ \wedge\ x \prec^{\mathsf{pl}} \mathsf{1}^{\mathsf{w}}_{(x^{\mathsf{w}})}\ \right]$$

states that there exists a term $x \in \mathbb{T}$ such that there are at least three elements with the same weight as x (including x itself). Note that the first and the third occurrences of x are integral while the second one is an ordinary term.

Definition 4.6 (Term Algebras with KBO and Its Extensions). *The structure of term algebras with KBO, extended with gap orders, boundary functions and Presburger arithmetic, is*

$$\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+} = \langle \mathrm{TA}_{\mathrm{kb}};\ \mathrm{TA}_{\mathbb{Z}};\ \prec^{\sharp}_n, \preceq^{\sharp}_n,\ \sharp \in \{\mathsf{kb},\mathsf{w},\mathsf{p},\mathsf{l},\mathsf{pl}\}, n \geq 0;\ \mathsf{0}^{*}_{(\ldots)}, \mathsf{1}^{*}_{(\ldots)},\ * \in \{\mathsf{w},\mathsf{p}\}\rangle\ .$$

We denote by $\mathscr{L}_{\mathrm{kb}^+}$ the language extending $\mathscr{L}_{\mathrm{kb}}$ with gap orders and boundary terms and by $\mathscr{L}_{\mathbb{Z}}$ the language of Presburger arithmetic (including weight functions on terms). The complete language is denoted by $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$.

4.2.6 Tuples of Terms

The extensions for tuples of terms are defined as follows:

Definition 4.7 (KBO on Tuples). *Let $\bar{\mathbf{u}} = \langle u_1, \ldots, u_k\rangle, \bar{\mathbf{v}} = \langle v_1, \ldots, v_k\rangle$ such that $\sum_{i=1}^{k} \mathsf{w}(u_i) = \sum_{i=1}^{k} \mathsf{w}(v_i)$. The lexicographic extension $\prec^{k;\mathrm{kb}}$ ($k \geq 1$) of $\prec^{\mathrm{kb}}$ on k-tuples of the same weight is defined such that $\bar{\mathbf{u}} \prec^{k;\mathrm{kb}} \bar{\mathbf{v}}$ if and only if* (4.1) *holds.*

Definition 4.8 (Suborders on Tuples). *Let $\bar{\mathbf{u}} = \langle u_1, \ldots, u_k\rangle$, $\bar{\mathbf{v}} = \langle v_1, \ldots, v_k\rangle \in \mathbb{T}^k$, $\sharp \in \{\mathsf{w},\mathsf{p},\mathsf{l},\mathsf{pl}\}$. We define those composite orders on tuples as follows.*

$$\bar{\mathbf{u}} \prec^{k;\sharp} \bar{\mathbf{v}} \ \leftrightarrow\ u_1 \prec^{\sharp} v_1 \ \vee\ (u_1 = v_1 \ \wedge\ \langle u_2, \ldots, u_k\rangle \prec^{k-1;\mathrm{kb}} \langle v_2, \ldots, v_k\rangle)\ .$$

We say that $\prec^{k;\sharp}$ is a tuple order of length k. We identify tuple orders of length 1 with term orders. We say that $\bar{\mathbf{u}} \prec^{k;\sharp} \bar{\mathbf{v}}$ is *proper* if $u_1 \prec^{\sharp} v_1$ and we have

$$\bar{\mathbf{u}} \prec^{k;\mathrm{kb}} \bar{\mathbf{v}} \ \leftrightarrow\ \bar{\mathbf{u}} \prec^{k;\mathsf{w}} \bar{\mathbf{v}} \ \vee\ \bar{\mathbf{u}} \prec^{k;\mathsf{p}} \bar{\mathbf{v}} \ \vee\ \bar{\mathbf{u}} \prec^{k;\mathsf{l}} \bar{\mathbf{v}}\ .$$

We write $\bar{\mathbf{u}} \prec^{k;\mathrm{pl}} \bar{\mathbf{v}}$ for $\bar{\mathbf{u}} \prec^{k;\mathsf{p}} \bar{\mathbf{v}} \vee \bar{\mathbf{u}} \prec^{k;\mathsf{l}} \bar{\mathbf{v}}$ and $\bar{\mathbf{u}} \preceq^{k;\sharp} \bar{\mathbf{v}}$ for $\bar{\mathbf{u}} = \bar{\mathbf{v}} \vee \bar{\mathbf{u}} \prec^{k;\sharp} \bar{\mathbf{v}}$ ($\sharp \in \{\mathsf{kb},\mathsf{w},\mathsf{p},\mathsf{l},\mathsf{pl}\}$).

Definition 4.9 (Tuple Gap Orders). *We define $\prec^{k;\mathrm{kb}}_n$ ($k > 0; n \geq 0$) such that $\prec^{k;\mathrm{kb}}_0$ is $=$ and for $n > 0$*

$$\bar{\mathbf{u}} \prec^{k;\mathrm{kb}}_n \bar{\mathbf{v}} \ \leftrightarrow\ (\exists \bar{\mathbf{u}}_1, \ldots, \exists \bar{\mathbf{u}}_n : \mathbb{T}^k)\left[\ \bar{\mathbf{u}} \prec^{k;\mathrm{kb}} \bar{\mathbf{u}}_1 \prec^{k;\mathrm{kb}} \cdots \prec^{k;\mathrm{kb}} \bar{\mathbf{u}}_n \preceq^{k;\mathrm{kb}} \bar{\mathbf{v}}\right]\ .$$

Let $\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}, \mathsf{pl}\}$. *Define two orders* $\prec_n^{k;\sharp}$ *and* $\preceqq_n^{k;\sharp}$ $(k \geq 1; n \geq 0)$ *such that* $\prec_0^{k;\sharp}$ *is* $\preceq^{k;\sharp}$, $\preceqq_0^{k;\sharp}$ *is* $=$, *and for* $n > 0$

$$\bar{\mathbf{u}} \prec_n^{k;\sharp} \bar{\mathbf{v}} \;\leftrightarrow\; \bar{\mathbf{u}} \prec_n^{k;\mathrm{kb}} \bar{\mathbf{v}} \wedge \bar{\mathbf{u}} \prec^{k;\sharp} \bar{\mathbf{v}} \;,$$
$$\bar{\mathbf{u}} \preceqq_n^{k;\sharp} \bar{\mathbf{v}} \;\leftrightarrow\; \bar{\mathbf{u}} \prec_n^{k;\sharp} \bar{\mathbf{v}} \wedge \neg\, (\bar{\mathbf{u}} \prec_{n+1}^{k;\sharp} \bar{\mathbf{v}}) \;.$$

Similar as term gap orders, $\prec_1^{k;\sharp}$ is just $\prec^{k;\sharp}$, and we have

$$\bar{\mathbf{u}} \prec_n^{k;\sharp} \bar{\mathbf{v}} \;\leftrightarrow\; \bar{\mathbf{u}} \prec_{n+1}^{k;\sharp} \bar{\mathbf{v}} \vee \bar{\mathbf{u}} \preceqq_n^{k;\sharp} \bar{\mathbf{v}} \;.$$

As before we call $\prec_n^{k;\sharp}$ and $\preceqq_n^{k;\sharp}$ *stretchable* tuple gap orders and *rigid* tuple gap orders, respectively. Accordingly, $u \prec_n^{k;\sharp} v$ are *stretchable* tuple gap order literals and $u \preceqq_n^{k;\sharp} v$ are *rigid* tuple gap order literals.

Definition 4.10 (Tuple Boundary Functions). *Let* $k, n, m, p > 0$. *Define partial functions:*

1. $\overline{0}^{k;\mathrm{kb}} : \mathbb{N} \to \mathbb{T}^k$ $(k \geq 1)$ *such that* $\overline{0}^{k;\mathrm{kb}}(n)$ *is the smallest k-tuple (w.r.t.* $\prec^{k;\mathrm{kb}}$*) of weight n.*
2. $\overline{0}^{k;\mathsf{w}} : \mathbb{N}^2 \to \mathbb{T}^k$ $(k \geq 1)$ *such that* $\overline{0}^{k;\mathsf{w}}(n, m)$ *is the smallest k-tuple (w.r.t.* $\prec^{k;\mathrm{kb}}$*) of weight n and its first component has weight m.*
3. $\overline{0}^{k;\mathsf{p}} : \mathbb{N}^3 \to \mathbb{T}^k$ $(k \geq 1)$ *such that* $\overline{0}^{k;\mathsf{p}}(n, m, p)$ *is the smallest k-tuple (w.r.t.* $\prec^{k;\mathrm{kb}}$*) of weight n and its first component has weight m and type* α_p.
4. $\overline{1}^{k;\mathrm{kb}} : \mathbb{N} \to \mathbb{T}^k$ $(k \geq 1)$ *such that* $\overline{0}^{k;\mathrm{kb}}(n)$ *is the largest k-tuple (w.r.t.* $\prec^{k;\mathrm{kb}}$*) of weight n.*
5. $\overline{1}^{k;\mathsf{w}} : \mathbb{N}^2 \to \mathbb{T}^k$ $(k \geq 1)$ *such that* $\overline{0}^{k;\mathsf{w}}(n, m)$ *is the largest k-tuple (w.r.t.* $\prec^{k;\mathrm{kb}}$*) of weight n and its first component has weight m.*
6. $\overline{1}^{k;\mathsf{p}} : \mathbb{N}^3 \to \mathbb{T}^k$ $(k \geq 1)$ *such that* $\overline{0}^{k;\mathsf{p}}(n, m, p)$ *is the largest k-tuple (w.r.t.* $\prec^{k;\mathrm{kb}}$*) of weight n and its first component has weight m and type* α_p.

As before these functions are made total by assigning $\langle \bot, \ldots \bot \rangle$ to undefined values. We write $\overline{0}_{(\ldots)}^{k;\sharp}$ for $\overline{0}^{k;\sharp}(\ldots)$ and $\overline{1}_{(\ldots)}^{k;\sharp}$ for $\overline{1}^{k;\sharp}(\ldots)$. We call $\overline{0}_{(\ldots)}^{k;\sharp}$ *lower tuple boundary functions* and $\overline{1}_{(\ldots)}^{k;\sharp}$ *upper tuple boundary functions*. Terms having one of these functions as root symbol are called *boundary tuples*. Correspondingly, we have *lower boundary tuples* and *upper boundary tuples*. As before we call a *tuple literal* $\bar{\mathbf{u}} \star \bar{\mathbf{v}}$ *open* if both $\bar{\mathbf{u}}$ and $\bar{\mathbf{v}}$ are ordinary tuples, *closed* if both $\bar{\mathbf{u}}$ and $\bar{\mathbf{v}}$ are boundary tuples, and *half-open* otherwise.

To avoid unnecessary complications, we choose to treat tuples (including boundary tuples) as "syntactic sugar"; they are only used in the intermediate steps of the reduction. Lemma 4.7 shows that literals containing tuples can be reduced to formulas in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$.

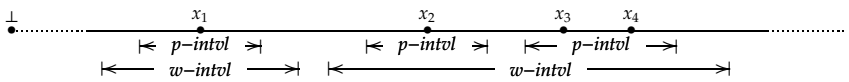


Figure 4-1: An order arrangement of $x_1 \prec^{\mathsf{w}}_{n_1} x_2 \prec^{\mathsf{p}}_{n_2} x_3 \prec^{\mathsf{l}}_{n_3} x_4$.

4.2.7 Delineated Gap Order Completion

Revisiting the transformation from (4.2) to (4.3), we see that the number of gap orders in (4.3) is quadratic in the number of gap orders in (4.2). This complicates the termination proof for the elimination procedure. Nevertheless, we can avoid this difficulty by postulating the relative positions of parameters. This leads to the notion of *order completion*.

Definition 4.11 (Gap Order Completion). *A* gap order completion *(GOC) of a set of terms* $\{t_1, \ldots, t_n\}$ *is the chain*

$$t_{f(1)} \trianglelefteq_1 t_{f(2)} \trianglelefteq_2 \cdots \trianglelefteq_{n-1} t_{f(n)} \ ,$$

where f *is a permutation function on* $\{1, \ldots, n\}$ *and* $\trianglelefteq_i$ $(0 < i \leq n)$ *stands for* $=$, $\leqq^{\sharp}_m$ *or* $\prec^{\sharp}_m$ $(\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}, \mathsf{pl}\}$, $m > 0)$*. By a GOC of a formula* φ*, we mean a GOC of* $\mathcal{T}(\varphi)$*, the set of terms in* φ*.*

Example 4.3 (Gap Order Completion). *A possible GOC of* $\varphi(x, y, z) : x \prec^{\mathsf{w}}_9 y \wedge x \prec^{\mathsf{pl}} z \wedge z \prec^{\mathsf{w}} y$ *is* $x \prec^{\mathsf{pl}}_5 z \prec^{\mathsf{w}}_4 y$.

However, gap order completions are not sufficient. It is quite clear to see $(\exists x : \mathbb{T})[u \prec^{\mathsf{w}} x \prec^{\mathsf{p}} v]$ implies $u \prec^{\mathsf{w}}_2 v$. But for the converse to hold, $v \neq \mathsf{0}^{\mathsf{w}}_{(v^{\mathsf{w}})}$ is required. As another example, $(\exists x : \mathbb{T})[u \prec^{\mathsf{p}} x \prec^{\mathsf{p}} v]$ implies $u \prec^{\mathsf{p}}_2 v$, but not vice versa, because there may exist only two p-intervals at weight $\mathsf{w}(u)$ while $(\exists x : \mathbb{T})[u \prec^{\mathsf{p}} x \prec^{\mathsf{p}} v]$ requires that u, x and v all be in different p-intervals. In order to preserve equivalence, intuitively, we need to "delineate" a GOC to make sure ordinary terms in different *intervals* (a notion to be define precisely soon) are not related in any gap orders.

Example 4.4 (Order Arrangement). *Figure 4-1 shows an order arrangement of the linear order*

$$x_1 \prec^{\mathsf{w}}_{n_1} x_2 \prec^{\mathsf{p}}_{n_2} x_3 \prec^{\mathsf{l}}_{n_3} x_4 \ .$$

The weight of x_1 *is strictly lower than that of* x_2, x_3, *and* x_4*. The weight of* x_2, x_3, *and* x_4 *is the same, but the precedence of* x_2 *is lower than that of* x_3 *and* x_4*. Finally,* x_3 *is smaller than* x_4 *in the lexicographic order. We call a maximal list of elements with the same weight a* w*-interval, and similarly a maximal list of elements with the same weight and precedence order a* p*-interval. Thus, the second* w*-interval above has two inner* p*-intervals.*

We want to avoid relating ordinary elements at different levels in different intervals. Therefore we augment the gap order completion with boundary terms, called a delineated gap order completion.

Definition 4.12 (Delineated Gap Order Completion). *A delineated gap order completion (DGOC) of a set of terms S is a GOC of S in which if there occurs the following pattern* $v_1 \trianglelefteq^{\sharp}_{n_1} u \trianglelefteq^{\natural}_{n_2} v_2$*, where* $n_1, n_2 > 0$*,* $\trianglelefteq$ *stands for either* $\prec$ *or* $\leqq$*,* $\sharp, \natural \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}, \mathrm{pl}\}$*, and u is an ordinary term in* $\mathscr{L}_{\mathrm{kb}}$*, then either* $\sharp \equiv \natural \equiv \mathrm{pl}$ *or* $\sharp \equiv \natural \equiv \mathsf{l}$*. I.e., ordinary terms do not delineate two intervals unless they are asserted equal to boundary terms. By a DGOC of a formula* φ*, we mean a DGOC of* $\mathcal{T}(\varphi)$*, the set of terms in* φ*.*

We assume in a DGOC that both the smallest element and the largest element are boundary terms.

Example 4.5 (Delineated Gap Order Completion). *Revisit Example 4.3. A possible DGOC of* $\varphi(x, y, z)$ *is*

$$\varphi'(x,y,z) \;:\; \underbrace{0^{\mathsf{w}}_{(x^{\mathsf{w}})} \prec^{\mathrm{pl}}_{1} x \prec^{\mathrm{pl}}_{5} z \prec^{\mathrm{pl}}_{2} 1^{\mathsf{w}}_{(x^{\mathsf{w}})}}_{\mathsf{w}\text{-interval}} \prec^{\mathsf{w}}_{1} \underbrace{0^{\mathsf{w}}_{(y^{\mathsf{w}})} \prec^{\mathrm{pl}}_{1} y \prec^{\mathrm{pl}}_{1} 1^{\mathsf{w}}_{(y^{\mathsf{w}})}}_{\mathsf{w}\text{-interval}} \;.$$

We have

$$(\exists z : \mathbb{T})\, \varphi'(x,y,z) \;\leftrightarrow\; 0^{\mathsf{w}}_{(x^{\mathsf{w}})} \prec^{\mathrm{pl}}_{1} x \prec^{\mathrm{pl}}_{7} 1^{\mathsf{w}}_{(x^{\mathsf{w}})} \prec^{\mathsf{w}}_{1} 0^{\mathsf{w}}_{(y^{\mathsf{w}})} \prec^{\mathrm{pl}}_{1} y \prec^{\mathrm{pl}}_{1} 1^{\mathsf{w}}_{(y^{\mathsf{w}})} \;.$$

4.2.8 Lemmas

In this section we state a sequence of lemmas which will justify the elimination procedure given in the next section. These lemmas share the following common features.

1. They state the soundness of symbolic transformations for formulas in *primitive form*, a special prenex form where the prefix only consists of existential quantifiers and the matrix is a conjunction of literals;

2. A formula φ is transformed to a finite disjunction $\bigvee_i \varphi_i$ where for any i, φ_i is in primitive form and the matrix of φ_i, a conjunction of literals, contains no more open gap order literals than that of φ does. In the following whenever we say that a reduction does not introduce more open gap order literals, we always mean that it does not introduce more in each resulting conjunction of literals.

To save space, however, we omit these conditions in the description of each lemma.

The following lemma allows us to assume that a formula in $\mathscr{L}_{\mathrm{kb}^+}$ only contains *positive* order literals, *positive* tester literals and term equality literals.

Lemma 4.1 (Elimination of Negative Literals). *Assume formulas are type complete.*

1. *Any negative tester literal is equivalent to a disjunction of positive tester literals.*

2. *Any negative term equality literal (i. e., disequality between* TA*-terms) is equivalent to a disjunction of positive order literals.*

3. *Any negative order literal is equivalent to a disjunction of integer literals, tester literals and positive order literals.*

The following lemma allows us to assume that the matrix in a primitive formula is a delineated gap order completion.

Lemma 4.2 (Delineated Gap Order Completion). *Any conjunction of positive order literals in* $\mathscr{L}_{\mathrm{kb}^+}$ *can be effectively transformed to an equivalent finite disjunction of delineated gap order completions.*

In principle, boundary terms can appear in the weight function or in selectors, selector terms can occur in the weight function, and the weight function can be used to construct boundary terms. Repeating this process we can build more and more complex terms. The following lemma eliminates this superficial complication.

Lemma 4.3 (No Embedding of Boundary Terms). *Any formula in* $\mathscr{L}^{\mathsf{Z}}_{\mathrm{kb}^+}$ *can be effectively reduced to an equivalent formula in which no boundary terms appear inside selectors or the weight function.*

From now on, we assume that boundary terms are not properly embedded in other terms. The following lemma states that term gap order literals between a variable x of non-constant type α_p and another TA-term can be expressed using x's immediate descendants $\mathsf{s}_i^{\alpha_p} x$.

Lemma 4.4 (Reduction of Term Gap Order Literals). *Let*

$$\star \in \{ <_n^{\mathsf{kb}}, <_n^{\mathsf{w}}, <_n^{\mathsf{p}}, <_n^{\mathsf{l}}, <_n^{\mathsf{pl}}, \leqq_n^{\mathsf{kb}}, \leqq_n^{\mathsf{w}}, \leqq_n^{\mathsf{p}}, \leqq_n^{\mathsf{l}}, \leqq_n^{\mathsf{pl}} \} \qquad (n > 0) \ .$$

If x *is a* TA*-variable of type* α_p *with* $\alpha_p = (\mathsf{s}_1^{\alpha_p}, \ldots, \mathsf{s}_k^{\alpha_p})$ *and* t *is an arbitrary* TA*-term, then* $x \star t$ $(t \star x)$ *can be effectively reduced to an equivalent quantifier-free formula* $\varphi(\mathsf{s}_1^{\alpha_p} x, \ldots, \mathsf{s}_k^{\alpha_p} x)$ *(in* $\mathscr{L}^{\mathsf{Z}}_{\mathrm{kb}^+}$*) where* x *only occurs in* $\mathsf{s}_i^{\alpha_p} x$ $(0 < i \leq k)$*.*

By this lemma we can always assume that all *term occurrences* of a TA-variable have the same depth, and hence we are able to reduce them all together to depth 0. As we mentioned before, this is the main battlefield of quantifier elimination. To streamline the proof, we introduce the following three lemmas. The first one takes care of closed term literals generated during the reduction.

Lemma 4.5 (Reduction of Closed Term Literals). *Let*

$$\star \in \{ =, <_n^{\mathsf{kb}}, <_n^{\mathsf{w}}, <_n^{\mathsf{p}}, <_n^{\mathsf{l}}, <_n^{\mathsf{pl}}, \leqq_n^{\mathsf{kb}}, \leqq_n^{\mathsf{w}}, \leqq_n^{\mathsf{p}}, \leqq_n^{\mathsf{l}}, \leqq_n^{\mathsf{pl}} \} \qquad (n > 0) \ .$$

If $u \star v$ *is closed, i. e., both* u *and* v *are boundary terms, then it can be effectively reduced to an equivalent Presburger formula.*

The second one solves the case where no term occurrence of x appears in the other side of the gap order literal.

Lemma 4.6 (Reduction of Non-closed Term Gap Order Literals). *Let*

$$\star \in \{ <_n^{\mathsf{kb}}, <_n^{\mathsf{w}}, <_n^{\mathsf{p}}, <_n^{\mathsf{l}}, <_n^{\mathsf{pl}}, \leqq_n^{\mathsf{kb}}, \leqq_n^{\mathsf{w}}, \leqq_n^{\mathsf{p}}, \leqq_n^{\mathsf{l}}, \leqq_n^{\mathsf{pl}} \} \qquad (n > 0) \ .$$

If x is a TA*-variable of type α_p with $\alpha_p = (\mathsf{S}_1^{\alpha_p}, \ldots, \mathsf{S}_k^{\alpha_p})$ and t is either a boundary term or an ordinary* TA*-term* not *containing x, then $x \star t$ ($t \star x$) can be effectively reduced to an equivalent quantifier-free formula $\varphi(\mathsf{S}_1^{\alpha_p} x, \ldots, \mathsf{S}_k^{\alpha_p} x)$ where x only occurs in $\mathsf{S}_i^{\alpha_p} x$ $(0 < i \leq k)$.*

This lemma in fact deals with the most sophisticated part of the reduction that eventually comes down to the success of reducing relations between tuples of the same weight, as is stated by the third lemma.

Lemma 4.7 (Reduction of Tuple Literals). *Let U, V be k-tuples of the same weight, and*

$$\star \in \{ =, <_n^{k;\mathsf{kb}}, <_n^{k;\mathsf{w}}, <_n^{k;\mathsf{p}}, <_n^{k;\mathsf{l}}, <_n^{k;\mathsf{pl}}, \leqq_n^{k;\mathsf{kb}}, \leqq_n^{k;\mathsf{w}}, \leqq_n^{k;\mathsf{p}}, \leqq_n^{k;\mathsf{l}}, \leqq_n^{k;\mathsf{pl}} \} \qquad (k > 0, n > 0) \ .$$

1. *If $U = \langle u_1, \ldots, u_k \rangle$ is an ordinary tuple, then $U \star V$ ($V \star U$) can be effectively reduced to an equivalent quantifier-free formula $\varphi(u_1, \ldots, u_k)$ (in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$) in which u_i $(0 < i \leq k)$ does not occur inside selectors.*
2. *If $U \star V$ ($V \star U$) is a closed tuple, i. e., both U and V are boundary tuples, then it can be effectively reduced to an equivalent Presburger formula.*

We never need to reduce open or half-open term equality literals, because, as shown below, we can carry out elimination on $(\exists x)[x = t \wedge \varphi(x)]$ even if t contains x.

Lemma 4.8 (Elimination of Term Quantifiers). *Let x be a term variable, $\varphi_{\mathrm{kb}^+}(x)$ a conjunction of literals in $\mathscr{L}_{\mathrm{kb}^+}$ with $\mathrm{depth}^{\varphi_{\mathrm{kb}^+}}(x) = 0$, and $\varphi_{\mathbb{Z}}(x)$ a Presburger formula in which x occurs inside the weight function. Then*

$$(\exists x : \mathbb{T}) \Big[\varphi_{\mathrm{kb}^+}(x) \ \wedge \ \varphi_{\mathbb{Z}}(x) \Big]$$

can be effectively reduced to $\varphi'_{\mathrm{kb}^+} \wedge \varphi'_{\mathbb{Z}}$ in which x does not occur and φ'_{kb^+} is quantifier-free.

In fact term quantifiers are reduced to integer quantifiers that can be eliminated according to the following lemma.

Lemma 4.9 (Elimination of Integer Quantifiers). *Let z be an integer variable, $\varphi_{\mathrm{kb}^+}(z)$ a conjunction of literals in $\mathscr{L}_{\mathrm{kb}^+}$ where z occurs inside boundary terms, and $\varphi_{\mathbb{Z}}(z)$ a Presburger formula. Then*

$$(\exists z : \mathbb{Z}) \Big[\varphi_{\mathrm{kb}^+}(z) \ \wedge \ \varphi_{\mathbb{Z}}(z) \Big]$$

can be effectively reduced to $\varphi'_{\mathrm{kb}^+} \wedge \varphi'_{\mathbb{Z}}$ *where no* z *occurs and* φ'_{kb^+} *is quantifier-free.*

4.3 Quantifier Elimination for $\mathrm{Th}(\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+})$

In this section we present a quantifier elimination procedure for $\mathrm{Th}(\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+})$ by extending the elimination procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$ ([64]), which is in turn based on the elimination procedure for Th(TA) [19]. We first recall the following notation conventions.

1. *Primitive Form.* All transformations are carried out on formulas of the form (1.1). Each step of the transformations manipulates (1.1) to produce a version of the same form (or multiple versions of the same form in case disjunctions are introduced).

2. *Positive Literals.* All order literals have positive occurrence. This is guaranteed by Lemma 4.1.

3. *Nondeterminism.* Whenever we say "guess ψ", we mean split on a finite disjunction $\bigvee_i \varphi_i$, which is valid in the context and contains ψ as a disjunct.

4. *Type Completeness.* All formulas are type-complete. But we omit listing tester literals unless they are needed for the correctness proof.

4.3.1 The Elimination Procedure

The elimination procedure consists of the following two algorithms:

Algorithm 4.1 (Elimination of Integer Quantifiers in $\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+}$). *We assume that formulas with quantifiers on integer variables are in the form*

$$(\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big[\ \varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \wedge\ \varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \Big] \ , \tag{4.5}$$

where $\bar{\mathbf{y}}$, $\bar{\mathbf{z}}$ *are integer variables,* $\bar{\mathbf{x}}$ *are term variables. Note that* $\bar{\mathbf{x}}$ *may occur inside the weight function in* $\varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ *and* $\bar{\mathbf{y}}, \bar{\mathbf{z}}$ *may appear inside boundary terms in* $\varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$.

Repeatedly apply the following steps (1) *and* (2) *to* (4.5) *until* $\bar{\mathbf{z}} = \emptyset$.

1. *If none of* $\bar{\mathbf{z}}$ *appears inside any boundary terms, then* $\varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$ *is just* $\varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}})$*, which can be moved out of* $\exists \bar{\mathbf{z}}$*. We then obtain*

$$(\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big[\ \varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ \wedge\ \varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \ .$$

Since $(\exists \bar{\mathbf{z}} : \mathbb{Z})[\varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})]$ *is in* $\mathscr{L}_{\mathbb{Z}}$*, we can proceed to remove the block of existential quantifiers using Cooper's method [12, 46]. As the elimination of integer quantifiers in* $\mathrm{TA}_{\mathbb{Z}}$*, we can defer the actual elimination on* $(\exists \bar{\mathbf{z}} : \mathbb{Z})[\varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})]$ *until all term quantifiers have been eliminated. The reason*

is the same, that the elimination of term quantifiers does not require the integer constraint in (4.6) *to be quantifier-free.*

2. *If for some $z \in \bar{\mathbf{z}}$, z occurs inside some boundary terms, we set*

$$\varphi'_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}} \setminus z) \ \wedge \ \varphi'_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}} \setminus z) \ := \ \textsc{eliminate-int}\Big((\exists z : \mathbb{Z}) \Big[\varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \wedge \ \varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \Big) \ ,$$
$$(\exists \bar{\mathbf{z}} : \mathbb{Z}) \Big[\varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \wedge \ \varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ := \ \Big(\exists (\bar{\mathbf{z}} \setminus z) : \mathbb{Z} \Big) \Big[\varphi'_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}} \setminus z) \ \wedge \ \varphi'_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}} \setminus z) \Big] \ .$$

The existence of eliminate-int *is guaranteed by Lemma 4.9.*

Algorithm 4.2 (Elimination of Term Quantifiers in $\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+}$). *We assume that formulas with quantifiers on term variables are in the form*

$$(\exists \bar{\mathbf{x}} : \mathbb{T}) \Big[\varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \wedge \ \varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ , \tag{4.6}$$

where $\bar{\mathbf{x}}$, $\bar{\mathbf{y}}$ are term variables, $\bar{\mathbf{z}}$ are integer variables. Note that $\bar{\mathbf{z}}$ may occur inside boundary terms in $\varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$, and $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ may occur inside the weight function in $\varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})$.

Repeatedly apply the following steps (1) *and* (2) *to* (4.6) *until $\bar{\mathbf{x}} = \emptyset$.*

1. Depth Reduction. *If $(\forall x \in \bar{\mathbf{x}})\ \mathrm{depth}^{\varphi_{\mathrm{kb}^+}}(x) > 0$.*

   (a) Variable Selection. *Select a α-typed variable $x \in \bar{\mathbf{x}}$ for some $\alpha = (\mathsf{s}^\alpha_1, \ldots, \mathsf{s}^\alpha_{\mathrm{ar}(\alpha)})$. This selection is always possible as $\mathrm{depth}^{\varphi_{\mathrm{kb}^+}}(x) > 0$. We require that in the next run of* (1a)*, we choose one of the variables generated by* this *run of* (1b)*. I.e., the variable selection is done in depth-first manner. This is crucial to guarantee that a run eventually leaves* (1)*. Let $\bar{\mathbf{x}}' \equiv \bar{\mathbf{x}} \setminus x$.*

   (b) Decomposition. *We rewrite* (4.6) *to:*

$$\Big(\exists \bar{\mathbf{x}}', x_1, \ldots, x_{\mathrm{ar}(\alpha)}, x : \mathbb{T}\Big) \Big[\mathrm{Is}_\alpha(x) \ \wedge \bigwedge_{0<i\leq \mathrm{ar}(\alpha)} \mathsf{s}^\alpha_i(x) = x_i \ \wedge \ \varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \wedge \ \varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ . \tag{4.7}$$

   (c) Simplification. *Exhaustively apply the following simplification rules to φ_{kb^+} and $\varphi_{\mathbb{Z}}$ in* (4.7)*:*

   i. replace $\mathsf{s}^\alpha_i(x)$ by x_i $(0 < i \leq \mathrm{ar}(\alpha))$;

   ii. replace x^{w} by $\mathsf{w}(\alpha) + \sum_{i=1}^{\mathrm{ar}(\alpha)} x_i^{\mathsf{w}}$;

   iii. replace $x \prec^\sharp_n t$ by depth-reduction *($x \prec^\sharp_n t$); similar for $t \prec^\sharp_n x$, $x \leqq^\sharp_n t$ and $t \leqq^\sharp_n x$.*

   The existence of depth-reduction *follows from Lemma 4.4. Let the resulting formula be*

$$\Big(\exists \bar{\mathbf{x}}', x_1, \ldots, x_{\mathrm{ar}(\alpha)}, x : \mathbb{T}\Big) \Big[\mathrm{Is}_\alpha(x) \ \wedge \bigwedge_{0<i\leq \mathrm{ar}(\alpha)} \mathsf{s}^\alpha_i(x) = x_i \ \varphi'_{\mathrm{kb}^+}(\bar{\mathbf{x}}', \mathsf{s}^\alpha_1(x), \ldots, \mathsf{s}^\alpha_{\mathrm{ar}(\alpha)}(x), \bar{\mathbf{y}}, \bar{\mathbf{z}}) \ \wedge \ \varphi'_{\mathbb{Z}}(\bar{\mathbf{x}}', \mathsf{s}^\alpha_1(x), \ldots, \mathsf{s}^\alpha_{\mathrm{ar}(\alpha)}(x), \bar{\mathbf{y}}, \bar{\mathbf{z}}) \Big] \ . \tag{4.8}$$

Input: $(\exists \bar{\mathbf{x}} : \mathbb{T})\,[\ \varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \wedge\ \varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ ]$.
while $\bar{\mathbf{x}} \neq \emptyset$ **do**
  if $(\forall x \in \bar{\mathbf{x}})\,\mathrm{depth}^{\varphi_{\mathrm{kb}^+}}(x) > 0$ **then**
    (1) *Depth Reduction*
      (1a) Variable Selection
      (1b) Decomposition
      (1c) Simplification
  else $\{(\exists x \in \bar{\mathbf{x}})\,\mathrm{depth}^{\varphi_{\mathrm{kb}^+}}(x) = 0\}$
    (2) *Elimination*
  end if
end while

Figure 4-2: Quantifier Elimination in $\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+}$.

It is now clear that if x occurs in φ'_{kb^+} and $\varphi'_{\mathbb{Z}}$ it occurs inside some of $\mathsf{s}^{\alpha}_1(x), \ldots, \mathsf{s}^{\alpha}_{\mathrm{ar}(\alpha)}(x)$. Since

$$\Big(\forall x_1, \ldots, x_{\mathrm{ar}(\alpha)} : \mathbb{T}\Big)\Big(\exists x : \mathbb{T}\Big)\Big[\,\mathrm{Is}_{\alpha}(x)\ \wedge \bigwedge_{0<i\leq \mathrm{ar}(\alpha)} \mathsf{s}^{\alpha}_i(x) = x_i\,\Big]$$

is valid in TA*, we can replace in* (4.8), $\mathsf{s}^{\alpha}_1(x), \ldots, \mathsf{s}^{\alpha}_{\mathrm{ar}(\alpha)}(x)$*, respectively, by $x_1, \ldots, x_{\mathrm{ar}(\alpha)}$, and hence remove $\bigwedge_{0<i\leq \mathrm{ar}(\alpha)} \mathsf{s}^{\alpha}_i(x) = x_i$, $\mathrm{Is}_{\alpha}(x)$ together with $\exists x$, obtaining*

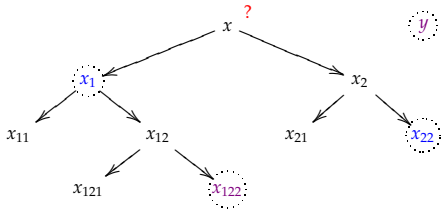
$$\Big(\exists\, \bar{\mathbf{x}}', x_1, \ldots, x_{\mathrm{ar}(\alpha)} : \mathbb{T}\Big)\ \Big[\ \varphi'_{\mathrm{kb}^+}(\bar{\mathbf{x}}', x_1, \ldots, x_{\mathrm{ar}(\alpha)}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \wedge\ \varphi'_{\mathbb{Z}}(\bar{\mathbf{x}}', x_1, \ldots, x_{\mathrm{ar}(\alpha)}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \Big]\ . \quad (4.9)$$

2. Elimination. *If* $(\exists x \in \bar{\mathbf{x}})\,\mathrm{depth}^{\varphi_{\mathrm{kb}^+}}(x) = 0$.
Take the x as in the guard condition, guess a DGOC for all terms related with x in gap order literals (by Lemma 4.2) and then eliminate x by Lemma 4.8. Formally we set

$$\varphi'_{\mathrm{kb}^+}(\bar{\mathbf{x}} \setminus x, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \wedge\ \varphi'_{\mathbb{Z}}(\bar{\mathbf{x}} \setminus x, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ :=\ \textsc{eliminate-term}\Big(\ (\exists x : \mathbb{T})\,\Big[\varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \wedge\ \varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\Big]\ \Big)\ ,$$
$$\Big(\exists \bar{\mathbf{x}} : \mathbb{T}\Big)\Big[\ \varphi_{\mathrm{kb}^+}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \wedge\ \varphi_{\mathbb{Z}}(\bar{\mathbf{x}}, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \Big]\ :=\ \Big(\exists(\bar{\mathbf{x}} \setminus x) : \mathbb{T}\Big)\Big[\ \varphi'_{\mathrm{kb}^+}(\bar{\mathbf{x}} \setminus x, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \wedge\ \varphi'_{\mathbb{Z}}(\bar{\mathbf{x}} \setminus x, \bar{\mathbf{y}}, \bar{\mathbf{z}})\ \Big]\ .$$

For the ease of understanding Algorithm 4.2, we show its high-level control-flow in Figure 4-2. It is now easily seen that Algorithm 4.2 is a greedy algorithm in the sense that it tries to do *Elimination* (step (2)) as soon as the *elimination condition* holds, that is, all term occurrences of this variable are of depth 0. Otherwise, the algorithm tries to create the elimination condition using *Depth Reduction* (step (1)) which includes three sequential sub-procedures: Variable Selection (step (1a)), Decomposition (step (1b)) and Simplification (step (1c)). We require that Variable Selection be done in depth-first manner. As all depths are finite, this guarantees that a run eventually leaves *Depth Reduction* and enter *Elimination*.

Lemma 4.10 (Soundness of Algorithms 4.1). *Each transformation step in Algorithms 4.1 and 4.2 preserves equivalence.*



$x : x$, $y : y$,
$x_1 : \mathrm{car}(x)$, $x_2 : \mathrm{cdr}(x)$,
$x_{21} : \mathrm{car}(\mathrm{car}(x))$, $x_{22} : \mathrm{cdr}(\mathrm{cdr}(x))$,
$x_{11} : \mathrm{car}(\mathrm{car}(x))$, $x_{12} : \mathrm{cdr}(\mathrm{car}(x))$,
$x_{121} : \mathrm{car}(\mathrm{cdr}(\mathrm{car}(x)))$, $x_{122} : \mathrm{cdr}(\mathrm{cdr}(\mathrm{car}(x)))$.

Figure 4-3: Quantifier Elimination on $(\exists x)\,[\mathrm{car}(x) \prec^{\mathrm{I}}_2 \mathrm{cdr}(\mathrm{cdr}(x)) \ \wedge\ \mathrm{cdr}(\mathrm{cdr}(\mathrm{car}(x))) \prec^{\mathrm{I}}_3 y]$.

Proof. Equivalence preservation has been assured by the lemmas in Section 4.2.8. More precisely, Lemma 4.9 justifies the soundness of Algorithm 4.1. For Algorithm 4.2, Axioms (1)-(6) in Section 2.2 justify step (1b); Lemma 4.4, with the help of Lemmas 4.5, 4.6 and 4.7, justifies step (1c); Lemma 4.8 justifies step (2). □

Lemma 4.11 (Termination). *Both Algorithm 4.1 and Algorithm 4.2 terminate.*

Theorem 4.1 (Decidability). $\mathrm{Th}(\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+})$ *is decidable, and hence so is* $\mathrm{Th}(\mathrm{TA}_{\mathrm{kb}})$.

Proof. By Lemmas 4.10 and 4.11. □

4.3.2 An Example

Example 4.6 (Quantifier Elimination in $\mathrm{List}^{\mathbb{Z}}_{\mathrm{kb}^+}$). *Let us go through an example with emphasis on the depth reduction. We only show* one *simple trace of the reduction. Consider in* $\mathrm{List}^{\mathbb{Z}}_{\mathrm{kb}^+}$ *the following formula*

$$(\exists x)\Big[\ \mathrm{car}(x) \prec^{\mathrm{I}}_2 \mathrm{cdr}(\mathrm{cdr}(x)) \ \wedge\ \mathrm{cdr}(\mathrm{cdr}(\mathrm{car}(x))) \prec^{\mathrm{I}}_3 y\ \Big]\ , \tag{4.10}$$

where $\mathrm{depth}^{(4.10)}(x) = 3$. *Figure 4-3 shows the DAG of* (4.10) *which is sibling complete. Formula* (4.10) *states that there exists a tree x whose descendants* x_1, x_{22}, x_{122} *and a parameter y satisfy certain order relations.*

At the first run of step (1)*, we introduce fresh variables* x_1 *and* x_2 *to replace* $\mathrm{car}(x)$ *and* $\mathrm{cdr}(x)$*, respectively. By a standard quantifier manipulation we obtain*

$$(\exists x_1 \exists x_2)\Big[\ x_1 \prec^{\mathrm{I}}_2 \mathrm{cdr}(x_2) \ \wedge\ \mathrm{cdr}(\mathrm{cdr}(x_1)) \prec^{\mathrm{I}}_3 y\ \Big]\ , \tag{4.11}$$

where $\mathrm{depth}^{(4.11)}(x_1) = 2$ *and* $\mathrm{depth}^{(4.11)}(x_2) = 1$*, both less than* $\mathrm{depth}^{(4.10)}(x)$*. In the second run of step* (1)*, we pick* x_1 *and replace* $x_1 \prec^{\mathrm{I}}_2 \mathrm{cdr}(x_2)$ *by* $\mathrm{car}(x_1) = \mathrm{car}(\mathrm{cdr}(x_2)) \wedge \mathrm{cdr}(x_1) \prec^{\mathrm{I}}_2 \mathrm{cdr}(\mathrm{cdr}(x_2))$ *(which is one of several choices). We obtain*

$$(\exists x_2 \exists x_{11} \exists x_{12})\Big[\ x_{11} = \mathrm{car}(\mathrm{cdr}(x_2)) \ \wedge\ x_{12} \prec^{\mathrm{I}}_2 \mathrm{cdr}(\mathrm{cdr}(x_2)) \ \wedge\ \mathrm{cdr}(x_{12}) \prec^{\mathrm{I}}_3 y\ \Big]\ . \tag{4.12}$$

At this point we have $\mathrm{depth}^{(4.12)}(x_{11}) = 0$ *and the run enters step* (2)*. In this case we can immediately remove* $\exists x_{11}$*, obtaining*

$$(\exists x_2 \exists x_{12}) \left[\; x_{12} \prec_2^{\mathsf{I}} \mathrm{cdr}(\mathrm{cdr}(x_2)) \;\wedge\; \mathrm{cdr}(x_{12}) \prec_3^{\mathsf{I}} y \; \right] , \tag{4.13}$$

where $\mathrm{depth}^{(4.13)}(x_{12}) = 1$ *and* $\mathrm{depth}^{(4.13)}(x_2) = 2$*. At the third run of step* (1)*, we select* x_{12}*. The run could give us*

$$(\exists x_2 \exists x_{121} \exists x_{122}) \left[\; x_{121} = \mathrm{car}(\mathrm{cdr}(\mathrm{cdr}(x_2))) \;\wedge\; x_{122} \prec_2^{\mathsf{I}} \mathrm{cdr}(\mathrm{cdr}(x_2)) \;\wedge\; x_{122} \prec_3^{\mathsf{I}} y \; \right] , \tag{4.14}$$

which as before by step (2) *simplifies to*

$$(\exists x_2 \exists x_{122}) \left[\; x_{122} \prec_2^{\mathsf{I}} \mathrm{cdr}(\mathrm{cdr}(x_2)) \;\wedge\; x_{122} \prec_3^{\mathsf{I}} y \; \right] . \tag{4.15}$$

Still we have $\mathrm{depth}^{(4.15)}(x_{122}) = 0$ *which justifies another run of step* (2)*. Let us take a gap order completion* $x_{122} \prec_2^{\mathsf{I}} \mathrm{cdr}(\mathrm{cdr}(x_2)) \prec_1^{\mathsf{I}} y$ *(which again is just one of many choices) and rewrite* (4.15) *to*

$$(\exists x_2 \exists x_{122}) \left[\; x_{122} \prec_2^{\mathsf{I}} \mathrm{cdr}(\mathrm{cdr}(x_2)) \prec_1^{\mathsf{I}} y \; \right] . \tag{4.16}$$

With the help of boundary functions, (4.16) *reduces to*

$$(\exists x_2) \left[\; 0^{\mathsf{w}}_{((\mathrm{cdr}(\mathrm{cdr}(x_2)))^{\mathsf{w}})} \prec_2^{\mathsf{I}} \mathrm{cdr}(\mathrm{cdr}(x_2)) \prec_1^{\mathsf{I}} y \; \right] . \tag{4.17}$$

The fourth and the fifth runs of step (1) *(with the same trick of quantifier manipulation) give us*

$$(\exists x_{222}) \left[\; 0^{\mathsf{w}}_{(x^{\mathsf{w}}_{222})} \prec_2^{\mathsf{I}} x_{222} \prec_1^{\mathsf{I}} y \; \right] . \tag{4.18}$$

After that the run comes back again to step (2) *as* $\mathrm{depth}^{(4.18)}(x_{222}) = 0$*. Here we have to reduce term quantifiers to integer quantifiers in that* x_{222} *also appears in boundary terms. By Lemma 4.8,* (4.18) *is equivalent to*

$$(\exists z) \left[\; 0^{\mathsf{w}}_{(z)} \prec_3^{\mathsf{I}} y \;\wedge\; \mathrm{Tree}^{\mathrm{cons}}(z) \; \right] , \tag{4.19}$$

which simplifies to $0^{\mathsf{w}}_{(y^{\mathsf{w}})} \prec_3^{\mathsf{I}} y \wedge \mathrm{Tree}^{\mathrm{cons}}(y^{\mathsf{w}})$*, and in turn to*

$$0^{\mathsf{w}}_{(y^{\mathsf{w}})} \prec_3^{\mathsf{I}} y \;, \tag{4.20}$$

as $0^{\mathsf{w}}_{(y^{\mathsf{w}})} \prec_3^{\mathsf{I}} y$ *implies* $\mathrm{Tree}^{\mathrm{cons}}(y^{\mathsf{w}})$*. It is not hard to verify that* (4.20) *implies* (4.10) *as desired. (We do not have equivalence because this is just one trace of the reduction.) In fact, the above elimination procedure tells us how to obtain an solution to* (4.10) *provided* (4.20) *holds. Let us show the construction in bottom-up*

fashion. By $x_\sharp$, we refer to labels in Figure 4-3. First, we set x_{122} to $\mathsf{0}^{\mathsf{w}}_{(y^{\mathsf{w}})}$ to take care of $x_{122} \prec^{\mathsf{l}}_3 y$. Second, we find x_{11}, x_{121} and x_{22} to satisfy $x_1 \prec^{\mathsf{l}}_2 x_{22}$. This can always be done as there is no constraints on x_{11}, x_{121} and x_{22}. Third, we set x_{21} to an arbitrary value. In this way we obtain an x which is the solution to (4.10).

We note that the depth reduction of a variable is at the expense of increasing the depth of a term on the other side of a gap order predicate. This happens when φ_{kb^+} contains $x \star t$ (or $t \star x$) and $\mathrm{depth}^{\varphi_{\mathrm{kb}^+}}(x) > 0$. For example, from (4.11) to (4.12), the depth of x_2 increases by 1. Moreover, the depth reduction in general introduces more existential quantifiers and more equalities in at least one resulting formula (e.g., this also happens in the reduction from (4.11) to (4.12)). In every step of the transformation, however, the number of open gap order literals in each resulting primitive formula is no more than that in the original formula. Moreover, the final elimination procedure removes at least one open gap order literal if the eliminated variable occurs in such literals (e.g., from (4.16) to (4.17) and from (4.18) to (4.19)). When all open gap order literals are gone, the depths of terms will be strictly decreasing. This forces the run of Algorithm 4.2 to eventually leave step (1) and from then on to stay in step (2) until all existential quantifiers are removed.

4.4 Presence of a 0-weight Unary Function

Up to now we have assumed that the language does not contain a unary function f of weight 0. However, the presence of such a function considerably simplifies the reduction procedure. First we note that there exists no maximum ground TA-term $\mathsf{1}^{\mathsf{w}}_{(m)}$ of weight m for any $m > 0$ (when we do not consider the artificial fix using the sink value $\bot$), because otherwise we will have a contradiction as $\mathsf{1}^{\mathsf{w}}_{(m)} \prec^{\mathrm{kb}} f(\mathsf{1}^{\mathsf{w}}_{(m)})$ and $\mathsf{w}(\mathsf{1}^{\mathsf{w}}_{(m)}) = \mathsf{w}(f(\mathsf{1}^{\mathsf{w}}_{(m)}))$. As a consequence, $\prec^{\mathsf{w}}_n$ is *dense* in the sense that if $u \prec^{\mathsf{w}}_n v$ ($n > 0$), there are infinitely many ground TA-terms in between u and v with respect to $\prec^{\mathrm{kb}}$. Similarly, there exists no maximum ground TA-term $\mathsf{1}^{\mathsf{p}}_{(m,p)}$ of weight m and type α_p except when α_p is a constant and $m = \mathsf{w}(\alpha_p)$, and hence $\prec^{\mathsf{p}}_n$ is *dense* except when the left operand is a constant. Moreover, $\prec^{\mathsf{l}}_n$ is *dense* except when the order is resolved between the corresponding rightmost immediate subterms of its operands. To make the above statements precise, we present reductions for open gap order literals involving $\prec^{\mathsf{w}}_n$, $\prec^{\mathsf{p}}_n$, $\prec^{\mathsf{l}}_n$, $\preceqq^{\mathsf{w}}_n$, $\preceqq^{\mathsf{p}}_n$, and $\preceqq^{\mathsf{l}}_n$ ($n > 0$).

4.4.1 Reduction of $u \prec^{\mathsf{w}}_n v$ and $u \preceqq^{\mathsf{w}}_n v$

Clearly, $u \prec^{\mathsf{w}}_n v$ implies $u \prec^{\mathsf{w}} v$ and assuming $u \prec^{\mathsf{w}} v$, we have an infinite chain

$$u \prec^{\mathrm{pl}} f(u) \prec^{\mathrm{pl}} f^2(u) \prec^{\mathrm{pl}} \cdots \prec^{\mathrm{pl}} f^n(u) \prec^{\mathrm{pl}} \cdots \prec^{\mathsf{w}} v \ .$$

Therefore $u \prec^{\mathsf{w}}_n v$ simplifies to $u \prec^{\mathsf{w}} v$, which further simplifies to the integer literal $\mathsf{w}(u) < \mathsf{w}(v)$. Due to the existence of the infinite chain from u to v, $u \preceqq^{\mathsf{w}}_n v$ simplifies to false.

4.4.2 Reduction of $u \prec_n^{\mathsf{p}} v$ and $u \preceq_n^{\mathsf{p}} v$

Reduction of $u \prec_n^{\mathsf{p}} v$ and $u \preceq_n^{\mathsf{p}} v$ when u is not a constant. Let t be a proper subterm of u and we denote u by $u[t]$ to emphasize this occurrence of t. Let $u[t']$ be the term obtained from u by substituting t' for t. Assuming $u \prec^{\mathsf{p}} v$, we have an infinite increasing chain

$$u[t] \prec^{\mathsf{l}} u[f(t)] \prec^{\mathsf{l}} u[f^2(t)] \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} u[f^n(t)] \prec^{\mathsf{l}} \cdots \prec^{\mathsf{p}} v \ .$$

Therefore, $u \prec_n^{\mathsf{p}} v$ simplifies to $u \prec^{\mathsf{p}} v$, which is just $q < p$ assuming $\alpha_p = \mathrm{type}(u)$ and $\alpha_q = \mathrm{type}(v)$. For the same reason as before, $u \preceq_n^{\mathsf{p}} v$ simplifies to false.

Reduction of $u \prec_n^{\mathsf{p}} v$ and $u \preceq_n^{\mathsf{p}} v$ when u is not a constant. Let $\mathrm{type}(v) = \alpha_q$. We first check if there exists a non-constant ground TA-term $t[s]$ (where s is a proper subterm) such that $\mathsf{w}(t[s]) = \mathsf{w}(u)$ and $\mathrm{type}(t[s]) = \alpha_p$, and $u \prec^{\Sigma} \alpha_p \prec^{\Sigma} \alpha_q$. This condition can be formally expressed as

$$\bigvee_{q<p<u,\, \alpha_p \notin \mathcal{A}} \mathrm{Tree}^{\alpha_p}(u^{\mathsf{w}}) \ , \tag{4.21}$$

whose truth value can be determined statically given a fixed signature. If (4.21) holds, we have an infinite chain

$$u \prec^{\mathsf{p}} t[s] \prec^{\mathsf{l}} t[f(s)] \prec^{\mathsf{l}} t[f^2(s)] \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} t[f^n(s)] \prec^{\mathsf{l}} \cdots \prec^{\mathsf{p}} v \ . \tag{4.22}$$

Therefore $u \prec_n^{\mathsf{p}} v$ simplifies to true and $u \preceq_n^{\mathsf{p}} v$ simplifies to false. If (4.21) does not hold, then all ground TA-terms in between u and $0^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)}$ must be constants. Let us assume there are $m-1$ constants in between u and $0^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)}$. If $m > n$, then $u \prec_n^{\mathsf{p}} v$ simplifies to true while $u \preceq_n^{\mathsf{p}} v$ simplifies to false. If $m = n$, then both $u \prec_n^{\mathsf{p}} v$ and $u \preceq_n^{\mathsf{p}} v$ simplify to true. If $m < n$, then $u \prec_n^{\mathsf{p}} v$ simplifies to $0^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} \prec^{\mathsf{l}}_{n-m} v$ while $u \preceq_n^{\mathsf{p}} v$ simplifies to $0^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} \preceq^{\mathsf{l}}_{n-m} v$. We need further consider three subcases under the assumption that $m < n$.

1. v is a constant. Then $0^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} = v$, and so both $u \prec_n^{\mathsf{p}} v$ and $u \preceq_n^{\mathsf{p}} v$ simplify to false.

2. v is of type f. Since (4.21) does not hold, $0^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)}$ must be $f(c_1)$ where c_1 is the smallest constant of weight $\mathsf{w}(v)$. Suppose that there are r constants of weight $\mathsf{w}(v)$ and they are ordered as

$$c_1 \prec^{\mathrm{kb}} c_2 \prec^{\mathrm{kb}} \cdots \prec^{\mathrm{kb}} c_r \ .$$

   Then v must be of the form $f^l(c_{r'})$ $(0 < r' \leq r)$, where f^l denotes l applications of f. We have

the following *tight* chain

$$\begin{aligned} & f(c_1) \prec^{\mathsf{kb}} \cdots \prec^{\mathsf{kb}} f(c_{r-1}) \prec^{\mathsf{kb}} f(c_r) \\ \prec^{\mathsf{kb}} & f^2(c_1) \prec^{\mathsf{kb}} \cdots \prec^{\mathsf{kb}} f^2(c_{r-1}) \prec^{\mathsf{kb}} f^2(c_r) \\ \prec^{\mathsf{kb}} & \cdots\cdots \\ \prec^{\mathsf{kb}} & f^l(c_1) \prec^{\mathsf{kb}} \cdots \prec^{\mathsf{kb}} f^l(c_{r'-1}) \prec^{\mathsf{kb}} f^l(c_{r'}) \ . \end{aligned}$$

So $\mathsf{0}^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} \prec^{\mathsf{l}}_{n-m} v$ is true if and only if $(l-1)r + (r'-1) \geq n-m$. The reduct can be formally expressed as

$$\bigvee_{1 \leq r' \leq r} \mathrm{Is}_f(v) \ \wedge \ \mathrm{Is}_f(\mathsf{s}^f v) \ \wedge \ \cdots \ \wedge \ \mathrm{Is}_f((\mathsf{s}^f)^{l(r')-1} v) \ \wedge \ c_{r'} = (\mathsf{s}^f)^{l(r')} v \ , \tag{4.23}$$

where $l(r')$ is the minimum integer such that $(l(r')-1)r + (r'-1) \geq n-m$. A similar simplification holds for $\mathsf{0}^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} \preceq^{\mathsf{l}}_{n-m} v$ except that we require that $(l(r')-1)r + (r'-1) = n-m$.

3. v is not a constant and not of type f. Let us assume $\mathrm{ar}(\alpha_q) = k > 0$. Let $\langle l_1, \ldots, l_k \rangle$ be the smallest k-integer tuple of weight $\mathsf{w}(v) - \mathsf{w}(\alpha_q)$ (with respect to the lexicographic extension of the integer ordering) such that $\mathrm{Tree}(l_i)$ holds for all $i \in [1,k]$. We first assume that for some $k' \in [1,k]$,

$$\bigwedge_{1 \leq i \leq k'-1} (\mathsf{s}_i^{\alpha_q} v)^{\mathsf{w}} = l_i \ \wedge \ (\mathsf{s}_{k'}^{\alpha_q} v)^{\mathsf{w}} \neq l_{k'} \ . \tag{4.24}$$

In fact we must have $(\mathsf{s}_{k'}^{\alpha_q} v)^{\mathsf{w}} > l_{k'}$ because $\langle l_1, \ldots, l_k \rangle$ is the smallest such k-tuples. Then we have $\mathsf{0}^{\mathsf{w}}_{(l_{k'})} \prec^{\mathsf{w}} \mathsf{s}_{k'}^{\alpha_q} v$, and hence $f^n(\mathsf{0}^{\mathsf{w}}_{(l_{k'})}) \prec^{\mathsf{w}} \mathsf{s}_{k'}^{\alpha_q} v$ for any $n > 0$. Therefore we have an infinite chain

$$\begin{aligned} \mathsf{0}^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} & \preceq^{\mathsf{l}} \alpha_q(\mathsf{0}^{\mathsf{w}}_{(i_1)}, \ldots, \mathsf{0}^{\mathsf{w}}_{(i_{k'})}, \ldots, \mathsf{0}^{\mathsf{w}}_{(i_k)}) \\ & \prec^{\mathsf{l}} \alpha_q(\mathsf{0}^{\mathsf{w}}_{(i_1)}, \ldots, f(\mathsf{0}^{\mathsf{w}}_{(i_{k'})}), \ldots, \mathsf{0}^{\mathsf{w}}_{(i_k)}) \\ & \prec^{\mathsf{l}} \alpha_q(\mathsf{0}^{\mathsf{w}}_{(i_1)}, \ldots, f^2(\mathsf{0}^{\mathsf{w}}_{(i_{k'})}), \ldots, \mathsf{0}^{\mathsf{w}}_{(i_k)}) \\ & \prec^{\mathsf{l}} \cdots\cdots \\ & \prec^{\mathsf{l}} \alpha_q(\mathsf{s}_1^{\alpha_q} v, \ldots, \mathsf{s}_{k'}^{\alpha_q} v, \ldots, \mathsf{s}_k^{\alpha_q} v) = v \ . \end{aligned}$$

Hence under condition (4.24), $\mathsf{0}^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} \prec^{\mathsf{l}}_{n-m} v$ simplifies to true and $\mathsf{0}^{\mathsf{p}}_{(v^{\mathsf{w}},\, q)} \preceq^{\mathsf{l}}_{n-m} v$ simplifies to false. Next we assume

$$\bigwedge_{0 < i \leq k} (\mathsf{s}_i^{\alpha_q} v)^{\mathsf{w}} = l_i \ \wedge \ \bigwedge_{0 < i \leq k} \mathsf{s}_i^{\alpha_q} v = \mathsf{0}^{\mathsf{w}}_{(l_i)} \ . \tag{4.25}$$

Under this condition, $0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} = v$, and hence both $0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} \prec^{\mathsf{l}}_{n-m} v$ and $0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} \preceq^{\mathsf{l}}_{n-m} v$ simplify to false. If we assume that for some $k' \in [1, k-1]$,

$$\bigwedge_{0<i\leq k} (\mathsf{s}_i^{\alpha_q} v)^{\mathsf{w}} = l_i \ \wedge\ \mathsf{s}_{k'}^{\alpha_q} v \neq 0^{\mathsf{w}}_{(l_{k'})} \ , \tag{4.26}$$

then we must have $0^{\mathsf{w}}_{(l_{k'})} \prec^{\mathrm{pl}} \mathsf{s}_{k'}^{\alpha_q} v$. So we still have an infinite chain

$$\begin{aligned} 0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} &\preceq^{\mathsf{l}} \alpha_q(0^{\mathsf{w}}_{(i_1)}, \ldots, 0^{\mathsf{w}}_{(i_{k'})}, \ldots, 0^{\mathsf{w}}_{(i_k)}) \\ &\prec^{\mathsf{l}} \alpha_q(0^{\mathsf{w}}_{(i_1)}, \ldots, 0^{\mathsf{w}}_{(i_{k'})}, \ldots, f(0^{\mathsf{w}}_{(i_k)})) \\ &\prec^{\mathsf{l}} \alpha_q(0^{\mathsf{w}}_{(i_1)}, \ldots, 0^{\mathsf{w}}_{(i_{k'})}, \ldots, f^2(0^{\mathsf{w}}_{(i_k)})) \\ &\prec^{\mathsf{l}} \ \ldots\ldots \\ &\prec^{\mathsf{l}} \alpha_q(\mathsf{s}_1^{\alpha_q} v, \ldots, \mathsf{s}_{k'}^{\alpha_q} v, \ldots, \mathsf{s}_k^{\alpha_q} v) = v \ . \end{aligned}$$

Therefore under condition (4.26), $0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} \prec^{\mathsf{l}}_{n-m} v$ simplifies to true and $0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} \preceq^{\mathsf{l}}_{n-m} v$ simplifies to false. The last case is that

$$\bigwedge_{0<i\leq k} (\mathsf{s}_i^{\alpha_q} v)^{\mathsf{w}} = l_i \ \wedge\ \bigwedge_{1\leq i\leq k-1} \mathsf{s}_i^{\alpha_q} v = 0^{\mathsf{w}}_{(l_i)} \ \wedge\ \mathsf{s}_k^{\alpha_q} v \neq 0^{\mathsf{w}}_{(l_k)} \ . \tag{4.27}$$

Then $0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} \prec^{\mathsf{l}}_{n-m} v$ reduces to $0^{\mathsf{w}}_{(l_k)} \prec^{\mathrm{pl}}_{n-m} \mathsf{s}_k^{\alpha_q} v$ and $0^{\mathsf{p}}_{(v^{\mathsf{w}},\,q)} \preceq^{\mathsf{l}}_{n-m} v$ reduces to $0^{\mathsf{w}}_{(l_k)} \preceq^{\mathrm{pl}}_{n-m} \mathsf{s}_k^{\alpha_q} v$. Now we have to reapply the above reduction to $0^{\mathsf{w}}_{(l_k)} \prec^{\mathrm{pl}}_{n-m} \mathsf{s}_k^{\alpha_q} v$ and to $0^{\mathsf{w}}_{(l_k)} \preceq^{\mathrm{pl}}_{n-m} \mathsf{s}_k^{\alpha_q} v$. But since $l_k < \mathsf{w}(v)$, this process eventually terminates.

Note that the lower boundary terms we used in the reduction are all defined on known constant weights, and hence they are fixed ground TA-terms which can be expressed using selector terms and constants.

4.4.3 Reduction of $u \prec^{\mathsf{l}}_n v$ and $u \preceq^{\mathsf{l}}_n v$

If either u or v is a constant, then both $u \prec^{\mathsf{l}}_n v$ and $u \preceq^{\mathsf{l}}_n v$ simplify to false. Now assume $\mathrm{type}(u) = \mathrm{type}(v) = \alpha_q$ and $\mathrm{ar}(\alpha_q) = k > 0$. So

$$u = \alpha_q(\mathsf{s}_1^{\alpha_q} u, \ldots, \mathsf{s}_k^{\alpha_q} u) \ , \qquad\qquad v = \alpha_q(\mathsf{s}_1^{\alpha_q} v, \ldots, \mathsf{s}_k^{\alpha_q} v) \ .$$

If we assume that

$$\bigwedge_{1\leq i\leq k'-1} \mathsf{s}_i^{\alpha_q} u = \mathsf{s}_i^{\alpha_q} v \ \wedge\ \mathsf{s}_{k'}^{\alpha_q} u \prec^{\mathrm{kb}} \mathsf{s}_{k'}^{\alpha_q} v \tag{4.28}$$

for some $k' \in [1, k-1]$, then we have an infinite chain

$$\begin{aligned} u &= \alpha_q(\mathsf{s}_1^{\alpha_q} u, \ldots, \mathsf{s}_{k'}^{\alpha_q} u \ldots, \mathsf{s}_k^{\alpha_q} u) \\ &\prec^{\mathsf{l}} \alpha_q(\mathsf{s}_1^{\alpha_q} u, \ldots, \mathsf{s}_{k'}^{\alpha_q} u \ldots, f(\mathsf{s}_k^{\alpha_q} u)) \\ &\prec^{\mathsf{l}} \alpha_q(\mathsf{s}_1^{\alpha_q} u, \ldots, \mathsf{s}_{k'}^{\alpha_q} u \ldots, f^2(\mathsf{s}_k^{\alpha_q} u)) \\ &\prec^{\mathsf{l}} \cdots\cdots \\ &\prec^{\mathsf{l}} \alpha_q(\mathsf{s}_1^{\alpha_q} v, \ldots, \mathsf{s}_{k'}^{\alpha_q} v \ldots, \mathsf{s}_k^{\alpha_q} v) = v \ . \end{aligned}$$

Therefore under condition (4.28), $u \prec_n^{\mathsf{l}} v$ simplifies to true and $u \preceq_n^{\mathsf{l}} v$ simplifies to false. The only possibility left is

$$\bigwedge_{1 \le i \le k-1} \mathsf{s}_i^{\alpha_q} u = \mathsf{s}_i^{\alpha_q} v \ \wedge \ \mathsf{s}_k^{\alpha_q} u \prec^{\mathsf{kb}} \mathsf{s}_k^{\alpha_q} v \ . \tag{4.29}$$

Under this condition, $u \prec_n^{\mathsf{l}} v$ simplifies to $\mathsf{s}_k^{\alpha_q} u \prec_n^{\mathsf{kb}} \mathsf{s}_k^{\alpha_q} v$, and $u \preceq_n^{\mathsf{l}} v$ simplifies to $\mathsf{s}_k^{\alpha_q} u \preceq_n^{\mathsf{kb}} \mathsf{s}_k^{\alpha_q} v$.

This concludes all reductions we need to consider. Although we carried out the reductions at semantic level by case distinction, all preconditions and the corresponding reducts are expressible in our formal language, and hence so are the reductions themselves. For example, The reductions of $u \prec_n^{\mathsf{l}} v$ and $u \preceq_n^{\mathsf{l}} v$, respectively, are

$$\begin{aligned} u \prec_n^{\mathsf{l}} v \quad &\to \quad \mathsf{s}_k^{\alpha_q} u \prec_n^{\mathsf{kb}} \mathsf{s}_k^{\alpha_q} v \ \vee \bigvee_{0 < i < k} \mathsf{s}_i^{\alpha_q} u \prec^{\mathsf{kb}} \mathsf{s}_i^{\alpha_q} v \ , \\ u \preceq_n^{\mathsf{l}} v \quad &\to \quad \mathsf{s}_k^{\alpha_q} u \prec_n^{\mathsf{kb}} \mathsf{s}_k^{\alpha_q} v \ \wedge \bigwedge_{0 < i < k} \mathsf{s}_i^{\alpha_q} u = \mathsf{s}_i^{\alpha_q} v \ . \end{aligned}$$

Note that it is to delineate $\prec_n^{\mathsf{w}}$, $\prec_n^{\mathsf{p}}$, $\preceq_n^{\mathsf{w}}$ and $\preceq_n^{\mathsf{p}}$, and to decompose $\prec_n^{\mathsf{l}}$ and $\preceq_n^{\mathsf{l}}$ that we introduced boundary functions. In the above reductions, however, not only are upper boundary functions undefined, but also the lower boundary functions have no use in these reductions (except in the intermediate steps). As a result, we do not need to introduce any boundary functions into $\mathscr{L}^{\mathbb{Z}}_{\mathsf{kb}^+}$ at all. Also note that it is to decompose $\prec_n^{\mathsf{l}}$ and $\preceq_n^{\mathsf{l}}$ that we introduced tuple gap orders and tuple boundary functions, which, however, are of no use in the above reductions. Hence we do not need to introduce tuple gap orders and tuple boundary functions into $\mathscr{L}^{\mathbb{Z}}_{\mathsf{kb}^+}$ either. All in all, the presence of a 0-weight unary function significantly simplifies the depth reduction procedure. In fact, we have already presented the proof for Lemmas 4.4. All other lemmas related to tuples and boundary functions in Section 4.2.8 and the corresponding proofs in Section 4.6 are no longer needed. Neither is Section 4.7 nor Section 4.8 therefore.

4.5 Conclusion

We showed the decidability of the first-order theory of term algebras with Knuth-Bendix order by quantifier elimination. Our method combines the extraction of integer constraints from term constraints with the reduction of quantifiers on term variables to quantifiers on integer variables. In fact, we established the decidability of a much more expressive theory.

Two problems related to practical complexity need further investigation. First, as a rule of thumb, more expressive power means higher complexity. Even if the theoretical complexity bound is the same, in practice the efficiency will be compromised. It is worthwhile to search for the smallest extension of KBO that admits quantifier elimination. Second, the elimination is intrinsically limited to processing quantified variables one at a time. We plan to extend the method in Chapter 2 to eliminate a block of quantifiers of the same kind in one step. We believe this will be a significant improvement in pragmatic terms, since in most applications the quantifier alternation depth is small.

We also plan to investigate the decidability issue of the first-order theory of KBO in a non-ground term domain [25], or with a partial precedence order on the signature [2].

4.6 Proofs

In this section we present the proofs not included in the main text of this chapter. These proofs rely on definitions about integer predicates and functions defined in Section 4.7 and reductions of equalities and gap order literals presented in Section 4.8. In this and the following sections we still assume that our language does not contain a 0-weight unary function. For ease of reference we restate the lemmas and theorems.

Lemma 4.1 (Elimination of Negative Literals). Assume formulas are type complete.

1. Any negative tester literal is equivalent to a disjunction of positive tester literals.
2. Any negative term equality literal (i. e., disequality between TA-terms) is equivalent to a disjunction of positive order literals.
3. Any negative order literal is equivalent to a disjunction of integer literals, tester literals and positive order literals.

Proof. Negative tester literals can be eliminated using the following two equivalence relations.

$$\neg\,\mathrm{Is}_{\alpha_p}(x) \;\leftrightarrow \bigvee_{\alpha_q\in C,\alpha_p\not\equiv\alpha_q} \mathrm{Is}_{\alpha_q}(x)\ , \qquad\qquad \neg\,\mathrm{Is}_{\mathsf{A}}(x) \;\leftrightarrow \bigvee_{\alpha_p\in C\setminus\mathcal{A}} \mathrm{Is}_{\alpha_p}(x)\ .$$

Term disequalities can be eliminated according to

$$u\not\equiv v \;\leftrightarrow\; u\prec^{\mathsf{w}} v \;\vee\; u\prec^{\mathsf{p}} v \;\vee\; u\prec^{\mathsf{l}} v \;\vee\; v\prec^{\mathsf{w}} u \;\vee\; v\prec^{\mathsf{p}} u \;\vee\; v\prec^{\mathsf{l}} u\ .$$

Now let $\sharp\in\{\mathsf{w},\mathsf{p},\mathsf{l}\}$ and $n>0$. It is easily seen that the following equivalence relations hold.

$$\begin{aligned}
u\prec^{\sharp}_{n} v &\;\leftrightarrow\; u\prec^{\sharp}_{n+1} v \;\vee\; u\preceq^{\sharp}_{n} v\ ,\\
\neg\,(u\prec^{\sharp}_{n} v) &\;\leftrightarrow\; \neg\,(u\prec^{\sharp}_{n+1} v) \;\wedge\; \neg\,(u\preceq^{\sharp}_{n} v)\ ,\\
u\preceq^{\sharp}_{n} v &\;\leftrightarrow\; (u\prec^{\sharp}_{n} v) \;\wedge\; \neg\,(u\prec^{\sharp}_{n+1} v)\ ,\\
\neg\,(u\preceq^{\sharp}_{n} v) &\;\leftrightarrow\; \neg\,(u\prec^{\sharp}_{n} v) \;\vee\; u\prec^{\sharp}_{n+1} v\ .
\end{aligned}$$

Therefore, for $n>0$, we have the following equivalence relations.

$$\begin{aligned}
\neg\,(u\prec^{\mathsf{w}}_{n} v) &\;\leftrightarrow\; v^{\mathsf{w}}\le u^{\mathsf{w}} \;\vee\; \bigvee_{i<n} u\preceq^{\mathsf{w}}_{i} v\ ,\\
\neg\,(u\preceq^{\mathsf{w}}_{n} v) &\;\leftrightarrow\; v^{\mathsf{w}}\le u^{\mathsf{w}} \;\vee\; \bigvee_{i<n} u\preceq^{\mathsf{w}}_{i} v \;\vee\; u\prec^{\mathsf{w}}_{n+1} v\ ,
\end{aligned}$$

$$\neg\,(u \prec_n^{\mathsf{p}} v) \;\leftrightarrow\; u^{\mathsf{w}} \neq v^{\mathsf{w}} \;\vee\; \Big(\bigvee_{0<p\le|\Sigma|} \mathrm{Is}_{\alpha_p}(u) \;\leftrightarrow\; \mathrm{Is}_{\alpha_p}(v)\Big) \;\vee\; v \prec^{\mathsf{p}} u \;\vee\; \bigvee_{i<n} u \preceq_i^{\mathsf{p}} v \;,$$

$$\neg\,(u \preceq_n^{\mathsf{p}} v) \;\leftrightarrow\; u^{\mathsf{w}} \neq v^{\mathsf{w}} \;\vee\; \Big(\bigvee_{0<p\le|\Sigma|} \mathrm{Is}_{\alpha_p}(u) \;\leftrightarrow\; \mathrm{Is}_{\alpha_p}(v)\Big) \;\vee\; v \prec^{\mathsf{p}} u \;\vee\; \bigvee_{i<n} u \preceq_i^{\mathsf{p}} v \;\vee\; u \prec_{n+1}^{\mathsf{p}} v \;,$$

$$\neg\,(u \prec_n^{\mathsf{l}} v) \;\leftrightarrow\; u^{\mathsf{w}} \neq v^{\mathsf{w}} \;\vee\; \Big(\bigwedge_{0<p\le|\Sigma|} \mathrm{Is}_{\alpha_p}(u) \;\leftrightarrow\; \neg\,\mathrm{Is}_{\alpha_p}(v)\Big) \;\vee\; v \prec^{\mathsf{l}} u \;\vee\; \bigvee_{i<n} u \preceq_i^{\mathsf{l}} v \;,$$

$$\neg\,(u \preceq_n^{\mathsf{l}} v) \;\leftrightarrow\; u^{\mathsf{w}} \neq v^{\mathsf{w}} \;\vee\; \Big(\bigwedge_{0<p\le|\Sigma|} \mathrm{Is}_{\alpha_p}(u) \;\leftrightarrow\; \neg\,\mathrm{Is}_{\alpha_p}(v)\Big) \;\vee\; v \prec^{\mathsf{l}} u \;\vee\; \bigvee_{i<n} u \preceq_i^{\mathsf{l}} v \;\vee\; u \prec_{n+1}^{\mathsf{l}} v \;.$$

Note that $\neg\,(u \prec_0^{\sharp} v)$ is $u \neq v \wedge \neg\,(u \prec^{\sharp} v)$ and $\neg\,(u \preceq_0^{\sharp} v)$ is just $u \neq v$. Under the assumption of type completeness, the truth values of tester literals are predetermined, and hence no conjunction is introduced in the above reductions. And obviously, none of the above reductions introduces more open gap order literals in each of the reducts. □

Lemma 4.2 (Delineated Gap Order Completion). Any conjunction of positive order literals in $\mathscr{L}_{\mathrm{kb}^+}$ can be effectively transformed to an equivalent finite disjunction of delineated gap order completions.

Proof. Let φ_{kb^+} be a conjunction of positive order literals in $\mathscr{L}_{\mathrm{kb}^+}$. Without loss of generality we assume that φ_{kb^+} only contains literals like $u \preceq_n^{\sharp} v$ and $u \prec_n^{\sharp} v$ ($\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$) for $n > 0$. Let $\mathcal{T}(\varphi_{\mathrm{kb}^+})$ be the set of all TA-terms in φ_{kb^+}. In the following we describe an nondeterministic algorithm to compute a disjunction of DGOCs equivalent to φ_{kb^+}.

We view φ_{kb^+} as a labeled directed graph G_{kb^+} in which TA-terms in $\mathcal{T}(\varphi_{\mathrm{kb}^+})$ are vertex labels and $u \preceq_n^{\sharp} v$ (or $u \prec_n^{\sharp} v$) represents a directed *edge* from u to v with number label n and color label $\sharp$. An edge corresponding to a stretchable gap order literal $u \prec_n^{\sharp} v$ (resp. a rigid gap order literal $u \preceq_n^{\sharp} v$) is called *stretchable* (resp. *rigid*). A path is *rigid* if every edge on the path is rigid; it is *stretchable* otherwise. A *length of a path* is the sum of all number labels on the path.

First we guess an equality partition for $\mathcal{T}(\varphi_{\mathrm{kb}^+})$. All vertexes in the same equivalence class are merged by consolidating corresponding outgoing and incoming edges. We choose an arbitrary representative for each equivalence class as the vertex label.

Second we guess a linear order with respect to $\preceq_1^{\sharp}$ and $\prec_1^{\sharp}$ for $\mathcal{T}(\varphi_{\mathrm{kb}^+})$. It amounts to adding to G_{kb^+} a Hamilton path, a path between two distinct vertexes of G_{kb^+} that visits each vertex exactly once. We call this path *primary path* and edges on the path *primary edges*. This primary path corresponds to a gap order completion and it is a prototype of the delineated gap order completion we are about to construct. At this phase, a primary edge may be labeled by any color in $\{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$, but all number labels are 1.

Third we insert boundary terms to delineate the primary path if necessary.

Up to now there may exist two types of inconsistency.

1. G_{kb^+} contains a loop.

2. The color of a non-primary edge is incompatible with colors on the corresponding primary path. Let order $\{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$ as $\mathsf{l} \prec^{\mathsf{c}} \mathsf{p} \prec^{\mathsf{c}} \mathsf{w}$. We say that the color c of a non-primary edge is incompatible with colors $\{c_i \mid 0 < i \leq k\}$ on the corresponding primary path if $c \neq \max\{c_i \mid 0 < i \leq k\}$ with respect to $\prec^{\mathsf{c}}$. For example, $u \prec^{\mathsf{l}}_{n_1} v \prec^{\mathsf{p}}_{n_2} w$ is incompatible with neither $u \prec^{\mathsf{w}}_{n_3} w$ nor $u \prec^{\mathsf{l}}_{n_4} w$ while it is compatible with $u \prec^{\mathsf{p}}_{n_5} w$.

If an inconsistency is detected, the constraint simplifies to false.

To reduce φ_{kb^+} to a DGOC, we need to remove all non-primary edges. A non-primary edge from u to v is called an *over-edge* (resp. *under-edge*, *para-edge*) if its length is greater than (resp. less than, equal to) the length of the primary path from u to v. We say that a non-primary edge from u to v *covers* all primary edges on the path from u to v.

We remove all *redundant edges* which either are para-edges or stretchable under-edges. If there exists a rigid under-edge, then the constraint simplifies to false. Now we assume that all under-edges are removed.

The existence of an over-edge means the current gap order completion "over-approximates" φ_{kb^+}. To discharge an over-edge, we distribute the extra length over the number labels on primary stretchable edges that is covered by this over-edge. For example consider the following primary path from u to v to w.

$$u \xrightarrow{p_1} v \xrightarrow{p_2} w \ , \tag{4.30}$$

where both two edges are stretchable. Suppose that there exists an over-edge from u to w with the number label $p > p_1 + p_2$. We replace the subgraph (4.30) by one of the following.

$$u \xrightarrow{p_1+n_1} v \xrightarrow{p_2+n_2} w \ , \tag{4.31}$$

where $n_1, n_2 \geq 0$ and $n_1 + n_2 = p - p_1 - p_2$. Note that we do not add extra length to rigid primary edges, and if the corresponding primary path is not stretchable, then φ_{kb^+} simplifies to false. After the distribution, we make each stretchable primary edge rigid if the corresponding over-edge that covers it is rigid. Now the original over-edge becomes redundant and we remove it. One by one, we can remove all over-edges and obtain a graph which only has primary edges (false can be viewed as $x \prec^{\mathsf{w}} x$). □

Lemma 4.12 (Open Gap Order Literals in DGOCs). *Formulas of the form*

$$\bigwedge_{i=1}^{m} u_i \trianglelefteq^{\sharp_i}_{p_i} x \ \wedge \ \bigwedge_{j=1}^{n} x \trianglelefteq^{\natural_j}_{q_j} v_j \ , \tag{4.32}$$

where x is an ordinary TA*-term, $\trianglelefteq$ stands for $\prec$ or $\leqq$, $\sharp_i, \natural_j \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$, and $p_i, q_j > 0$, can be effectively transformed to an equivalent finite disjunction of DGOCs and the number of open gap order literals in each*

DGOC is no more than that in the original formula. Moreover, if both the predecessor and the successor of x in a DGOC are boundary terms, then the DGOC has one fewer open gap order literals than the original formula provided the original formula contains at least one open gap order literal.

Proof. By Lemma 4.2 it suffices to show that in each DGOC the number of open gap order literals is no more than that in the original formula. Let us assume there are $m_1 \leq m$ ordinary TA-terms which are less than x and $n_1 \leq n$ ordinary TA-terms which are greater than x. So in the original formula there are $m_1 + n_1 + 1$ ordinary terms and $m_1 + n_1$ open gap order literals. In each obtained DGOC, the number of ordinary TA-terms can not exceed $m_1 + n_1 + 1$ as no new ordinary TA-terms are introduced. It is easily seen that in a linear order containing $k > 0$ ordinary TA-terms, the number of open gap order literals is at most $k - 1$. So there are at most $m_1 + n_1$ open gap order literals at each resulting DGOC. The introduction of boundary terms (to delineate a gap order) and the merge of two vertexes (to equalize two terms) can only decrease the number of open gap order literals. Moreover, suppose the original formula contains at least one open gap order literal. Without loss of generality, let us assume it is $u_1 \trianglelefteq_1 x$, and so $m_1 > 0$. Since both the predecessor and the successor of x in the DGOC are boundary terms, x is separated from other ordinary TA-terms, and so there are at most $m_1 - 1$ open gap order literals at the left-hand side of x in the DGOC. Also there are at most n_1 open gap order literals at the right-hand side of x. Therefore, the total number of open gap order literals is $m_1 + n_1 - 1$, one less than that in the original formula. □

Lemma 4.3 (No Embedding of Boundary Terms). Any formula in $\mathscr{L}^{Z}_{\mathrm{kb}^+}$ can be effectively reduced to an equivalent formula in which no boundary terms appear inside selectors or the weight function.

Proof. By the following equivalence relations we can eliminate all integer terms formed by applying the weight function to boundary terms.

$$
\begin{aligned}
(0^{\mathsf{w}}_{(m)})^{\mathsf{w}} &\leftrightarrow (\mathrm{WD}_{0^{\mathsf{w}}}(m) \wedge m) \vee (\neg\, \mathrm{WD}_{0^{\mathsf{w}}}(m) \wedge \mathsf{w}(\bot)) \ , \\
(0^{\mathsf{p}}_{(m,p)})^{\mathsf{w}} &\leftrightarrow (\mathrm{WD}_{0^{\mathsf{w}}}(m,p) \wedge m) \vee (\neg\, \mathrm{WD}_{0^{\mathsf{w}}}(m,p) \wedge \mathsf{w}(\bot)) \ , \\
(1^{\mathsf{w}}_{(m)})^{\mathsf{w}} &\leftrightarrow (\mathrm{WD}_{1^{\mathsf{w}}}(m) \wedge m) \vee (\neg\, \mathrm{WD}_{1^{\mathsf{w}}}(m) \wedge \mathsf{w}(\bot)) \ , \\
(1^{\mathsf{p}}_{(m,p)})^{\mathsf{w}} &\leftrightarrow (\mathrm{WD}_{1^{\mathsf{w}}}(m,p) \wedge m) \vee (\neg\, \mathrm{WD}_{1^{\mathsf{w}}}(m,p) \wedge \mathsf{w}(\bot)) \ .
\end{aligned}
$$

Next we show how to eliminate selectors in front of boundary terms. For a selector term $\mathsf{s}^{\alpha_p}_i 0^{\mathsf{w}}_{(m)}$ (where $\mathrm{ar}(\alpha_p) = k$), we consider four cases.

1. $0^{\mathsf{w}}_{(m)}$ is not well-defined. In this case $0^{\mathsf{w}}_{(m)} = \bot$, and hence $\mathsf{s}^{\alpha_p}_i 0^{\mathsf{w}}_{(m)}$ simplifies to $\bot$.
2. $0^{\mathsf{w}}_{(m)}$ is well-defined, but $\mathrm{Tree}^{(\alpha_p)}(m)$ is false. Since $\mathrm{Tree}^{(\alpha_p)}(m)$ does not hold, $0^{\mathsf{w}}_{(m)}$ can not be of type α_p, and so $\mathsf{s}^{\alpha_p}_i 0^{\mathsf{w}}_{(m)}$ simplifies to $0^{\mathsf{w}}_{(m)}$.
3. $\mathrm{Tree}^{(\alpha_p)}(m)$ holds but there exists p' such that $p < p' \leq |\Sigma|$ and $\mathrm{Tree}^{(\alpha_{p'})}(m)$ holds. Since $\alpha_{p'} \prec^{\Sigma} \alpha_p$, $0^{\mathsf{w}}_{(m)}$ can not be of type α_p, and hence $\mathsf{s}^{\alpha_p}_i 0^{\mathsf{w}}_{(m)}$ simplifies to $0^{\mathsf{w}}_{(m)}$.

4. $\mathrm{Tree}^{(\alpha_p)}(m)$ is true and for any p' such that $p < p' \leq |\Sigma|$, $\mathrm{Tree}^{(\alpha_{p'})}(m)$ does not hold. The condition says that $0^{\mathsf{w}}_{(m)}$ is of type α_p, i. e., $0^{\mathsf{w}}_{(m)} = \alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha_p)})$. It is not hard to argue that $\langle t_1, \ldots, t_{\mathrm{ar}(\alpha_p)} \rangle$ should be the smallest k-tuple of weight n. Therefore, $\mathsf{s}^{\alpha_p}_i 0^{\mathsf{w}}_{(m)}$ simplifies to $0^{\mathsf{w}}_{(\mathrm{CWS}^k_i(m - \mathsf{w}(\alpha_p)))}$, where $\mathrm{CWS}^k_i(n)$ returns the weight of the i^{th} component of a k-tuple which is the smallest (with respect to $\prec^{k;\mathrm{kb}}$) k-tuple of weight n. Here $\mathrm{CWS}^k_i(n)$ is defined in Presburger arithmetic (Section 4.7).

The above four preconditions can be respectively expressed as

$$\begin{aligned}
&\neg\, \mathrm{WD}_{0^{\mathsf{w}}}(m) \ , \\
&\mathrm{WD}_{0^{\mathsf{w}}}(m) \ \wedge \ \neg\, \mathrm{Tree}^{(\alpha_p)}(m) \ , \\
&\mathrm{Tree}^{(\alpha_p)}(m) \ \wedge \bigvee_{p < p' \leq |\Sigma|} \mathrm{Tree}^{(\alpha_{p'})}(m) \ , \\
&\mathrm{Tree}^{(\alpha_p)}(m) \ \wedge \bigwedge_{p < p' \leq |\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_{p'})}(m) \ .
\end{aligned}$$

Similarly we can handle terms of the form $\mathsf{s}^{\alpha_p}_j 1^{\mathsf{w}}_{(m)}$.

For a selector term $\mathsf{s}^{\alpha_p}_j 0^{\mathsf{p}}_{(m,p')}$ (where $\mathrm{ar}(\alpha_p) = k$), we consider three cases.

1. $0^{\mathsf{p}}_{(m,p')}$ is not well-defined. Then $\mathsf{s}^{\alpha_p}_j 0^{\mathsf{p}}_{(m,p')}$ simplifies to $\bot$.
2. $0^{\mathsf{p}}_{(m,p')}$ is well-defined, but $p \neq p'$. Then $\mathsf{s}^{\alpha_p}_j 0^{\mathsf{p}}_{(m,p')}$ simplifies to $0^{\mathsf{p}}_{(m,p')}$.
3. $0^{\mathsf{p}}_{(m,p')}$ is well-defined, and $p = p'$. The condition says that $0^{\mathsf{p}}_{(m,p)} = \alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha_p)})$ and $\langle t_1, \ldots, t_{\mathrm{ar}(\alpha_p)} \rangle$ is the smallest k-tuple of weight n. As case (4) in the reduction of $\mathsf{s}^{\alpha_p}_i 0^{\mathsf{w}}_{(m)}$, $\mathsf{s}^{\alpha_p}_j 0^{\mathsf{p}}_{(m,p')}$ simplifies to $0^{\mathsf{w}}_{(\mathrm{CWS}^k_j(m - \mathsf{w}(\alpha_p)))}$.

The above three preconditions can be respectively expressed as

$$\begin{aligned}
&\neg\, \mathrm{WD}_{0^{\mathsf{p}}}(m, p') \ , \\
&\mathrm{WD}_{0^{\mathsf{p}}}(m, p') \ \wedge \ p \neq p' \ , \\
&\mathrm{WD}_{0^{\mathsf{p}}}(m, p') \ \wedge \ p = p' \ .
\end{aligned}$$

Similarly we can handle terms of the form $\mathsf{s}^{\alpha_p}_j 1^{\mathsf{p}}_{(m,p')}$.

Repeating this transformation, eventually all selectors in front of boundary terms are removed and we obtain a formula in which no boundary terms occur inside selectors.

We note that the above reduction is purely syntactic as for each case, both the precondition and the corresponding reduct are expressible in our formal language. In general, let $\varphi[t]$ be a formula in which term t occurs. Suppose there are n disjoint cases in total for the reduction of t and for each case i $(0 < i \leq n)$, t is reduced to t_i under the precondition θ_i. Then $\varphi[t]$ can be equivalently

rewritten as

$$\bigvee_{0<i\leq n} \left(\theta_i \wedge \varphi[t_i]\right) \ . \tag{4.33}$$

It is easily seen from (4.33) that the transformation involves disjunctive splittings and since θ_i are Presburger formulas, the transformation does not increase the number of open gap order literals in each of the resulting conjunctions of literals. □

Lemma 4.4 (Reduction of Term Gap Order Literals). Let

$$\star \in \{ \prec_n^{\mathsf{kb}}, \prec_n^{\mathsf{w}}, \prec_n^{\mathsf{p}}, \prec_n^{\mathsf{l}}, \prec_n^{\mathsf{pl}}, \preceq_n^{\mathsf{kb}}, \preceq_n^{\mathsf{w}}, \preceq_n^{\mathsf{p}}, \preceq_n^{\mathsf{l}}, \preceq_n^{\mathsf{pl}} \} \qquad (n > 0) \ .$$

If x is a TA-variable of type α_p with $\alpha_p = (\mathsf{s}_1^{\alpha_p}, \ldots, \mathsf{s}_k^{\alpha_p})$ and t is an arbitrary TA-term, then $x \star t$ ($t \star x$) can be effectively reduced to an equivalent quantifier-free formula $\varphi(\mathsf{s}_1^{\alpha_p} x, \ldots, \mathsf{s}_k^{\alpha_p} x)$ (in $\mathscr{L}^{Z}_{\mathsf{kb}^+}$) where x only occurs in $\mathsf{s}_i^{\alpha_p} x$ $(0 < i \leq k)$.

Proof. It suffices to only consider $\star \in \{\prec_n^{\mathsf{w}}, \prec_n^{\mathsf{p}}, \prec_n^{\mathsf{l}}, \preceq_n^{\mathsf{w}}, \preceq_n^{\mathsf{p}}, \preceq_n^{\mathsf{l}}, \}$ for $n > 0$. If t is a TA-term not containing x, then the result follows directly from Lemma 4.6. Suppose otherwise. Then t must be in the form Lx where L is a non-empty block of selectors. We have two cases depending on whether Lx is at the left-hand side or the right-hand side of $\star$.

Case 1. $x \star Lx$. Since L is a non-empty and constraints are type-complete, Lx is a proper subterm of x. If $x \star Lx$, then $x \prec^{\mathsf{kb}} Lx$, violating the subterm property of KBO [2]. Therefore, $x \star Lx$ simplifies to false.

Case 2. $Lx \star x$. If $\star$ is $\prec_n^{\mathsf{w}}$ (resp. $\prec_n^{\mathsf{p}}$), reduction 4.197 (resp. 4.198) puts x and Lx in separate literals, and hence the problem goes away. Similarly if $\star$ is $\preceq_n^{\mathsf{w}}$ or $\preceq_n^{\mathsf{p}}$. The last two cases are $Lx \prec_n^{\mathsf{l}} x$ and $Lx \preceq_n^{\mathsf{l}} x$, which are only possible when the language contains the unary function f of weight 0. To save notation, we use s_f to denote the corresponding selector of f. Now $Lx \prec_n^{\mathsf{l}} x$ and $Lx \preceq_n^{\mathsf{l}} x$, respectively, are in the forms

$$\mathsf{s}_f^i(x) \prec_n^{\mathsf{l}} x \qquad \text{and} \qquad \mathsf{s}_f^i(x) \preceq_n^{\mathsf{l}} x \ ,$$

where $i > 0$ and s_f^i denote i applications of s_f. Note that both $\mathsf{s}_f^i(x) \prec_n^{\mathsf{l}} x$ and $\mathsf{s}_f^i(x) \preceq_n^{\mathsf{l}} x$ imply $\mathrm{Is}_f(\mathsf{s}_f^i(x))$. In the rest of the proof, we view x as a fixed ground TA-term unless it appears in a formula in the formal language. Let us consider two subcases.

A. There is a non-constant ground TA-term whose weight is $\mathsf{w}(x)$ and type is not f.
Let $g(t_1, \ldots, t_j)$ be such a term; i. e., $g \not\equiv f$ and $\mathsf{w}(g(t_1, \ldots, t_j)) = \mathsf{w}(x)$.

(a) If $\mathsf{s}^i_f(x)$ is $f^k(a)$ (where $k > 0$ and a is a constant), then x is $f^{i+k}(a)$. We have either $a \prec^{\mathsf{p}} g(t_1, \ldots, t_j)$ or $g(t_1, \ldots, t_j) \prec^{\mathsf{p}} a$, as $\mathsf{w}(g(t_1, \ldots, t_j)) = \mathsf{w}(x) = \mathsf{w}(a)$.

i. $a \prec^{\mathsf{p}} g(t_1, \ldots, t_j)$. Then we have

$$f^k(a) \prec^{\mathsf{l}} f^k(g(t_1, \ldots, t_j)) \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} f^k(g(f^l(t_1), \ldots, t_j)) \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} f^{i+k}(a) \ .$$

ii. $g(t_1, \ldots, t_j) \prec^{\mathsf{p}} a$. Then we have

$$f^k(a) \prec^{\mathsf{l}} f^{i+k}(g(t_1, \ldots, t_j)) \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} f^{i+k}(g(f^l(t_1), \ldots, t_j)) \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} f^{i+k}(a) \ .$$

(b) If $\mathsf{s}^i_f(x)$ is $f^k(h(t_1, \ldots, t_j))$ (where $k > 0$ and $h \not\equiv f$), then x is $f^{k+i}(h(t_1, \ldots, t_j))$. Then we have

$$f^k(h(t_1, \ldots, t_j)) \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} f^k(h(f^l(t_1), \ldots, t_j)) \prec^{\mathsf{l}} \cdots \prec^{\mathsf{l}} f^{i+k}(a) \ .$$

In all cases, there are infinitely many terms in between $\mathsf{s}^i_f(x)$ and x. Hence $\mathsf{s}^i_f(x) \prec^{\mathsf{l}}_n x$ simplifies to $\mathrm{Is}_f(\mathsf{s}^i_f(x))$ and $\mathsf{s}^i_f(x) \preceq^{\mathsf{l}}_n x$ simplifies to false.

B. Any term of weight $\mathsf{w}(x)$ is either a f-term (f-type term) or a constant. The condition says that $\mathsf{s}^i_f(x)$ must be in the form $f^k(c)$ (for some constant c and $k > 0$) and then x is in the form $f^{k+i}(c)$. In this case, $f^k(c) \preceq^{\mathsf{l}}_r f^{k+i}(c)$ where r is i times the number of constants of weight $\mathsf{w}(x)$. More precisely, let us assume that there are m constants of weight $\mathsf{w}(x)$ ordered as follows:

$$c_1 \prec^{\mathrm{kb}} \cdots \prec^{\mathrm{kb}} c_j \prec^{\mathrm{kb}} \cdots \prec^{\mathrm{kb}} c_m \ ,$$

where $c \equiv c_j$. Then we have *i tight* chains each of which has length m as follows.

$$\begin{array}{l} f^k(c_j) \prec^{\mathrm{kb}} \cdots \prec^{\mathrm{kb}} f^k(c_m) \prec^{\mathrm{kb}} f^{k+1}(c_1) \prec^{\mathrm{kb}} \cdots \prec^{\mathrm{kb}} f^{k+1}(c_j) \ , \\ \cdots\cdots \\ f^{k+i-1}(c_j) \prec^{\mathrm{kb}} \cdots \prec^{\mathrm{kb}} f^k(c_m) \prec^{\mathrm{kb}} f^{k+i}(c_1) \prec^{\mathrm{kb}} \cdots \prec^{\mathrm{kb}} f^{k+i}(c_j) \ . \end{array}$$

Therefore, $\mathsf{s}^i_f(x) \prec^{\mathsf{l}}_n x$ simplifies to

$$\mathrm{Is}_f(\mathsf{s}^i_f(x)) \ \wedge \ \mathrm{CNT}_m(x^{\mathsf{w}}) \ ,$$

where $m = \lceil n/i \rceil - 1$ and $\mathrm{CNT}_m(x^{\mathsf{w}})$ is defined in the language with f removed, and $\mathsf{s}^i_f(x) \preceq^{\mathsf{l}}_n x$ simplifies to

$$\mathrm{Is}_f(\mathsf{s}^i_f(x)) \ \wedge \ \mathrm{CNT}_m(x^{\mathsf{w}}) \ \wedge \ \neg\, \mathrm{CNT}_{m+1}(x^{\mathsf{w}}) \ \wedge \ i \mid n \ ,$$

where $m = \lceil n/i \rceil - 1$ and $\mathrm{CNT}_m(x^{\mathsf{w}})$ is defined the same.

Finally we note that each precondition can be expressed in Presburger arithmetic. Condition (A) is

$$\bigvee_{\alpha \in \mathcal{C} \setminus \mathcal{A} \setminus \{f\}} \mathrm{Tree}^{\alpha}(x^{\mathsf{w}}) \ , \tag{4.34}$$

and condition (B) is just the negation of (4.34). □

Lemma 4.5 (Reduction of Closed Term Literals). Let

$$\star \in \{ =, <_n^{\mathsf{kb}}, <_n^{\mathsf{w}}, <_n^{\mathsf{p}}, <_n^{\mathsf{l}}, <_n^{\mathsf{pl}}, \leqq_n^{\mathsf{kb}}, \leqq_n^{\mathsf{w}}, \leqq_n^{\mathsf{p}}, \leqq_n^{\mathsf{l}}, \leqq_n^{\mathsf{pl}} \} \qquad (n > 0) \ .$$

If $u \star v$ is closed, i. e., both u and v are boundary terms, then it can be effectively reduced to an equivalent Presburger formula.

Proof. There are many combinations of closed term literals. Here we only highlight the key components in the proof and leave the details to Sections 4.8.1, 4.8.2 and 4.8.3.

Counting Constraints. A closed term literal states that a certain number of ground TA-terms exist in a continuous segment formed by concatenating adjacent w-intervals or p-intervals. The idea of the reduction is to count how many ground TA-terms are in each interval. For example, $1^{\mathsf{w}}_{(m)} = 0^{\mathsf{w}}_{(m')}$ means that $m = m'$ and there is exactly one ground TA-term of weight m, which can be formally expressed as

$$m = m' \ \wedge \ \mathrm{Tree}(m) \ \wedge \ \neg \, \mathrm{CNT}_1(m) \ .$$

As another example, $1^{\mathsf{w}}_{(m)} <_n^{\mathsf{w}} 0^{\mathsf{w}}_{(m')}$ $(n > 0)$ reduces to (reduction (4.93), copied below)

$$\left(\begin{array}{l} \qquad\qquad (n = 1 \ \rightarrow \ m < m') \\ \qquad\qquad\qquad \wedge \\ n > 1 \ \rightarrow \bigvee_{0<r<n} (\exists z_1 \ldots \exists z_r) \left(\begin{array}{c} m < z_1 < \cdots < z_r < m' \\ \wedge \\ \bigvee_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_1, \ldots, n_r > 0}} \bigwedge_{0<i\leq r} \mathrm{CNT}_{n_i - 1}(z_i) \end{array} \right) \end{array} \right) \ . \tag{4.93}$$

We briefly explain the meaning of (4.93). Since $1^{\mathsf{w}}_{(m)}$ is the largest TA-term of weight m and $0^{\mathsf{w}}_{(m')}$ is the smallest TA-term of weight of m', $1^{\mathsf{w}}_{(m)} <_n^{\mathsf{w}} 0^{\mathsf{w}}_{(m')}$ means that there are at least $n - 1$ TA-terms whose weights are in (m, m'). If $n = 1$, then $1^{\mathsf{w}}_{(m)} <_n^{\mathsf{w}} 0^{\mathsf{w}}_{(m')}$ obviously reduces to $m < m'$. In case $n > 1$,

we assume the $n-1$ TA-terms in between $1^{\mathsf{w}}_{(m)}$ and $0^{\mathsf{w}}_{(m')}$ are distributed in r ($0 < r < n$) different levels measured by weights, which is formally represented by

$$\bigvee_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_1,\ldots,n_r>0}} \bigwedge_{0<i\leq r} \mathrm{CNT}_{n_i-1}(z_i) \ .$$

Since r ranges from 1 to $n-1$ which is a constant, (4.93) is an abbreviation for a first-order formula.

The reduction of $1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')}$ is similarly obtained as (reduction (4.141), copied below)

$$\left(\begin{array}{c} n = 1 \ \rightarrow \ \Big(m < m' \ \wedge \ \forall z \Big(m < z < m' \ \rightarrow \ \neg \, \mathrm{Tree}(z) \Big) \Big) \\ \wedge \\ n > 1 \ \rightarrow \ \bigvee\limits_{0<r<n} (\exists z_1 \ldots \exists z_r) \left(\begin{array}{c} m < z_1 < \cdots < z_r < m' \\ \wedge \\ \neg \, (\exists z_1 \ldots \exists z_{r+1}) \Big(m < z_1 < \cdots < z_{r+1} < m' \Big) \\ \wedge \\ \bigvee\limits_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_1,\ldots,n_r>0}} \bigwedge\limits_{0<i\leq r} \widehat{\mathrm{CNT}}_{n_i-1}(z_i) \end{array} \right) \end{array} \right) \ . \tag{4.141}$$

Here we need to add two more conjuncts. When $n = 1$ we need

$$\forall z \Big(m < z < m' \ \rightarrow \ \neg \, \mathrm{Tree}(z) \Big)$$

to make sure that there exists no term whose weight is in (m, m'). When $n > 1$, we need

$$\neg \, (\exists z_1 \ldots \exists z_r) \Big(m < z_1 < \cdots < z_{r+1} < m' \Big)$$

to guarantee that for a fixed $r \in (0, n)$ there are exactly r legitimate weights in (m, m').

Well-definedness of Boundary Terms. The reductions listed in Sections 4.8.1-4.8.5 implicitly assume well-definedness of boundary terms. The final reduction results should incorporate well-definedness predicates (in short, WD predicates, see Section 4.7). For example, for $n > 1$, $1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')}$ is equivalent to

$$\begin{array}{c} \Big(\mathrm{WD}_{1^{\mathsf{w}}}(m) \ \wedge \ \mathrm{WD}_{0^{\mathsf{w}}}(m') \ \wedge \ (4.93) \Big) \ \vee \ \Big(\neg \, \mathrm{WD}_{1^{\mathsf{w}}}(m) \ \wedge \ \mathrm{WD}_{0^{\mathsf{w}}}(m') \ \wedge \ \bot <^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \Big) \\ \vee \ \Big(\mathrm{WD}_{1^{\mathsf{w}}}(m) \ \wedge \ \neg \, \mathrm{WD}_{0^{\mathsf{w}}}(m') \ \wedge \ 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} \bot \Big) \ \vee \ \Big(\neg \, \mathrm{WD}_{1^{\mathsf{w}}}(m) \ \wedge \ \neg \, \mathrm{WD}_{0^{\mathsf{w}}}(m') \ \wedge \ \bot <^{\mathsf{w}}_{n} \bot \Big) \ . \end{array}$$

However, we do not need to go through this detour if the parameters in boundary terms come from weights of TA-terms, because WD predicates are valid. For example, for $n > 1$, the final reduction result of $1^{\mathsf{w}}_{(x^{\mathsf{w}})} \prec^{\mathsf{w}}_n 0^{\mathsf{w}}_{(y^{\mathsf{w}})}$ is just

$$\left(\begin{array}{l} \qquad\qquad (n = 1 \;\rightarrow\; x^{\mathsf{w}} < y^{\mathsf{w}}) \\ \qquad\qquad\qquad \wedge \\ n > 1 \;\rightarrow\; \bigvee\limits_{0<r<n} (\exists z_1 \ldots \exists z_r) \left(\begin{array}{c} x^{\mathsf{w}} < z_1 < \cdots < z_r < y^{\mathsf{w}} \\ \wedge \\ \bigvee\limits_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_1,\ldots,n_r>0}} \bigwedge\limits_{0<i\leq r} \mathrm{CNT}_{n_i-1}(z_i) \end{array}\right) \end{array}\right) .$$

Note that we use $\perp$ as the "sink value" to make boundary functions total. In general, the choice of a sink value affects the truth values of formulas in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$; i. e., $\mathrm{Th}(\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+})$ varies with the sink value. However, $\mathrm{Th}(\mathrm{TA}_{\mathrm{kb}})$ remains the same thanks to the incorporation of WD predicates.

Reduction of Relations Involving Constants. There is no separate list of reductions involving constants, as constants are just boundary terms. For example, $0^{\mathsf{p}}_{(\mathsf{w}(\alpha_p),p)}$ is another name for a constant α_p. So is $1^{\mathsf{p}}_{(\mathsf{w}(\alpha_p),p)}$.

Summary

1. Section 4.8.1 lists reductions of closed term equalities.
2. Section 4.8.2 lists reductions of closed stretchable term gap order literals.
3. Section 4.8.3 lists reductions of closed rigid term gap order literals. □

Lemma 4.6 (Reduction of Non-closed Term Gap Order Literals). Let

$$\star \in \{\, \prec^{\mathrm{kb}}_n, \prec^{\mathsf{w}}_n, \prec^{\mathsf{p}}_n, \prec^{\mathsf{l}}_n, \prec^{\mathrm{pl}}_n, \preceq^{\mathrm{kb}}_n, \preceq^{\mathsf{w}}_n, \preceq^{\mathsf{p}}_n, \preceq^{\mathsf{l}}_n, \preceq^{\mathrm{pl}}_n \,\} \qquad (n > 0) \ .$$

If x is a TA-variable of type α_p with $\alpha_p = (\mathsf{s}^{\alpha_p}_1, \ldots, \mathsf{s}^{\alpha_p}_k)$ and t is either a boundary term or an ordinary TA-term *not* containing x, then $x \star t$ ($t \star x$) can be effectively reduced to an equivalent quantifier-free formula $\varphi(\mathsf{s}^{\alpha_p}_1 x, \ldots, \mathsf{s}^{\alpha_p}_k x)$ where x only occurs in $\mathsf{s}^{\alpha_p}_i x$ $(0 < i \leq k)$.

Proof. It suffices to only consider $\star \in \{\prec^{\mathsf{w}}_n, \prec^{\mathsf{p}}_n, \prec^{\mathsf{l}}_n, \preceq^{\mathsf{w}}_n, \preceq^{\mathsf{p}}_n, \preceq^{\mathsf{l}}_n, \}$ for $n > 0$. As before, we leave the detailed reductions to Sections 4.8.4-4.8.7, and only highlight the key components different from those in the proof of Lemma 4.5.

Gap Order Delineation. The general reduction rule is to delineate a gap order to "indecomposable" intervals by inserting boundary terms. Let us first consider open gap order literals. Suppose both u and v are ordinary TA-terms, $u \prec^{\mathsf{w}}_n v$ $(n > 1)$ reduces to

$$\bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} u \prec^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(v^{\mathsf{w}})} \prec^{\mathsf{pl}}_{n_3} v \ . \tag{4.35}$$

Here is the intuition behind this reduction. $u \prec^{\mathsf{w}}_n v$ states that $u \prec^{\mathsf{w}} v$ and there is an increasing chain of length n from u to v. On this chain from u to v we have to go across two boundaries, $1^{\mathsf{w}}_{(u^{\mathsf{w}})}$ first and $0^{\mathsf{w}}_{(v^{\mathsf{w}})}$ second. The two boundaries delineate the chain into three segments and the sum of the lengths of the three segments should be greater than or equal to n. Since there are infinitely many such combinations, a naive representation would require infinitely many disjunctions of conjunctions of literals. However, we can find a finite *cover* (an equivalent finite subset) by requiring that the sum of the lengths of the three segments is just equal to n, because for any $k > 0$,

$$\forall n_1, \ldots, \forall n_k \Big(\sum_{i=1}^{k} n_i \geq n \ \rightarrow \ (\exists n'_1 \leq n_1), \ldots, (\exists n'_k \leq n_k) \sum_{i=1}^{k} n'_i = n \Big)$$

is valid in $\mathbb{N}$, and because for any $n, n' \in \mathbb{N}$, $n < n'$ implies that

$$\forall x, y \Big(x \prec^{\sharp}_n y \ \leftrightarrow \ (x \prec^{\sharp}_n y \ \vee \ x \prec^{\sharp}_{n'} y) \Big) \ ,$$

where $\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$, is valid in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$.

The reduction for rigid gap order literals is similar. For example, $u \leqq^{\mathsf{w}}_n v$ $(n > 1)$ reduces to

$$\bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} u \leqq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \leqq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(v^{\mathsf{w}})} \leqq^{\mathsf{pl}}_{n_3} v \ .$$

Recall that $x \prec^{\mathsf{pl}} y \equiv x \prec^{\mathsf{p}} y \vee x \prec^{\mathsf{l}} y$. Suppose we choose the branch

$$u \prec^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(v^{\mathsf{w}})} \prec^{\mathsf{p}}_{n_3} v \ . \tag{4.36}$$

Assuming u is of type α_p, v is of type α_q, we can further delineate (4.36) to

$$\bigvee_{\substack{n_1=n_{11}+n_{12} \\ n_3=n_{31}+n_{32}}} u \prec^{\mathsf{l}}_{n_{11}} 1^{\mathsf{p}}_{(u^{\mathsf{w}},p)} \prec^{\mathsf{p}}_{n_{12}} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(v^{\mathsf{w}})} \prec^{\mathsf{p}}_{n_{31}} 0^{\mathsf{p}}_{(v^{\mathsf{w}},q)} \prec^{\mathsf{pl}}_{n_{32}} v \ . \tag{4.37}$$

In general we delineate $u \prec^{\sharp}_n v$ by inserting m boundary terms $t_1, \ldots, t_m$ in between u and v

satisfying

$$u \preceq^{\sharp_0} t_1 \preceq^{\sharp_1} t_2 \preceq^{\sharp_2} \cdots \preceq^{\sharp_{m-1}} t_m \preceq^{\sharp_m} v \ , \tag{4.38}$$

where $\sharp_i \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$ $(i \leq m)$ and $\max\{\sharp_i \mid i \leq m\} = \sharp$. Recall that $\mathsf{l} \prec^{\mathsf{c}} \mathsf{p} \prec^{\mathsf{c}} \mathsf{w}$. For example, $\max\{\mathsf{w}, \mathsf{p}\} = \mathsf{w}$ and $\min\{\mathsf{p}, \mathsf{l}\} = \mathsf{l}$. It is not hard to argue that $u \prec_n^{\sharp} v$ is equivalent to

$$\bigvee_{n_0+\cdots+n_m=n} u \prec_{n_0}^{\sharp_0} t_1 \prec_{n_1}^{\sharp_1} t_2 \prec_{n_2}^{\sharp_2} \cdots \prec_{n_{m-1}}^{\sharp_{m-1}} t_m \prec_{n_m}^{\sharp_m} v \ . \tag{4.39}$$

In such a delineation, $\prec_0^{\sharp}$ can be read as $=$. This saves unnecessary splittings for $\preceq^{\sharp}$. However, it is not essential as the termination of our reductions does not rely on the decrease of gap counts.

Similarly, $u \preceqq_n^{\sharp} v$ can be delineated as

$$\bigvee_{n_0+\cdots+n_m=n} u \preceqq_{n_0}^{\sharp_0} t_1 \preceqq_{n_1}^{\sharp_1} t_2 \preceqq_{n_2}^{\sharp_2} \cdots \preceqq_{n_{m-1}}^{\sharp_{m-1}} t_m \preceqq_{n_m}^{\sharp_m} v \ . \tag{4.40}$$

The delineation of half open gap order literals is similar.

Termination The termination of these reductions does not rely on the decrease of gap counts. Rather, it relies on the fact that a gap order is eventually delineated to "indecomposable" p-intervals. A closed equality literal or a closed gap order literal can be reduced to a Presburger formula (Sections 4.8.1- 4.8.3). So eventually half open gap order literals and open gap order literals only involve TA-terms in the same p-interval; i. e., they are in the forms $x \prec_n^{\mathsf{l}} t$, $t \prec_n^{\mathsf{l}} x$, $x \preceqq_n^{\mathsf{l}} t$ and $t \preceqq_n^{\mathsf{l}} x$ $(n > 0)$ where x and t are of type α_p for some $\alpha_p = (\mathsf{s}_1^{\alpha_p}, \ldots, \mathsf{s}_k^{\alpha_p})$.

Let us first consider literals like $x \prec_n^{\mathsf{l}} t$. Suppose that t is an ordinary term. Then $x = \alpha_p(\mathsf{s}_1^{\alpha_p} x, \ldots, \mathsf{s}_k^{\alpha_p} x)$, $t = \alpha_p(\mathsf{s}_1^{\alpha_p} t, \ldots, \mathsf{s}_k^{\alpha_p} t)$, and $x \prec_n^{\mathsf{l}} t$ is equivalent to

$$\langle \mathsf{s}_1^{\alpha_p} x, \ldots, \mathsf{s}_k^{\alpha_p} x \rangle \prec_n^{k;\mathrm{kb}} \langle \mathsf{s}_1^{\alpha_p} t, \ldots, \mathsf{s}_k^{\alpha_p} t \rangle \ ,$$

which, by Lemma 4.7, further reduces to a formula in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$ in terms of $\mathsf{s}_i^{\alpha_p} x$ and $\mathsf{s}_i^{\alpha_p} t$ $(0 < i \leq k)$.

Now suppose that t is a boundary term, say $t \equiv \mathsf{0}^{\mathsf{w}}_{(m)}$. We can rewrite t as $\alpha_p(\mathsf{s}_1^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)}, \ldots, \mathsf{s}_k^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)})$. So $x \prec_n^{\mathsf{l}} t$ is equivalent to

$$\langle \mathsf{s}_1^{\alpha_p} x, \ldots, \mathsf{s}_k^{\alpha_p} x \rangle \prec_n^{k;\mathrm{kb}} \langle \mathsf{s}_1^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)}, \ldots, \mathsf{s}_k^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)} \rangle \ ,$$

which, by Lemma 4.7, further reduces to a formula in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$ in terms of $\mathsf{s}_i^{\alpha_p} x$ and $\mathsf{s}_i^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)}$ $(0 < i \leq k)$. Each $\mathsf{s}_i^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)}$ must be the smallest (w.r.t. $\prec^{\mathrm{kb}}$) of weight $(\mathsf{s}_i^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)})^{\mathsf{w}}$. By Lemma 4.3, $\mathsf{s}_1^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)}, \ldots, \mathsf{s}_k^{\alpha_p}\mathsf{0}^{\mathsf{w}}_{(m)}$ are simplified to boundary terms not occurring inside selectors, say $\mathsf{0}^{\mathsf{w}}_{(f_1(m))}, \ldots, \mathsf{0}^{\mathsf{w}}_{(f_k(m))}$ where $f_1, \ldots, f_k$ are integer functions definable in Presburger arithmetic (Section 4.7).

We have similar results for $t \prec_n^{\mathsf{l}} x$, $x \preceqq_n^{\mathsf{l}} t$ and $t \preceqq_n^{\mathsf{l}} x$.

Summary

1. Section 4.8.4 lists reductions of half-open stretchable gap order literals.
2. Section 4.8.5 lists reductions of half-open rigid gap order literals.
3. Section 4.8.6 lists reductions of open stretchable gap literals.
4. Section 4.8.7 lists reductions of open rigid gap literals. □

Lemma 4.7 (Reduction of Tuple Literals). Let U, V be k-tuples of the same weight, and

$$\star \in \{ =, \prec_n^{k;\mathsf{kb}}, \prec_n^{k;\mathsf{w}}, \prec_n^{k;\mathsf{p}}, \prec_n^{k;\mathsf{l}}, \prec_n^{k;\mathsf{pl}}, \preceqq_n^{k;\mathsf{kb}}, \preceqq_n^{k;\mathsf{w}}, \preceqq_n^{k;\mathsf{p}}, \preceqq_n^{k;\mathsf{l}}, \preceqq_n^{k;\mathsf{pl}} \} \qquad (k > 0, n > 0) \ .$$

1. If $U = \langle u_1, \ldots, u_k \rangle$ is an ordinary tuple, then $U \star V$ ($V \star U$) can be effectively reduced to an equivalent quantifier-free formula $\varphi(u_1, \ldots, u_k)$ (in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$) in which u_i $(0 < i \leq k)$ does not occur inside selectors.
2. If $U \star V$ ($V \star U$) is a closed tuple, i. e., both U and V are boundary tuples, then it can be effectively reduced to an equivalent Presburger formula.

Proof. There are many types of tuple literals, especially the closed tuple literals. However, as tuples are only used in the intermediate steps in the reduction, we only encounter a small portion of the combinations. The detailed reductions are given in Sections 4.8.8-4.8.15. Here we only highlight the key components of the proof.

Notations First recall that we define tuple relations between tuples of the same weight. So tuple relations are not only parameterized by k, the tuple length, but also parameterized by m, the total weight.

Second recall that we define suborders on tuples $\langle u_1, \ldots, u_k \rangle \prec^{k;\sharp} \langle v_1, \ldots, v_k \rangle$ ($\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$) as

$$u_1 \prec^{\sharp} v_1 \ \vee \ \Big(u_1 = v_1 \ \wedge \ \langle u_2, \ldots, u_k \rangle \prec^{k;\mathrm{kb}} \langle v_2, \ldots, v_k \rangle \Big) \ , \tag{4.41}$$

instead of as

$$\exists i (0 < i \leq k) \Big[\ u_i \ \prec^{\sharp} \ v_i \ \wedge \ \forall j (1 \leq j < i) u_j = v_j \ \Big] \ . \tag{4.42}$$

As we shall see soon, this technical choice gives reducts (reduced formulas) a uniform appearance.

Third recall that $\langle u_1, \ldots, u_k \rangle \prec_n^{k;\sharp} \langle v_1, \ldots, v_k \rangle$ is *proper* if $u_1 \prec^\sharp v_1$. We claim that improper orders can be decomposed to a Boolean combination of proper orders between k-tuples and proper orders between tuples of shorter length. We first show that we can always write $U \prec_n^{k;\sharp} V$ into the form

$$\langle u_1, \ldots, u_k \rangle \prec_n^{k;\sharp} \langle v_1, \ldots, v_k \rangle \ .$$

This trivially holds if both U and V are ordinary tuples. Now suppose U is a boundary tuple. We claim that $U = \langle v_1, \ldots, v_k \rangle$ where $u_1, \ldots, u_k$ are boundary terms. In fact we have

$$\begin{aligned}
\overrightarrow{1}^{k;\mathsf{kb}}_{(sum)} &= \langle\, 1^{\mathsf{w}}_{(\mathrm{CWL}^k_1(sum))}, 1^{\mathsf{w}}_{(\mathrm{CWL}^k_2(sum))}, \ldots, 1^{\mathsf{w}}_{(\mathrm{CWL}^k_k(sum))} \,\rangle \ , \\
\overrightarrow{1}^{k;\mathsf{w}}_{(sum,m)} &= \langle\, 1^{\mathsf{w}}_{(m)}, 1^{\mathsf{w}}_{(\mathrm{CWL}^{k-1}_1(sum-m))}, \ldots, 1^{\mathsf{w}}_{(\mathrm{CWL}^{k-1}_{k-1}(sum-m))} \,\rangle \ , \\
\overrightarrow{1}^{k;\mathsf{p}}_{(sum,m,p)} &= \langle\, 1^{\mathsf{w}}_{(m,p)}, 1^{\mathsf{w}}_{(\mathrm{CWL}^{k-1}_1(sum-m))}, \ldots, 1^{\mathsf{w}}_{(\mathrm{CWL}^{k-1}_{k-1}(sum-m))} \,\rangle \ , \\
\overrightarrow{0}^{k;\mathsf{kb}}_{(sum)} &= \langle\, 0^{\mathsf{w}}_{(\mathrm{CWS}^k_1(sum))}, 0^{\mathsf{w}}_{(\mathrm{CWS}^k_2(sum))}, \ldots, 0^{\mathsf{w}}_{(\mathrm{CWS}^k_k(sum))} \,\rangle \ , \\
\overrightarrow{0}^{k;\mathsf{w}}_{(sum,m)} &= \langle\, 0^{\mathsf{w}}_{(m)}, 0^{\mathsf{w}}_{(\mathrm{CWS}^{k-1}_1(sum-m))}, \ldots, 0^{\mathsf{w}}_{(\mathrm{CWS}^{k-1}_{k-1}(sum-m))} \,\rangle \ , \\
\overrightarrow{0}^{k;\mathsf{p}}_{(sum,m,p)} &= \langle\, 0^{\mathsf{w}}_{(m,p)}, 0^{\mathsf{w}}_{(\mathrm{CWS}^{k-1}_1(sum-m))}, \ldots, 0^{\mathsf{w}}_{(\mathrm{CWS}^{k-1}_{k-1}(sum-m))} \,\rangle \ ,
\end{aligned}$$

where $\mathrm{CWL}^k_i(n)$ (resp. $\mathrm{CWS}^k_i(n)$) gives the weight of the i^{th} component of the largest (resp. the smallest) k-tuple (w.r.t. $\prec^{k;\mathrm{kb}}$) of weight n (Section 4.7). The case that V is a boundary tuple is similar.

Therefore, $U \prec_n^{k;\sharp} V$ reduces to

$$\Big((u_1 = v_1) \ \wedge\ \langle u_2, \ldots, u_k \rangle \prec_n^{k;\mathrm{kb}} \langle v_2, \ldots, v_k \rangle\Big) \ \vee\ \Big((u_1 \prec^\sharp v_1) \ \wedge\ \langle u_1, \ldots, u_k \rangle \prec_n^{k;\sharp} \langle v_1, \ldots, v_k \rangle\Big) \ ,$$

which contains two disjuncts; the first is a tuple order on tuples of a shorter length and the second is a proper tuple order on tuples of the same length. Iteratively applying the process, an improper k-tuple order can be decomposed to a Boolean combination of proper k'-tuple orders for $k' \leq k$. Therefore, it suffices to show *proper reductions*, the reductions of proper orders on tuples of the same weight, which are given in Sections 4.8.8-4.8.15. All reduction rules therefore contain an implicit side condition $u_1 \prec^\sharp v_1$ or $u_1 \preceq^\sharp v_1$ in the redex (target formula).

Gap Order Delineation and Tuple Length Reduction. The general reduction rule is the same as in the proof of Lemma 4.6, namely, using boundary tuples to delineate a tuple order to "indecomposable" intervals. For example, Let $U = \langle u_1, \ldots, u_k \rangle$, $V = \langle v_1, \ldots, v_k \rangle$ be ordinary tuples, and

$sum = \sum_{i=1}^{k} u_i^{\mathsf{w}} = \sum_{i=1}^{k} v_i^{\mathsf{w}}$. $U \prec_n^{k;\mathsf{w}} V$ $(n > 0)$ reduces to (reduction (4.339), copied below)

$$\bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \langle u_1, \ldots, u_k\rangle \prec_{n_1}^{k;\mathrm{pl}} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \prec_{n_2}^{k;\mathsf{w}} \overline{0}^{k;\mathsf{w}}_{(sum,v_1^{\mathsf{w}})} \prec_{n_3}^{k;\mathrm{pl}} \langle v_1, \ldots, v_k\rangle \ . \tag{4.339}$$

Now it is clear why we define $\langle u_1, \ldots, u_k\rangle \prec_n^{k;\sharp} \langle v_1, \ldots, v_k\rangle$ by (4.41), other than by (4.42). Suppose we did it in the other way. Then in the reduction of (4.339), we are not able to have $\langle u_1, \ldots, u_k\rangle \prec_{n_1}^{k;\mathrm{pl}} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})}$, because the order between $\langle u_1, \ldots, u_k\rangle$ and $\overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})}$ may be determined by the weight of the i^{th} components for some $i > 1$.

Besides gap order delineation, we also need to decompose a tuple gap order using tuple gap orders of shorter lengths and term gap orders. For example, $\langle u_1, \ldots, u_k\rangle \prec_{n_1}^{k;\mathsf{p}} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})}$, which is a conjunct in a disjunct of (4.339), reduces to (reduction (4.322), copied below)

$$\begin{aligned}\bigvee_{\substack{n_{11}(n_{12}+1)+n_{13}\geq n_1,\\ n_{11},n_{12},n_{13}\leq n_1}} u_1 \prec_{n_{11}}^{\mathsf{p}} 1^{\mathsf{w}}_{(u_1^{\mathsf{w}})} \wedge \overline{0}^{k-1;\mathrm{kb}}_{(sum-u_1^{\mathsf{w}})} \prec_{n_{12}}^{k-1;\mathrm{kb}} \overline{1}^{k-1;\mathrm{kb}}_{(sum-u_1^{\mathsf{w}})}\\ \wedge \langle u_2, \ldots, u_k\rangle \prec_{n_{13}}^{k-1;\mathrm{kb}} \overline{1}^{k-1;\mathrm{kb}}_{(sum-u_1^{\mathsf{w}})} \ .\end{aligned} \tag{4.322}$$

We briefly explain the reason behind (4.322). The tuples in between $\langle u_1, \ldots, u_k\rangle$ and $\overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})}$ can be divided into $n_{11} + 1$ segments where $n_{11} + 1$ is the number of ground TA-terms in between u_1 and $1^{\mathsf{w}}_{(u_1^{\mathsf{w}})}$ (including u_1 and $1^{\mathsf{w}}_{(u_1^{\mathsf{w}})}$). Each of the first n_{11} segments contains $n_{12} + 1$ k-tuples, where $n_{12} + 1$ is the number of $(k-1)$-tuples of weight $sum - u_1^{\mathsf{w}}$. The last segment contains n_{13} k-tuples because n_{13} is the number of $(k-1)$-tuples greater than or equal to $\langle u_2, \ldots, u_k\rangle$ but less than $\overline{1}^{k-1;\mathrm{kb}}_{(sum-u_1^{\mathsf{w}})}$. We require $n_{11}(n_{12} + 1) + n_{13} \geq n_1$ and $n_{11}, n_{12}, n_{13} \leq n_1$ to have finitely many disjunctions. It suffices because for any $k > 0$ and for any polynomial $P(x_1, \ldots, x_k)$ only containing positive coefficients, the following formula is valid in $\mathbb{N}$.

$$\forall m_1, \ldots, \forall m_k \Big[P(m_1, \ldots, m_k) \geq m \ \rightarrow\ \exists m_1' \leq m, \ldots, \exists m_k' \leq m \Big(\bigwedge_{i=1}^{k} m_i' \leq m_i \ \wedge\ P(m_1', \ldots, m_k') \geq m \Big) \Big] \ .$$

The reduction of $U \leqq_n^{k;\sharp} V$ is similarly defined with stretchable gap orders replaced by corresponding rigid gap orders. For example, $\langle u_1, \ldots, u_k\rangle \leqq_{n_1}^{k;\mathsf{p}} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})}$, reduces to (reduction (4.334), copied below)

$$\begin{aligned}\bigvee_{\substack{n_{11}(n_{12}+1)+n_{13}=n_1,\\ n_{11},n_{12},n_{13}\leq n_1}} u_1 \leqq_{n_{11}}^{\mathsf{p}} 1^{\mathsf{w}}_{(u_1^{\mathsf{w}})} \wedge \overline{0}^{k-1;\mathrm{kb}}_{(sum-u_1^{\mathsf{w}})} \leqq_{n_{12}}^{k-1;\mathrm{kb}} \overline{1}^{k-1;\mathrm{kb}}_{(sum-u_1^{\mathsf{w}})}\\ \wedge \langle u_2, \ldots, u_k\rangle \leqq_{n_{13}}^{k-1;\mathrm{kb}} \overline{1}^{k-1;\mathrm{kb}}_{(sum-u_1^{\mathsf{w}})} \ .\end{aligned} \tag{4.334}$$

Here we require $n_{11}(n_{12} + 1) + n_{13} = n_1$ instead of $n_{11}(n_{12} + 1) + n_{13} \geq n_1$.

As an example for closed relations, we consider $\overline{1}^{k;\mathsf{w}}_{(sum,m)} \prec^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')}$ which reduces to (reduction (4.243), copied below)

$$\left(\begin{array}{l} \qquad\qquad (n = 1 \ \rightarrow \ m < m') \\ \qquad\qquad\qquad \wedge \\ n > 1 \ \rightarrow \\ \bigvee\limits_{0<r<n} \left(\begin{array}{c} m < \mathrm{CW}^k_1(sum,m,m') < \cdots < \mathrm{CW}^k_r(sum,m,m') < m' \\ \wedge \\ \bigvee\limits_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_i > 0}} \bigwedge\limits_{0<i\leq r} \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \prec^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \end{array} \right) \end{array} \right) . \tag{4.243}$$

We explain the meaning of (4.243). Since $\overline{1}^{k;\mathsf{w}}_{(sum,m)}$ is the largest k-tuple of weight sum with the first component having weight m and $0^{\mathsf{w}}_{(m')}$ is the smallest tuple of weight of sum with the first component having weight m', $\overline{1}^{k;\mathsf{w}}_{(sum,m)} \prec^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')}$ means that there are at least $n-1$ k-tuples of weight sum in between $\overline{1}^{k;\mathsf{w}}_{(sum,m)}$ and $\overline{0}^{k;\mathsf{w}}_{(sum,m')}$ (but including neither $\overline{1}^{k;\mathsf{w}}_{(sum,m)}$ nor $\overline{0}^{k;\mathsf{w}}_{(sum,m')}$) and in each of these k-tuples the weight of the first component is greater than m but smaller than m'. If $n = 1$, then obviously $\overline{1}^{k;\mathsf{w}}_{(sum,m)} \prec^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')}$ reduces to $m < m'$. If $n > 1$, then for some r $(0 < r < n)$, the $n-1$ k-tuples distribute into r different levels according to their weights, which is formally represented as

$$\bigvee_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_i \geq 1}} \ \bigwedge_{0<i\leq r} \ \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \prec^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \ ,$$

where $\mathrm{CW}^k_i(n,m,m')$ gives the i^{th} smallest integer in (m,m') which is the weight of a component in a k-tuple of weight n (Section 4.7), and

$$\overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \prec^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))}$$

states that there are n_i k-tuples of weight sum with the first component of weight $\mathrm{CW}^k_i(sum,m,m')$. We require $\sum_{i=1}^{r} n_i = n-1$ and $n_i \geq 1$ for $1 \leq i \leq r$.

The reduction of $\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')}$ is similarly obtained as (reduction (4.291), copied below)

$$\left(\begin{array}{l} n = 1 \ \rightarrow \ \Big(m < m' \ \wedge \ \forall z \Big(m < z < m' \ \rightarrow \ \neg \, \mathrm{IsTW}^k(m)\Big)\Big) \\ \qquad\qquad\qquad \wedge \\ n > 1 \ \rightarrow \\ \displaystyle\bigvee_{0<r<n} \left(\begin{array}{c} m < \mathrm{CW}^k_1(sum,m,m') < \cdots < \mathrm{CW}^k_r(sum,m,m') < m' \\ \wedge \\ \neg \Big(m < \mathrm{CW}^k_1(sum,m,m') < \cdots < \mathrm{CW}^k_{r+1}(sum,m,m') < m'\Big) \\ \wedge \\ \displaystyle\bigvee_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_i > 0}} \bigwedge_{0<i\leq r} \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \leqq^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \end{array}\right) \end{array}\right) . \tag{4.291}$$

Here we need to add two more conjuncts. When $n = 1$ we need

$$\forall z \Big(m < z < m' \ \rightarrow \ \neg \, \mathrm{IsTW}^k(m)\Big)$$

to make sure that there exists no k-tuple whose weight is in (m, m'). When $n > 1$, we need

$$\neg \Big(m < \mathrm{CW}^k_1(sum,m,m') < \cdots < \mathrm{CW}^k_{r+1}(sum,m,m') < m'\Big)$$

to guarantee that for a fixed $r \in (0, n)$ there are exactly r legitimate (tuple) weights in (m, m').

Termination As Lemma 4.6, the termination of these reductions does not rely on the decrease of gap counts. Rather, a tuple gap order is first delineated to "indecomposable" intervals which is in turn reduced to tuple gap orders of shorter lengths. Recall that tuple gap orders of length 1 are essentially term gap orders. Repeatedly applying the reduction, eventually only term gap orders $<^{\sharp}_{n}$ and $\leqq^{\sharp}_{n}$ ($\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$) appear, i. e., a tuple gap order reduces to a formula in $\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$, and hence the reduction terminates. In particular, a closed tuple gap order or a closed tuple equality literal reduces to a Presburger formula.

Well-definedness of Boundary Tuples Reductions listed in Sections 4.8.8-4.8.15 implicitly rely on the assumption that boundary tuples are well-defined. This assumption holds because parameters of boundary tuples come from weights of well-defined TA-terms.

Reduction of Literals Involving $\overline{0}^{k;\mathrm{kb}}_{(sum)}$ or $\overline{1}^{k;\mathrm{kb}}_{(sum)}$. There is no separate list for reductions of tuple gap orders involving $\overline{0}^{k;\mathrm{kb}}_{(sum)}$ or $\overline{1}^{k;\mathrm{kb}}_{(sum)}$, as they can be expressed, respectively, by

$$\overline{0}^{k;\mathrm{kb}}_{(sum)} = \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{MinCW}^k(sum))} \ , \qquad \overline{1}^{k;\mathrm{kb}}_{(sum)} = \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{MaxCW}^k(sum))} \ ,$$

where $\mathrm{MinCW}^k(sum)$ (resp. $\mathrm{MaxCW}^k(sum)$) gives the smallest (resp. the largest) integer in $(0, sum{+}1)$ that is the weight of a component in a k-tuple having weight sum (Section 4.7).

The Number of Open Gap Order Literals. As we mentioned before, tuple gap orders reduce to term gap orders. The reduction does not increase the number of open term gap order literals. The only places generating new open term gap order literals are the reduction of $\langle u_1, \ldots, u_k\rangle \prec^{k;\mathsf{l}}_n \langle v_1, \ldots, v_k\rangle$ (reduction (4.341)) and the reduction of $\langle u_1, \ldots, u_k\rangle \preceq^{k;\mathsf{l}}_n \langle v_1, \ldots, v_k\rangle$ (reduction (4.344)). Consider reduction (4.341) (copied below)

$$\bigvee_{\substack{n_1+n_2+n_3=n\\ n_3>0}} \left(\begin{array}{c} \langle u_2, \ldots, u_k\rangle \prec^{k-1;\mathrm{kb}}_{n_1} \overline{1}^{k-1;\mathrm{kb}}_{(rem)} \\ \wedge \\ \overline{0}^{k-1;\mathrm{kb}}_{(rem)} \prec^{k-1;\mathrm{kb}}_{n_2} \langle v_2, \ldots, v_k\rangle \\ \wedge \\ \displaystyle\bigvee_{\substack{(m_1+1)(m_2-1)\geq(n_3-1)\\ m_2>0, m_1, m_2<n_3}} \left(\begin{array}{c} \overline{0}^{k-1;\mathrm{kb}}_{(rem)} \prec^{k-1;\mathrm{kb}}_{m_1} \overline{1}^{k-1;\mathrm{kb}}_{(rem)} \\ \wedge \\ u_1 \prec^{\mathsf{l}}_{m_2} v_1 \end{array} \right) \end{array} \right) \ , \tag{4.341}$$

where $rem = \sum_{i=2}^{k} u_i^{\mathsf{w}} = \sum_{i=2}^{k} v_i^{\mathsf{w}}$. It is clear that in (4.341), each disjunct (conjunction of literals) only contains one open gap order literal, namely $u_1 \prec^{\mathsf{l}}_{m_2} v_1$. Since the reduction is triggered by an open gap order literal, there is no increase to the number of open gap order literals. Though $u_1 \prec^{\mathsf{l}} v_1$ is an implicit open gap order literal in the redex, it is not included in any reducts, as it has been replaced by the more precise relation $u_1 \prec^{\mathsf{l}}_{m_2} v_1$.

The case for reduction (4.344) is similar.

Summary

1. Section 4.8.8 lists reductions of closed tuple equality literals.
2. Section 4.8.9 lists reductions of half-open tuple equality literals.
3. Section 4.8.10 lists reductions of closed stretchable tuple gap order literals.
4. Section 4.8.11 lists reductions of closed rigid tuple gap order literals.

5. Section 4.8.12 lists reductions of half-open stretchable tuple gap order literals.
6. Section 4.8.13 lists reductions of half-open rigid tuple gap order literals.
7. Section 4.8.14 lists reductions of open stretchable tuple gap order literals.
8. Section 4.8.15 lists reductions of open rigid tuple gap order literals. □

Lemma 4.8 (Elimination of Term Quantifiers). Let x be a term variable, $\varphi_{\text{kb}^+}(x)$ a conjunction of literals in $\mathscr{L}_{\text{kb}^+}$ with $\text{depth}^{\varphi_{\text{kb}^+}}(x) = 0$, and $\varphi_{\mathbb{Z}}(x)$ a Presburger formula in which x occurs inside the weight function. Then

$$(\exists x : \mathbb{T})\Big[\ \varphi_{\text{kb}^+}(x)\ \wedge\ \varphi_{\mathbb{Z}}(x)\ \Big]$$

can be effectively reduced to $\varphi'_{\text{kb}^+} \wedge \varphi'_{\mathbb{Z}}$ in which x does not occur and φ'_{kb^+} is quantifier-free.

Proof. The antecedent of the lemma only requires that all *term occurrences* of x have depth 0 and so x may have arbitrarily many integer occurrences of the form $(Lx)^{\mathsf{w}}$ where L is a (possibly empty) selector sequence. We divide the proof into two parts.

Part 1. $\varphi_{\text{kb}^+}(x)$ contains $x = t$ where t may contain x. We rewrite $(\exists x : \mathbb{T})[\varphi_{\text{kb}^+}(x) \wedge \varphi_{\mathbb{Z}}(x)]$ as

$$(\exists x : \mathbb{T})\Big[\ x = t\ \wedge\ \varphi(x)\ \Big]\ , \tag{4.43}$$

where $\varphi(x)$ is a conjunction of literals in $\mathscr{L}^{\mathbb{Z}}_{\text{kb}^+}$. We consider four cases.

1. t does not contain x. Then (4.43) simplifies to $\varphi(t)$.
2. $t \equiv x$. Then (4.43) simplifies to $(\exists x)\varphi(x)$.
3. $t \equiv Lx$ where L is a non-empty selector block. Then (4.43) simplifies to false, since we assume that terms like Lx are proper (Definition 2.2).
4. t is a boundary term in which $(L_i x)^{\mathsf{w}}$ $(i \leq n)$ occur in an integer function. In this case we assume that (4.43) has the form

$$(\exists x : \mathbb{T})\Big[\ x = t[f((L_0x)^{\mathsf{w}}, \ldots, (L_nx)^{\mathsf{w}})]\ \wedge\ \varphi(x)\ \Big]\ , \tag{4.44}$$

where $L_0x \equiv x$ (i. e., $L_0 \equiv \emptyset$), $L_0x, \ldots, L_nx$ enumerate all selector terms containing x (including improper selector terms inside selectors), and $f((L_0x)^{\mathsf{w}}, \ldots, (L_nx)^{\mathsf{w}})$ denotes the maximum integer term in t. We assume $t[f((L_0x)^{\mathsf{w}}, \ldots, (L_nx)^{\mathsf{w}})]$ is a well-defined boundary term (Section 4.7). Otherwise $t = \bot$ and hence (4.43) reduces to $\varphi(\bot)$. The set of terms $\{L_ix \mid i \leq n\}$ corresponds a

tree with root x. We assume that this tree is *sibling complete*, that is, sibling nodes coexist. For example, a formula in List is sibling complete if for any term t it either contains both car(t) and cdr(t) or contains neither. In general, if we need $L'x$ to appear, we can just set

$$f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}},(L'x)^{\mathsf{w}}) := f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}},x^{\mathsf{w}}) + (L'x)^{\mathsf{w}} - (L'x)^{\mathsf{w}} \ .$$

It is easily seen that (4.44) is equivalent to

$$(\exists x:\mathbb{T})\Big[\ x = t[f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}})]\ \wedge\ \varphi(t[f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}})])\ \Big]\ , \tag{4.45}$$

where x only has one term occurrence, namely, as the left-hand side of

$$x = t[f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}})]\ . \tag{4.46}$$

Note that the substitution that we used to transform (4.44) to (4.45) may put a boundary term inside a selector term. In fact, it is only this type of substitution that makes a boundary term appear inside a selector. However, Lemma 4.3 eliminates this superficial complication. We claim that (4.45) is equivalent to

$$(\exists(L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}}:\mathbb{Z})\Big[\ \varphi(t[f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}})])\ \wedge\ \varphi_\Delta((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}})\ \Big]\ , \tag{4.47}$$

where $(L_ix)^{\mathsf{w}}$ are pseudo integer variables, and φ_Δ is the integer constraint on weights of subterms of x satisfying the following conditions.

(a) φ_Δ requires that x be a legitimate term.

i. For each leaf node L_ix, φ_Δ includes $\mathrm{Tree}^{\alpha_p}((L_ix)^{\mathsf{w}})$, where $\mathrm{type}(L_ix) = \alpha_p$. This states that each leaf has a well-defined weight.

ii. For each non-leaf node L_ix, φ_Δ includes

$$(L_ix)^{\mathsf{w}} = \mathsf{w}(\alpha_p) + \sum_{j=1}^{k}(\mathsf{s}_j^{\alpha_p}L_ix)^{\mathsf{w}}\ ,$$

where $\mathrm{type}(L_ix) = \alpha_p = (\mathsf{s}_1^{\alpha_p},\ldots,\mathsf{s}_k^{\alpha_p})$ and $\mathsf{s}_1^{\alpha_p}L_ix,\ldots,\mathsf{s}_k^{\alpha_p}L_ix$ are immediate children of L_ix. This states that the weight of a tree is the sum of the weight of the root and the weights of the immediate children of the root.

(b) φ_Δ requires that x be a legitimate boundary term. For example, suppose

$$t \equiv 0^{\mathsf{w}}_{(f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}}))}\ . \tag{4.48}$$

i. φ_Δ includes $x^{\mathsf{w}} = f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}})$.

ii. Each node $L_i x$ must be the smallest (w.r.t. $\prec^{\text{kb}}$) of weight $(L_i x)^{\mathsf{w}}$. If $\text{type}(L_i x) = \alpha_p$, then there exists no p' such that $p < p' \leq |\Sigma|$ and $\text{Tree}^{\alpha_{p'}}((L_i x)^{\mathsf{w}})$ holds. This constraint is formally expressed as

$$\bigwedge_{p<p'\leq|\Sigma|} \neg\, \text{Tree}^{\alpha_{p'}}((L_i x)^{\mathsf{w}}) \ .$$

iii. Each non-leaf node must have the smallest tuple of children (w.r.t. $\prec^{k;\text{kb}}$). If $L_i x$ has k immediate siblings $\mathsf{s}_1^{\alpha_p} L_i x, \ldots, \mathsf{s}_k^{\alpha_p} L_i x$ (in the order from left to right), we require that $\langle \mathsf{s}_1^{\alpha_p} L_i x, \ldots, \mathsf{s}_k^{\alpha_p} L_i x \rangle$ be the smallest k-tuple (w.r.t. $\prec^{k;\text{kb}}$) of weight $n = \sum_{j=1}^{k} (\mathsf{s}_j^{\alpha_p} L_i x)^{\mathsf{w}}$. This constraint is formally expressed as

$$\bigwedge_{0<j\leq k} (\mathsf{s}_j^{\alpha_p} L_i x)^{\mathsf{w}} = \text{CWS}_j^k(n) \ ,$$

where $\text{CWS}_j^k(n)$ gives the weight of the j^{th} component of the smallest k-tuple (w.r.t. $\prec^{k;\text{kb}}$) of weight n (Section 4.7).

Similarly we can obtain constraints for

$$t \equiv 0^{\mathsf{p}}_{(f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}}),p)} \ , \qquad t \equiv 1^{\mathsf{w}}_{(f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}}))} \ , \qquad t \equiv 1^{\mathsf{p}}_{(f((L_0x)^{\mathsf{w}},\ldots,(L_nx)^{\mathsf{w}}),p)} \ .$$

The intuition behind this reduction is that when a term x occurs inside the weight function, all its term properties (that are expressible in the formal language) can be fully characterized by integer properties that are expressible in $\mathscr{L}_{\mathbb{Z}}$. We therefore view $(L_i x)^{\mathsf{w}}$ as syntactical integer variables. By variable renaming we transform (4.47) to

$$(\exists z_0, \ldots, z_n : \mathbb{Z}) \Big[\ \varphi(t[f(z_0, \ldots, z_n)]) \ \wedge\ \varphi_\Delta(z_0, \ldots, z_n) \ \Big] \ . \tag{4.49}$$

We show equivalence of (4.45) and (4.49). Suppose x is a ground boundary term satisfying (4.45), then $\mathsf{w}(L_0 x), \ldots, \mathsf{w}(L_n x)$ clearly satisfy $\varphi_\Delta((L_0x)^{\mathsf{w}}, \ldots, (L_nx)^{\mathsf{w}})$ that describes integer constraints on $L_0 x, \ldots, L_n x$. Therefore, (4.45) implies (4.49). For the reverse direction, let $z_0, \ldots, z_n$ be integers satisfying (4.49). Without loss of generality we assume t to be as in (4.48). We obtain an x by setting $L_i x := 0^{\mathsf{p}}(z_i, p)$ for each leaf node $L_i x$ with type α_p. Such x is clearly well-defined ground boundary term $0^{\mathsf{w}}_{(f(z_0,\ldots,z_n))}$ due to the fact that $\varphi_\Delta(z_0, \ldots, z_n)$ is a complete integer constraints on weights of all subterms of $0^{\mathsf{w}}_{(f(z_0,\ldots,z_n))}$. Therefore, (4.45) is so satisfied.

Part 2. $\varphi_{\text{kb}^+}(x)$ does not contain equalities like $x = t$. Since x is the only term occurrence of x in φ_{kb^+}, we can move other literals not containing x out of $(\exists x : \mathbb{T})$, and hence we assume that φ_{kb^+} is

in the form

$$\bigwedge_{0<i\le l} u_i \preceq_i x \wedge \bigwedge_{0<j\le m} x \preceq_{l+j} v_j \ , \tag{4.50}$$

where $\preceq_i$ $(0 < i \le l + m)$ are gap orders. By Lemma 4.12, φ_{kb^+} is equivalent to a disjunction of delineated gap order completions, each of which contains no more open gap order literals than $\varphi_{\mathrm{kb}^+}(x)$ does. Now we assume $\varphi_{\mathrm{kb}^+}(x)$ is a DGOC, denoted by

$$\mathrm{DGOC}(x, (L_0x)^{\mathsf{w}}, \ldots, (L_nx)^{\mathsf{w}}) \ ,$$

where (as in Part 1) $L_0 \equiv x$, $L_0x, \ldots, L_nx$ enumerate *all* selector terms containing x and form a sibling-complete tree with x being the root, and $(L_0x)^{\mathsf{w}}, \ldots, (L_nx)^{\mathsf{w}}$ enumerate all integer occurrences of x that may occur in $\varphi_{\mathbb{Z}}(x)$ as well as inside boundary terms in $\varphi_{\mathrm{kb}^+}(x)$. Let $\{x\}^{\mathsf{w}}$ abbreviate $(L_0x)^{\mathsf{w}}, \ldots, (L_nx)^{\mathsf{w}}$. Since $\mathrm{DGOC}(x, \{x\}^{\mathsf{w}})$ is delineated, we assume that $\mathrm{DGOC}(x, \{x\}^{\mathsf{w}})$ has the form

$$\underbrace{t_1 \preceq_1 \cdots \preceq_{i-2} t_{i-1}}_{\mathrm{Head}(\{x\}^{\mathsf{w}})} \preceq_{i-1} x \preceq_i \underbrace{t_{i+1} \preceq_{i+1} \cdots \preceq_{n-1} t_n}_{\mathrm{Tail}(\{x\}^{\mathsf{w}})} \ ,$$

where $\preceq_{i-1} \in \{\prec^{\sharp}_{n_1}, \preceq^{\sharp}_{n_1}\}$, $\preceq_i \in \{\prec^{\natural}_{n_2}, \preceq^{\natural}_{n_2}\}$, $\sharp \equiv \natural \in \{\mathsf{l}, \mathsf{pl}\}$, and $\mathrm{Head}(\{x\}^{\mathsf{w}})$ (resp. $\mathrm{Tail}(\{x\}^{\mathsf{w}})$) denote the linear order before x (resp. after x) in the above sequence. Without loss of generality, we further assume $\preceq_{i-1}$ is $\prec^{\mathsf{l}}_{n_1}$ and $\preceq_i$ is $\prec^{\mathsf{l}}_{n_2}$ for $n_1, n_2 > 0$. Accordingly, $\mathrm{DGOC}(x, \{x\}^{\mathsf{w}})$ has the form

$$\mathrm{Head}(\{x\}^{\mathsf{w}}) \prec^{\mathsf{l}}_{n_1} x \prec^{\mathsf{l}}_{n_2} \mathrm{Tail}(\{x\}^{\mathsf{w}}) \ .$$

We write $\mathrm{Head}(\{x\}^{\mathsf{w}}) \prec^{\sharp}_{n_1+n_2} \mathrm{Tail}(\{x\}^{\mathsf{w}})$ to denote

$$\underbrace{t_1 \preceq_1 \cdots \preceq_{i-2} t_{i-1}}_{\mathrm{Head}(\{x\}^{\mathsf{w}})} \prec^{\mathsf{l}}_{n_1+n_2} \underbrace{t_{i+1} \preceq_{i+1} \cdots \preceq_{n-1} t_n}_{\mathrm{Tail}(\{x\}^{\mathsf{w}})} \ ,$$

the ordering obtained from $\mathrm{DGOC}(x, \{x\}^{\mathsf{w}})$ by removing the *term occurrence* of x and accordingly joining two gap orders before and after such occurrence. We claim that

$$(\exists x : \mathbb{T}) \Big[\mathrm{Head}(\{x\}^{\mathsf{w}}) \prec^{\mathsf{l}}_{n_1} x \prec^{\mathsf{l}}_{n_2} \mathrm{Tail}(\{x\}^{\mathsf{w}}) \wedge \varphi_{\mathbb{Z}}(\{x\}^{\mathsf{w}}) \Big] \tag{4.51}$$

is equivalent to

$$(\exists \{x\}^{\mathsf{w}} : \mathbb{Z}) \Big[\mathrm{Head}(\{x\}^{\mathsf{w}}) \prec^{\mathsf{l}}_{n_1+n_2} \mathrm{Tail}(\{x\}^{\mathsf{w}}) \wedge \varphi_{\Delta}(\{x\}^{\mathsf{w}}) \wedge \varphi_{\mathbb{Z}}(\{x\}^{\mathsf{w}}) \Big] \ , \tag{4.52}$$

where $\{x\}^{\mathsf{w}}$ are pseudo integer variables as in Part 1, $\varphi_{\Delta}(\{x\}^{\mathsf{w}})$ consists of $x^{\mathsf{w}} = (t_{i-1})^{\mathsf{w}}$, and integer constraints requiring that $L_0x, \ldots, L_nx$ form a legitimate tree (condition (4a) for $\varphi_{\Delta}(\{x\}^{\mathsf{w}})$ in Part 1). By variable renaming we rewrite (4.52) as

$$(\exists \bar{\mathbf{z}} : \mathbb{Z})\Big[\ \mathrm{Head}(\bar{\mathbf{z}}) \ \prec^{\mathsf{I}}_{n_1+n_2}\ \mathrm{Tail}(\bar{\mathbf{z}}) \ \wedge\ \varphi_{\Delta}(\bar{\mathbf{z}}) \ \wedge\ \varphi_{\mathbb{Z}}(\bar{\mathbf{z}})\ \Big]\ , \tag{4.53}$$

where $\bar{\mathbf{z}} = \{z_0, \ldots, z_n\}$.

We show equivalence of (4.51) and (4.53). It is clear that (4.51) is equivalent to

$$(\exists x : \mathbb{T})(\exists \bar{\mathbf{z}} : \mathbb{Z})\Big[\ \varphi_{\Delta}(\{x\}^{\mathsf{w}}) \ \wedge\ \varphi_{\mathbb{Z}}(\{x\}^{\mathsf{w}}) \ \wedge\ \mathrm{Head}(\{x\}^{\mathsf{w}}) \ \prec^{\mathsf{I}}_{n_1}\ x \ \prec^{\mathsf{I}}_{n_2}\ \mathrm{Tail}(\{x\}^{\mathsf{w}}) \ \wedge\ \bar{\mathbf{z}} = \{x\}^{\mathsf{w}}\ \Big]\ , \tag{4.54}$$

where $\bar{\mathbf{z}} = \{x\}^{\mathsf{w}}$ stands for $\bigwedge_{i=0}^{n} z_i = (L_ix)^{\mathsf{w}}$. We can rewrite (4.54) to

$$(\exists \bar{\mathbf{z}} : \mathbb{Z})(\exists x : \mathbb{T})\Big[\ \varphi_{\Delta}(\bar{\mathbf{z}}) \ \wedge\ \varphi_{\mathbb{Z}}(\bar{\mathbf{z}}) \ \wedge\ \mathrm{Head}(\bar{\mathbf{z}}) \ \prec^{\mathsf{I}}_{n_1}\ x \ \prec^{\mathsf{I}}_{n_2}\ \mathrm{Tail}(\bar{\mathbf{z}}) \ \wedge\ \bar{\mathbf{z}} = \{x\}^{\mathsf{w}}\ \Big]\ , \tag{4.55}$$

which is equivalent to (4.53) after eliminating x using the fact that weight properties about x and its subterms are completely characterized by $\varphi_{\mathbb{Z}}(\{x\}^{\mathsf{w}})$. Now by Lemma 4.9 we eliminate $\exists \bar{\mathbf{z}}$ from (4.53), obtaining $\varphi'_{\mathrm{kb}^+} \wedge \varphi'_{\mathbb{Z}}$ in which x does not occur and φ'_{kb^+} is quantifier-free, as desired.

The elimination procedure needs no essential change for other choices of $\trianglelefteq_{i-1}$ and $\trianglelefteq_i$. We only note that if both $\trianglelefteq_{i-1}$ and $\trianglelefteq_i$ are rigid gap orders, then the order between $\mathrm{Head}(\{x\}^{\mathsf{w}})$ and $\mathrm{Tail}(\{x\}^{\mathsf{w}})$ should be rigid too. For example, let $\trianglelefteq_{i-1}$ be $\leqq^{\mathsf{I}}_{n_1}$ and $\trianglelefteq_i$ be $\leqq^{\mathsf{I}}_{n_2}$. We have

$$(\exists x : \mathbb{T})\Big[\ \mathrm{Head}(\{x\}^{\mathsf{w}}) \ \leqq^{\mathsf{I}}_{n_1}\ x \ \leqq^{\mathsf{I}}_{n_2}\ \mathrm{Tail}(\{x\}^{\mathsf{w}}) \ \wedge\ \varphi_{\mathbb{Z}}(\{x\}^{\mathsf{w}})\ \Big]\ ,$$

which reduces to

$$(\exists \bar{\mathbf{z}} : \mathbb{Z})\Big[\ \mathrm{Head}(\bar{\mathbf{z}}) \ \leqq^{\mathsf{I}}_{n_1+n_2}\ \mathrm{Tail}(\bar{\mathbf{z}}) \ \wedge\ \varphi_{\Delta}(\bar{\mathbf{z}}) \ \wedge\ \varphi_{\mathbb{Z}}(\bar{\mathbf{z}})\ \Big]\ .$$

Finally note that we carry out the elimination even if x has other integer occurrences with depths greater than 0. We could have strengthened the precondition that requires that *all* occurrences of x has depth 0. To satisfy this precondition, however, the simplification procedure, step (1c) in Algorithm 4.2 has to include a sub-procedure for decomposing literals like $x = t$ using proper selector terms containing x. We choose to hide this complexity in the proof rather than exposing it in the algorithm. □

Lemma 4.9 (Elimination of Integer Quantifiers). Let z be an integer variable, $\varphi_{\mathrm{kb}^+}(z)$ a conjunction of literals in $\mathscr{L}_{\mathrm{kb}^+}$ where z occurs inside boundary terms, and $\varphi_{\mathbb{Z}}(z)$ a Presburger formula. Then

$$(\exists z : \mathbb{Z})\Big[\ \varphi_{\mathrm{kb}^+}(z) \ \wedge\ \varphi_{\mathbb{Z}}(z)\ \Big]$$

can be effectively reduced to $\varphi'_{\mathrm{kb}^+} \wedge \varphi'_{\mathbb{Z}}$ where no z occurs and φ'_{kb^+} is quantifier-free.

Proof. Let z, $\varphi_{\mathrm{kb}^+}(z)$ and $\varphi_{\mathbb{Z}}(z)$ be as stated above. Let us assume z has n occurrences in $\varphi_{\mathrm{kb}^+}(z)$, namely in n maximum integer functions enumerated below

$$f_1(z) \ , \quad \ldots, \quad f_n(z) \ ,$$

where f_i $(0 < i \leq n)$ are arbitrary integer functions definable in Presburger arithmetic, and $f_i(z)$ $(0 < i \leq n)$ occur inside boundary terms. Note that f_i's are introduced during reductions in Section 4.8, and they are constructed using those integer functions defined in Section 4.7.

For each $0 < i \leq n$, we proceed as follows. Without loss of generality, we assume $f_i(z)$ *properly* occurs inside a boundary term t, that is, t has one of the following forms:

$$\mathsf{0}^{\mathsf{w}}_{(f_i(z))} \ , \qquad \mathsf{0}^{\mathsf{p}}_{(f_i(z),p)} \ , \qquad \mathsf{1}^{\mathsf{w}}_{(f_i(z))} \ , \qquad \mathsf{1}^{\mathsf{p}}_{(f_i(z),p)} \ .$$

We do case analysis according to the occurrences of t in $\varphi_{\mathrm{kb}^+}(z)$.

1. t occurs in literals of the forms

   $$t \prec^{\sharp}_n t' \ , \qquad t \preceq^{\sharp}_n t' \ , \qquad t' \prec^{\sharp}_n t \ , \qquad t' \preceq^{\sharp}_n t \ , \qquad t = t' \ ,$$

   where $n > 0$, $\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$ and t' is also a boundary term. By Lemma 4.6, we can remove all such closed literals from $\varphi_{\mathrm{kb}^+}(z)$ and add the corresponding equivalent Presburger formulas to $\varphi_{\mathbb{Z}}(z)$.

2. t occurs in literals of the forms

   $$t \prec^{*}_n t' \ , \qquad t \preceq^{*}_n t' \ , \qquad t' \prec^{*}_n t \ , \qquad t' \preceq^{*}_n t \ , \qquad t = t' \ ,$$

   where $n > 0$, $* \in \{\mathsf{p}, \mathsf{l}\}$ and t' is an ordinary term. Then we replace every occurrence of $f_i(z)$ in $\varphi_{\mathrm{kb}^+}(z)$ by $(t')^{\mathsf{w}}$ and add $f_i(z) = (t')^{\mathsf{w}}$ to $\varphi_Z(z)$. Formally, we set

   $$\begin{aligned} \varphi_{\mathrm{kb}^+}(z) &:= \varphi_{\mathrm{kb}^+}(z)\left[\, f_i(z)/(t')^{\mathsf{w}} \,\right] \ , \\ \varphi_{\mathbb{Z}}(z) &:= \varphi_{\mathbb{Z}}(z) \ \wedge \ f_i(z) = (t')^{\mathsf{w}} \ . \end{aligned}$$

3. t occurs in literals of the forms

   $$t \prec^{\mathsf{w}}_n t' \ , \qquad t \preceq^{\mathsf{w}}_n t' \ , \qquad t' \prec^{\mathsf{w}}_n t \ , \qquad t' \preceq^{\mathsf{w}}_n t \ ,$$

   where $n > 0$ and t' is an ordinary term. In this case, we introduce new boundary terms to

delineate those gap orders. For example, $t \prec^{\mathsf{w}}_{n} t'$ is replaced by

$$\bigvee_{n_1+n_2=n} t \prec^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{((t')^{\mathsf{w}})} \prec^{\mathrm{pl}}_{n_2} t' \ .$$

Similar transformations apply to $t \preceq^{\mathsf{w}}_{n} t'$, $t' \prec^{\mathsf{w}}_{n} t$ and $t' \preceq^{\mathsf{w}}_{n} t$. Then we reduce this case to case 1 and to case 2.

In this way we can remove all $f_i(z)$ from φ_{kb^+}, obtaining

$$(\exists z : \mathbb{Z}) \left[\varphi'_{\mathrm{kb}^+} \ \wedge \ \varphi'_{\mathbb{Z}}(z) \right] ,$$

where φ'_{kb^+} does not contain z. So we can move φ'_{kb^+} out of $\exists z$, obtaining

$$\varphi'_{\mathrm{kb}^+} \ \wedge \ (\exists z : \mathbb{Z})\, \varphi'_{\mathbb{Z}}(z) \ ,$$

as desired. □

Lemma 4.11 (Termination). Both Algorithm 4.1 and Algorithm 4.2 terminate.

Proof. The termination of Algorithm 4.1 is straightforward. The termination of Algorithm 4.2 needs a careful examination. The subtlety comes from step (1). It is clear that each run of step (1) terminates as the decomposition (step (1b)) is done in depth-first manner, which eventually produces a variable whose term occurrences are all of depth 0. However, this step may introduce more existentially quantified variables, gap order literals and equality literals and it may also increase the depth of variables that are not the target of the reduction. Consider in $\mathrm{List}^{\mathbb{Z}}_{\mathrm{kb}^+}$. The formula

$$(\exists x) \left[x \prec^{\mathsf{l}}_{n_1} t \ \wedge \ \mathrm{car}(x) \prec^{\mathsf{l}}_{n-2} t \right] ,$$

where $\mathrm{depth}(x) = 1$ and $\mathrm{depth}(t) = 0$, reduces to (one of choices)

$$(\exists x_1)(\exists x_2) \left[x_1 = \mathrm{car}(t) \ \wedge \ x_2 \prec^{\mathsf{l}}_{n_1} \mathrm{cdr}(t) \ \wedge \ x_1 \prec^{\mathsf{l}}_{n-2} t \right] ,$$

where $\mathrm{depth}(x_1) = \mathrm{depth}(x_2) = 0$ but $\mathrm{depth}(t) = 1$. Here we have one more existentially quantified variable and a new equality literal, and the depth of t is increased by 1. Reduction (4.197) shows a case where the number of gap order literals increases.

A careful examination of the reductions in Section 4.8 reveals that in all transformations the number of open gap order literals is never increasing, which is crucial in this termination proof. We proceed by first establishing five claims.

1. The number of open gap order literals in each reduct is no more than that in the corresponding redex.

2. The total number of occurrences of newly introduced variables in the reduct is no more than the number of occurrences of the decomposed variable in the redex.
3. Without reductions (4.199) and (4.202), the sum of the depths of variables strictly decreases in each run of step (1).
4. Reductions (4.199) and (4.202) are only triggered by step (1) targeting variables occurring in open gap order literals.
5. A run of step (1) targeting a variable x that occurs in an open gap order literal produces four types of outcomes.
   (a) The sum of the depths of variables in the reduct decreases.
   (b) The reduct contains fewer open gap order literals than the redex.
   (c) The reduct contains a newly introduced variable x' that occurs in an open gap order literal and is the target for the next run of step (1).
   (d) The reduct contains a newly introduced variable x' that occurs in an open gap order literal and is the target for the next run of step (2).

Proof of Claim (1). Recall that this statement is included in every lemma regarding equivalent transformation. Here we list the transformation steps and their corresponding justifications.

Depth Reduction	Lemma 4.6
Elimination of Negative Literals	Lemma 4.1
Equality Elimination	Lemma 4.8, Part 1
Term Elimination	Lemma 4.8, Part 2
Delineated Gap Order Completion	Lemma 4.12

Proof of Claim (2). In all tuple reductions in Sections 4.8.8-4.8.15, if $\langle u_1, \ldots, u_k \rangle$ appears in a redex, then each u_i appears at most once in each corresponding reduct. The claim is then proved by induction.

Proof of Claim (3). It can be shown by checking reductions in Sections 4.8.6, 4.8.7, 4.8.14 and 4.8.15 that (i) the depth of a term can only increase in reductions (4.341) and (4.344) which are triggered by reductions (4.199) and (4.202), respectively, or in a substitution due to eliminating x appearing in an open equality literal of the form $x = t$ (Part 1 of the proof of Lemma 4.8); (ii) an open equality literal is only generated in reductions (4.341) and (4.344). As a consequence, reductions (4.199) and (4.202) are the only sources of increasing depths of variables. In each run of step (1), the depths of newly introduced variables are less than the depth of the decomposed variable in the redex. By claim (2), without reductions (4.199) and (4.202), the sum of the depths of variables must be strictly decreasing in each run of step (1).

Proof of Claim (4). A check of the reductions in Section 4.8 justifies this claim.

Proof of Claim (5). There are two main cases.

1. All occurrences of the targeted variable x have depth greater than 0.
   Then step (1) only involves simplification of selector terms (step (1(c)i)) and simplification of integer terms (step (1(c)ii)). The simplification of integer terms clearly does not affect the depths of variables. For the simplification of selector terms, let us consider $L\mathsf{s}_i^\alpha x \star L'\mathsf{s}_{i'}^{\alpha'} x'$ where $\mathrm{type}(x) = \alpha$, $\mathrm{type}(x') = \alpha'$. If $x \equiv x'$, then $\alpha \equiv \alpha'$, and this literal simplifies to $Lx_i \star L'x_{i'}$. If $x \not\equiv x'$, then this literal simplifies to $Lx_i \star L'\mathsf{s}_{i'}^{\alpha'} x'$. In both cases, $\mathrm{depth}(x_i)$ is less than $\mathrm{depth}(x)$, and the depths of other variables do not increase. Therefore, the sum of the depths of variables decreases. The case for $L'\mathsf{s}_{i'}^{\alpha'} x' \star L\mathsf{s}_i^\alpha x$ is similar.

2. $\mathrm{depth}(x) > 0$ and $x \star t$ (or $t \star x$) occurs.
   Then step (1) also involves reductions of gap order literals (step (1(c)iii)). If neither reduction (4.199) nor reduction (4.202) is used, then x and t are separated by boundary terms, and hence at least one open gap order literal is removed with no new one generated. If either reduction (4.199) or reduction (4.202) is used, then $\star$ must be $\prec_n^{\mathsf{l}}$ or $\preceq_n^{\mathsf{l}}$. Let us consider $x \prec_n^{\mathsf{l}} t$ (the cases for $t \prec_n^{\mathsf{l}} x$, $x \preceq_n^{\mathsf{l}} t$ and $t \preceq_n^{\mathsf{l}} x$ are similar). Reduction (4.199) triggers reduction (4.341), which produces a reduct containing the following conjuncts

$$x_1 = \mathsf{s}_1^\alpha t \ , \quad \ldots \ , \quad x_{i-1} = \mathsf{s}_{i-1}^\alpha t \ , \quad x_i \prec_{n'}^\sharp \mathsf{s}_i^\alpha t \ ,$$

   where $\prec^\sharp \in \{\mathsf{w}, \mathsf{p}, \mathsf{l}\}$ and $n' \leq n$. All newly introduced variables $x_1, \ldots, x_{i-1}$ will be removed by substitution directly. If $\mathrm{depth}(x_i) > 0$ in the reduct, then x_i becomes the target in the next run of step (1). Otherwise, step (2) is activated with x_i being the elimination target.

Now suppose that Algorithm 4.2 does not terminate. The run must visit step (1) infinitely many times, because only step (1) can generate more existentially quantified variables. We claim that either reduction (4.199) or reduction (4.202) runs infinitely often. Suppose otherwise, then from some point on, there are no runs of reductions (4.199) and (4.202). Then by claim (3), a run of step (1) always reduces the sum of the depths of variables. Eventually the depths of all variables are 0 and step (1) is disabled and the run of Algorithm 4.2 stays in step (2) forever, a contradiction.

Now we assume that either reduction (4.199) or reduction (4.202) occurs infinitely often. It follows that there must be infinitely many runs of step (1) targeting variables occurring in open gap order literals, because by claim (4) only such runs can trigger reduction (4.199) or reduction (4.202). However, it is not going to be the case even if there are infinitely many such runs of step (1). We analyze all types of outcomes of step (1) targeting variables occurring in open gap order literals. Outcome (5b) clearly does not happen infinitely many times because at least one open gap order literal is removed each time. Outcome (5d) cannot happen infinitely often either, because (thanks

to Lemma 4.12) each occurrence triggers step (2) that reduces the number of open gap order literals at least by one. Thanks to variable selection (step (1a)) in depth-first manner, outcome (5c) cannot happen continuously infinitely many times as the depths of variables are finite. If outcome (5c) happens, then the next outcome is either (5c) again or (5d). So if outcome (5c) happens infinitely many times, then so does outcome (5d), a contradiction. The only possibility left is that outcome (5a) happens infinitely often. And in fact it will be continuously happening from some point on. This means that from that point on the sum of the depths of variables strictly decreases, and hence eventually step (1) is disabled forever, contradicting our initial assumption.

We have exhausted all cases and all of them lead to contradictions. Hence Algorithm 4.2 terminates. □

4.7 Definitions

In this section, we show that the integer functions and predicates used in the proofs in Section 4.6 and in the reductions in Section 4.8 can be defined in Presburger arithmetic.

Definition of Integer Predicates

1. $\widehat{\mathrm{CNT}}^{\alpha}_{k,n}(x)$ ($k > 0, n \geq 0$): there are *exactly* $n+1$ different α-terms of length x in TA with $|\mathcal{A}| = k$.

$$\widehat{\mathrm{CNT}}^{\alpha}_{k,n}(x) \quad \stackrel{\text{def}}{=\!=} \quad \mathrm{CNT}^{\alpha}_{k,n}(x) \ \wedge \ \neg\, \mathrm{CNT}^{\alpha}_{k,n+1}(x) \ .$$

   $\widehat{\mathrm{CNT}}_{k,n}(x)$ is similarly defined with α-terms replaced by TA-terms.

$$\widehat{\mathrm{CNT}}_{k,n}(x) \quad \stackrel{\text{def}}{=\!=} \quad \mathrm{CNT}_{k,n}(x) \ \wedge \ \neg\, \mathrm{CNT}_{k,n+1}(x) \ .$$

2. $\mathrm{IsTW}^k(n)$: n is the weight of a k-tuple.

$$\mathrm{IsTW}^k(n) \quad \stackrel{\text{def}}{=\!=} \quad (\exists z_1, \ldots, \exists z_k > 0)\Big(\sum_{i=1}^{k} z_i = n \ \wedge \ \bigwedge_{0<i\leq k} \mathrm{Tree}(z_i) \Big) \ .$$

3. $\mathrm{IsCW}^k(n, m)$: m is the weight of a component in a k-tuple of weight n.

$$\mathrm{IsCW}^k(n,m) \quad \stackrel{\text{def}}{=\!=} \quad \mathrm{IsTW}^k(n) \ \wedge \ \Big((k > 1 \ \wedge \ \mathrm{IsTW}^{k-1}(n-m)) \ \vee \ (k = 1 \ \wedge \ m = n) \Big) \ .$$

4. $\mathrm{IsTW}^k(n,p)$: n is the weight of a k-tuple whose first component is of type α_p.

$$\mathrm{IsTW}^k(n,p) \quad \stackrel{\mathsf{def}}{==} \quad (\exists z_1,\ldots,\exists z_k>0)\Big(\sum_{i=1}^{k} z_i = n \ \wedge \bigwedge_{0<i\le k} \mathrm{Tree}(z_i) \ \wedge \ \mathrm{Tree}^{\alpha_p}(z_1)\Big) \ .$$

5. $\mathrm{IsCW}^k(n,m,p)$: n is the weight of a k-tuple whose first component has weight m and type α_p.

$$\mathrm{IsCW}^k(n,m,p) \quad \stackrel{\mathsf{def}}{==} \quad \mathrm{IsCW}^k(n,m) \ \wedge \ \mathrm{Tree}^{(\alpha_p)}(m) \ .$$

6. $\mathrm{WD}_\Omega(\Delta) : \Omega(\Delta)$ is a well-defined boundary term or boundary tuple.

$$\begin{array}{rcl@{\qquad}rcl}
\mathrm{WD}_{0^{\mathsf{w}}}(m) & \stackrel{\mathsf{def}}{==} & \mathrm{Tree}(m) \ , & \mathrm{WD}_{1^{\mathsf{w}}}(m) & \stackrel{\mathsf{def}}{==} & \mathrm{Tree}(m) \ , \\
\mathrm{WD}_{0^{\mathsf{p}}}(m,p) & \stackrel{\mathsf{def}}{==} & \mathrm{Tree}^{(\alpha_p)}(m) \ , & \mathrm{WD}_{1^{\mathsf{p}}}(m,p) & \stackrel{\mathsf{def}}{==} & \mathrm{Tree}^{(\alpha_p)}(m) \ , \\
\mathrm{WD}_{\overline{0}^{k;\mathsf{kb}}}(n) & \stackrel{\mathsf{def}}{==} & \mathrm{IsTW}^k(n) \ , & \mathrm{WD}_{\overline{1}^{k;\mathsf{kb}}}(n) & \stackrel{\mathsf{def}}{==} & \mathrm{IsTW}^k(n) \ , \\
\mathrm{WD}_{\overline{0}^{k;\mathsf{w}}}(n,m) & \stackrel{\mathsf{def}}{==} & \mathrm{IsCW}^k(n,m) \ , & \mathrm{WD}_{\overline{1}^{k;\mathsf{w}}}(n,m) & \stackrel{\mathsf{def}}{==} & \mathrm{IsCW}^k(n,m) \ , \\
\mathrm{WD}_{\overline{0}^{k;\mathsf{p}}}(n,m,p) & \stackrel{\mathsf{def}}{==} & \mathrm{IsCW}^k(n,m,p) \ , & \mathrm{WD}_{\overline{1}^{k;\mathsf{p}}}(n,m,p) & \stackrel{\mathsf{def}}{==} & \mathrm{IsCW}^k(n,m,p) \ .
\end{array}$$

Definition of Integer Functions

1. $x = \mathrm{MinCW}^k(n,r,r')$: x is the smallest integer in (r,r') which is the weight of a component in a k-tuple of weight n; $x = r$ if there is no such integer.

$$\Big[\Big(\neg\,\mathrm{IsTW}^k(n) \ \vee \ \forall z(r<z<r' \ \rightarrow \ \neg\,\mathrm{IsCW}^k(n,z))\Big) \ \wedge \ x=r\Big] \quad \vee$$
$$\Big[\,\mathrm{IsCW}^k(n,x) \ \wedge \ r<x<r' \ \wedge \ \forall z(r<z<x \ \rightarrow \ \neg\,\mathrm{IsCW}^k(n,z))\,\Big] \ . \quad (4.56)$$

2. $x = \mathrm{MaxCW}^k(n,r,r')$: x is the largest integer in (r,r') which is the weight of a component in a k-tuple of weight n; $x = r$ if there is no such integer.

$$\Big[\Big(\neg\,\mathrm{IsTW}^k(n) \ \vee \ \forall z(r<z<r' \ \rightarrow \ \neg\,\mathrm{IsCW}^k(n,z))\Big) \ \wedge \ x=r\Big] \quad \vee$$
$$\Big[\,\mathrm{IsCW}^k(n,x) \ \wedge \ r<x<r' \ \wedge \ \forall z(x<z<r' \ \rightarrow \ \neg\,\mathrm{IsCW}^k(n,z))\,\Big] \ . \quad (4.57)$$

3. $x = \mathrm{CW}^k_i(n,r,r')$ $(i>0)$: x is the i^{th} smallest integer in (r,r') which is the weight of a component in a k-tuple of weight n; $x = \mathrm{MaxCW}^k(n,r,r')$ if no such integer exists. We give the definition by induction on i.

Case $i = 1$.

$$x = \mathrm{MinCW}^k(n, r, r').$$

Case $i > 1$.

$$\begin{aligned}\Big[\, x > \mathrm{CW}^k_{i-1}(n,r,r') \;\wedge\; r < x < r' \;\wedge\; \mathrm{IsCW}^k(n,x) \;\wedge\; \\ \forall z \Big(\mathrm{CW}^k_{i-1}(n,r,r') < z < x \;\rightarrow\; \neg\, \mathrm{IsCW}^k(n,z)\Big) \,\Big] \quad \vee \\ \Big[\, x = \mathrm{CW}^k_{i-1}(n,r,r') \;\wedge\; \forall z \Big((z > \mathrm{CW}^k_{i-1}(n,r,r') \;\wedge\; r < z < r') \;\rightarrow\; \neg\, \mathrm{IsCW}^k(n,z)\Big) \,\Big] \,. \quad (4.58)\end{aligned}$$

With no risk of confusion, we make the following abbreviations.

$$\begin{array}{lll} \mathrm{MinCW}^k(n) & \text{abbr.} & \mathrm{MinCW}^k(n, 0, n+1) \;, \\ \mathrm{MaxCW}^k(n) & \text{abbr.} & \mathrm{MaxCW}^k(n, 0, n+1) \;, \\ \mathrm{CW}^k_i(n) & \text{abbr.} & \mathrm{CW}^k_i(n, 0, n+1) \;. \end{array}$$

4. $x = \mathrm{CWS}^k_i(n)$ $(0 < i \leq k)$: x is the weight of the i^{th} component of the smallest k-tuple (w.r.t. $\prec^{k;\mathrm{kb}}$) of weight n; $x = 0$ if no such weight exists. We define it inductively as follows.
Case $i = 1$.

$$x = \mathrm{MinCW}^k(n) \;.$$

Case $1 < i \leq k$.

$$\Big(\, \neg\, \mathrm{IsTW}^k(n) \;\rightarrow\; x = 0 \,\Big) \;\wedge\; \Big(\, \mathrm{IsTW}^k(n) \;\rightarrow\; x = \mathrm{MinCW}^{k-i+1}\Big(n - \sum_{j=1}^{i-1} \mathrm{CWS}^k_j(n)\Big)\Big) \;.$$

5. $x = \mathrm{CWL}^k_i(n)$ $(0 < i \leq k)$: x is the weight of the i^{th} component of the largest k-tuple (w.r.t. $\prec^{k;\mathrm{kb}}$) of weight n; $x = 0$ if no such weight exists. We define it inductively as follows.
Case $i = 1$.

$$x = \mathrm{MaxCW}^k(n) \;.$$

Case $1 < i \leq k$.

$$\Big(\, \neg\, \mathrm{IsTW}^k(n) \;\rightarrow\; x = 0 \,\Big) \;\wedge\; \Big(\, \mathrm{IsTW}^k(n) \;\rightarrow\; x = \mathrm{MaxCW}^{k-i+1}\Big(n - \sum_{j=1}^{i-1} \mathrm{CWS}^k_j(n)\Big)\Big) \;.$$

4.8 Reductions

In this section we present all reductions that can occur in Algorithm 4.2. The following outlines the contents in each subsection.

4.8.1	Closed Term Equalities	4.8.8	Closed Tuple Equalities
——	(No Half-open Term Equalities)	4.8.9	Half-open Tuple Equalities
4.8.2	Closed Stretchable Gap Orders	4.8.10	Closed Stretchable Tuple Gap Orders
4.8.3	Closed Rigid Gap Orders	4.8.11	Closed Rigid Tuple Gap Orders
4.8.4	Half-open Stretchable Gap Orders	4.8.12	Half-open Stretchable Tuple Gap Orders
4.8.5	Half-open Rigid Gap Orders	4.8.13	Half-open Rigid Tuple Gap Orders
4.8.6	Open Stretchable Gap Orders	4.8.14	Open Stretchable Tuple Gap Orders
4.8.7	Open Rigid Gap Orders	4.8.15	Open Rigid Tuple Gap Orders

We assume that all gap order counts in redexes are positive numbers.

4.8.1 Reduction of Closed Equalities

$$0^{\mathsf{w}}_{(m)} = 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \tag{4.59}$$

$$0^{\mathsf{w}}_{(m)} = 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \bigwedge_{p'<i\leq|\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_i)}(m) \tag{4.60}$$

$$0^{\mathsf{w}}_{(m)} = 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \ \wedge \ \neg\, \mathrm{CNT}_1(m) \tag{4.61}$$

$$0^{\mathsf{w}}_{(m)} = 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \bigwedge_{p'<i\leq|\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_i)}(m) \ \wedge \ \neg\, \mathrm{CNT}_1^{(\alpha_{p'})}(m) \tag{4.62}$$

$$0^{\mathsf{p}}_{(m,p)} = 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \ p = p' \tag{4.63}$$

$$0^{\mathsf{p}}_{(m,p)} = 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \ \wedge \bigwedge_{0<i<p} \neg\, \mathrm{Tree}^{(\alpha_i)}(m) \ \wedge \ \neg\, \mathrm{CNT}_1^{(\alpha_p)}(m) \tag{4.64}$$

$$0^{\mathsf{p}}_{(m,p)} = 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \ p = p' \ \wedge \ \neg\, \mathrm{CNT}_1^{(\alpha_p)}(m) \tag{4.65}$$

$$1^{\mathsf{w}}_{(m)} = 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \tag{4.66}$$

$$1^{\mathsf{w}}_{(m)} = 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \bigwedge_{0<i<p'} \neg\, \mathrm{Tree}^{(\alpha_i)}(m) \tag{4.67}$$

$$1^{\mathsf{p}}_{(m,p)} = 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \ p = p' \tag{4.68}$$

4.8.2 Reduction of Closed Stretchable Gap Orders

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} \mathsf{0}^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} \mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{pl}}_{n_1} \mathsf{1}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_2} \mathsf{0}^{\mathsf{w}}_{(m')} \tag{4.69}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n} \mathsf{0}^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.70}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n} \mathsf{0}^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.71}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} \mathsf{0}^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} \mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{pl}}_{n_1} \mathsf{1}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_2} \mathsf{0}^{\mathsf{w}}_{(m')} <^{\mathsf{w}}_{n_3} \mathsf{0}^{\mathsf{p}}_{(m',p')} \tag{4.72}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n} \mathsf{0}^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \bigvee_{n_{p'+1}+\cdots+n_{|\Sigma|}=n} \bigwedge_{p'<i\leq|\Sigma|} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m)\right) \tag{4.73}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n} \mathsf{0}^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.74}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} \mathsf{1}^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} \mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{pl}}_{n_1} \mathsf{1}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_2} \mathsf{0}^{\mathsf{w}}_{(m')} <^{\mathsf{pl}}_{n_3} \mathsf{1}^{\mathsf{w}}_{(m')} \tag{4.75}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n} \mathsf{1}^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \wedge \bigvee\limits_{\substack{p,p'\in\{1,\ldots,|\Sigma|\} \\ p\neq p'}} \left(\mathrm{Tree}^{(\alpha_p)}(m) \ \wedge \ \mathrm{Tree}^{(\alpha_{p'})}(m)\right) \\ \qquad \wedge \bigvee\limits_{n_1+\cdots+n_{|\Sigma|}=n+1} \bigwedge\limits_{0<i\leq|\Sigma|} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m)\right) \end{array} \right) \tag{4.76}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n} \mathsf{1}^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \wedge \bigwedge\limits_{\substack{p,p'\in\{1,\ldots,|\Sigma|\} \\ p\neq p'}} \left(\mathrm{Tree}^{(\alpha_p)}(m) \ \rightarrow \ \neg\, \mathrm{Tree}^{(\alpha_{p'})}(m)\right) \\ \qquad \wedge \bigvee\limits_{0<i\leq|\Sigma|} \mathrm{CNT}^{(\alpha_i)}_{n}(m) \end{array} \right) \tag{4.77}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} \mathsf{1}^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_2>0}} \mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{pl}}_{n_1} \mathsf{1}^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_2} \mathsf{0}^{\mathsf{w}}_{(m')} <^{\mathsf{pl}}_{n_3} \mathsf{0}^{\mathsf{w}}_{(m',p')} <^{\mathsf{pl}}_{n_4} \mathsf{1}^{\mathsf{w}}_{(m',p')} \tag{4.78}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n} \mathsf{1}^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \wedge \bigvee\limits_{p'<p\leq|\Sigma|} \mathrm{Tree}^{(\alpha_p)}(m) \\ \qquad \wedge \bigvee\limits_{n_{p'}+\cdots+n_{|\Sigma|}=n+1} \bigwedge\limits_{p'\leq i\leq|\Sigma|} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m)\right) \end{array} \right) \tag{4.79}$$

$$\mathsf{0}^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n} \mathsf{1}^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \bigwedge_{p'<p\leq|\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_p)}(m) \ \wedge \ \mathrm{CNT}^{(\alpha_{p'})}_{n}(m) \tag{4.80}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \tag{4.81}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.82}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.83}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_3>0}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{pl}}_{n_1} 1^{\mathsf{p}}_{(m,p)} <^{\mathsf{pl}}_{n_2} 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_3} 0^{\mathsf{w}}_{(m')} <^{\mathsf{pl}}_{n_4} 0^{\mathsf{p}}_{(m',p')} \tag{4.84}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} m = m' \ \wedge \ p' < p \\ \wedge \\ \displaystyle\bigvee_{n_{p'}+\cdots+n_{p-1}=n} \ \bigwedge_{p' \le i < p} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \tag{4.85}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.86}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_3>0}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{pl}}_{n_1} 1^{\mathsf{p}}_{(m,p)} <^{\mathsf{pl}}_{n_2} 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_3} 0^{\mathsf{w}}_{(m')} <^{\mathsf{pl}}_{n_4} 1^{\mathsf{w}}_{(m')} \tag{4.87}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{c} m = m' \ \wedge \ \displaystyle\bigvee_{0<i<p} \mathrm{Tree}^{(\alpha_i)}(m) \\ \wedge \ \displaystyle\bigvee_{n_1+\cdots+n_p=n+1} \ \bigwedge_{0<i\le p} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \tag{4.88}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \ \wedge \ \bigwedge_{0<i<p} \neg \ \mathrm{Tree}^{(\alpha_i)}(m) \ \wedge \ \mathrm{CNT}^{(\alpha_p)}_{n}(m) \tag{4.89}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{\sum_{i=1}^{5} n_i=n \\ n_3>0}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m,p)} <^{\mathsf{pl}}_{n_2} 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_3} 0^{\mathsf{w}}_{(m')} <^{\mathsf{pl}}_{n_4} 0^{\mathsf{p}}_{(m',p')} <^{\mathsf{l}}_{n_5} 1^{\mathsf{p}}_{(m',p')} \tag{4.90}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} m = m' \ \wedge \ p' < p \\ \wedge \\ \displaystyle\bigvee_{n_{p'}+\cdots+n_p=n+1} \ \bigwedge_{p' \le i \le p} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \tag{4.91}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \ p = p' \ \wedge \ \mathrm{CNT}^{(\alpha_p)}_{n}(m) \tag{4.92}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{c} (n = 1 \ \rightarrow \ m < m') \\ \wedge \\ n > 1 \ \rightarrow \ \bigvee\limits_{0<r<n} (\exists z_1 \ldots \exists z_r) \left(\begin{array}{c} m < z_1 < \cdots < z_r < m' \\ \wedge \\ \bigvee\limits_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_1,\ldots,n_r>0}} \bigwedge\limits_{0<i\leq r} \mathrm{CNT}_{n_i-1}(z_i) \end{array} \right) \end{array} \right) \tag{4.93}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{p}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.94}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{l}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.95}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(m')} \prec^{\mathrm{pl}}_{n_2} 0^{\mathsf{p}}_{(m',p')} \tag{4.96}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{p}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.97}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{l}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.98}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(m')} \prec^{\mathrm{pl}}_{n_2} 1^{\mathsf{w}}_{(m')} \tag{4.99}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.100}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.101}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(m')} \prec^{\mathrm{pl}}_{n_2} 0^{\mathsf{w}}_{(m',p')} \prec^{\mathrm{pl}}_{n_3} 1^{\mathsf{w}}_{(m',p')} \tag{4.102}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.103}$$

$$1^{\mathsf{w}}_{(m)} \prec^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.104}$$

$$1^{\mathsf{p}}_{(m,p)} \prec^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} \prec^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \tag{4.105}$$

$$1^{\mathsf{p}}_{(m,p)} \prec^{\mathsf{p}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.106}$$

$$1^{\mathsf{p}}_{(m,p)} \prec^{\mathsf{l}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.107}$$

$$1^{\mathsf{p}}_{(m,p)} \prec^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} \prec^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \prec^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \prec^{\mathsf{w}}_{n_3} 0^{\mathsf{p}}_{(m',p')} \tag{4.108}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} n = 1 \ \rightarrow \ (m = m' \ \wedge \ p' < p) \\ \wedge \\ n > 1 \ \rightarrow \left(\begin{array}{c} m = m' \ \wedge \ p' < p - 1 \\ \wedge \\ \bigvee\limits_{n_{p'+1}+\cdots+n_{p-1}=n-1} \ \bigwedge\limits_{p'<i<p} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \end{array} \right) \tag{4.109}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.110}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} <^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} <^{\mathrm{pl}}_{n_3} 1^{\mathsf{w}}_{(m')} \tag{4.111}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \ \wedge \ \bigvee_{n_1+\cdots+n_{p-1}=n} \ \bigwedge_{0<i<p} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m) \right) \tag{4.112}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.113}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} <^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} <^{\mathrm{pl}}_{n_3} 0^{\mathsf{w}}_{(m',p')} <^{\mathrm{pl}}_{n_4} 1^{\mathsf{w}}_{(m',p')} \tag{4.114}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} m = m' \ \wedge \ p' < p \\ \wedge \\ \bigvee\limits_{n_{p'}+\cdots+n_{p-1}=n} \ \bigwedge\limits_{p'\leq i<p} \left(n_i > 0 \ \rightarrow \ \mathrm{CNT}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \tag{4.115}$$

$$1^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.116}$$

4.8.3 Reduction of Closed Rigid Gap Orders

$$0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \tag{4.117}$$

$$0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{p}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.118}$$

$$0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{l}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.119}$$

$$0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \leqq^{\mathsf{w}}_{n_3} 0^{\mathsf{p}}_{(m',p')} \tag{4.120}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{p}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \bigvee_{n_{p'+1}+\cdots+n_{|\Sigma|}=n} \bigwedge_{p'<i\leq|\Sigma|} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m)\right) \tag{4.121}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{l}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.122}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} 0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \trianglelefteq^{\mathsf{pl}}_{n_3} 1^{\mathsf{w}}_{(m')} \tag{4.123}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \wedge \bigvee\limits_{\substack{p,p'\in\{1,\ldots,|\Sigma|\} \\ p\neq p'}} \left(\mathrm{Tree}^{(\alpha_p)}(m) \ \wedge \ \mathrm{Tree}^{(\alpha_{p'})}(m)\right) \\ \qquad \wedge \bigvee\limits_{n_1+\cdots+n_{|\Sigma|}=n+1} \bigwedge\limits_{0<i\leq|\Sigma|} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m)\right) \end{array} \right) \tag{4.124}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \wedge \bigwedge\limits_{\substack{p,p'\in\{1,\ldots,|\Sigma|\} \\ p\neq p'}} \left(\mathrm{Tree}^{(\alpha_p)}(m) \ \rightarrow \ \neg\, \mathrm{Tree}^{(\alpha_{p'})}(m)\right) \\ \qquad \wedge \bigvee\limits_{0<i\leq|\Sigma|} \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n}(m) \end{array} \right) \tag{4.125}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_2>0}} 0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \trianglelefteq^{\mathsf{pl}}_{n_3} 0^{\mathsf{w}}_{(m',p')} \trianglelefteq^{\mathsf{pl}}_{n_4} 1^{\mathsf{w}}_{(m',p')} \tag{4.126}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \wedge \bigvee\limits_{p'<p\leq|\Sigma|} \mathrm{Tree}^{(\alpha_p)}(m) \\ \qquad \wedge \bigvee\limits_{n_{p'}+\cdots+n_{|\Sigma|}=n+1} \bigwedge\limits_{p'\leq i\leq|\Sigma|} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m)\right) \end{array} \right) \tag{4.127}$$

$$0^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \bigwedge_{p'<p\leq|\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_p)}(m) \ \wedge \ \widehat{\mathrm{CNT}}^{(\alpha_{p'})}_{n}(m) \tag{4.128}$$

$$0^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} 0^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \tag{4.129}$$

$$0^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{p}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.130}$$

$$0^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{l}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.131}$$

$$0^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_3>0}} 0^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{pl}}_{n_1} 1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{pl}}_{n_2} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_3} 0^{\mathsf{w}}_{(m')} \trianglelefteq^{\mathsf{pl}}_{n_4} 0^{\mathsf{p}}_{(m',p')} \tag{4.132}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} m = m' \ \wedge \ p' < p \\ \wedge \\ \bigvee\limits_{n_{p'}+\cdots+n_{p-1}=n} \ \bigwedge\limits_{p' \leq i < p} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \tag{4.133}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.134}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_3>0}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{pl}}_{n_1} 1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{pl}}_{n_2} 1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_3} 0^{\mathsf{w}}_{(m')} \leqq^{\mathsf{pl}}_{n_4} 1^{\mathsf{w}}_{(m')} \tag{4.135}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \wedge \ \bigvee\limits_{0<i<p} \mathrm{Tree}^{(\alpha_i)}(m) \\ \qquad \wedge \ \bigvee\limits_{n_1+\cdots+n_p=n+1} \ \bigwedge\limits_{0<i\leq p} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \tag{4.136}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \ \wedge \ \bigwedge_{0<i<p} \neg \, \mathrm{Tree}^{(\alpha_i)}(m) \ \wedge \ \widehat{\mathrm{CNT}}^{(\alpha_p)}_{n}(m). \tag{4.137}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{\sum_{i=1}^{5} n_i = n \\ n_3>0}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{pl}}_{n_2} 1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_3} 0^{\mathsf{w}}_{(m')} \leqq^{\mathsf{pl}}_{n_4} 0^{\mathsf{p}}_{(m',p')} \leqq^{\mathsf{l}}_{n_5} 1^{\mathsf{p}}_{(m',p')} \tag{4.138}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} m = m' \ \wedge \ p' < p \\ \wedge \\ \bigvee\limits_{n_{p'}+\cdots+n_{p}=n+1} \ \bigwedge\limits_{p' \leq i \leq p} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \tag{4.139}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad m = m' \ \wedge \ p = p' \ \wedge \ \widehat{\mathrm{CNT}}^{(\alpha_p)}_{n}(m) \tag{4.140}$$

$$1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \left(\begin{array}{l} n = 1 \ \rightarrow \ \left(m < m' \ \wedge \ \forall z \left(m < z < m' \ \rightarrow \ \neg \, \mathrm{Tree}(z) \right) \right) \\ \qquad \wedge \\ n > 1 \ \rightarrow \ \bigvee\limits_{0<r<n} (\exists z_1 \ldots \exists z_r) \left(\begin{array}{c} m < z_1 < \cdots < z_r < m' \\ \wedge \\ \neg \, (\exists z_1 \ldots \exists z_{r+1}) \\ \left(m < z_1 < \cdots < z_{r+1} < m' \right) \\ \wedge \\ \bigvee\limits_{\substack{\sum_{i=1}^{r} n_i = n-1 \\ n_1,\ldots,n_r>0}} \ \bigwedge\limits_{0<i\leq r} \widehat{\mathrm{CNT}}_{n_i-1}(z_i) \end{array} \right) \end{array} \right) \tag{4.141}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{p}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.142}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{l}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.143}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(m')} \trianglelefteq^{\mathsf{pl}}_{n_2} 0^{\mathsf{p}}_{(m',p')} \tag{4.144}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{p}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.145}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{l}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.146}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(m')} \trianglelefteq^{\mathsf{pl}}_{n_2} 1^{\mathsf{w}}_{(m')} \tag{4.147}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.148}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.149}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(m')} \trianglelefteq^{\mathsf{pl}}_{n_2} 0^{\mathsf{w}}_{(m',p')} \trianglelefteq^{\mathsf{pl}}_{n_3} 1^{\mathsf{w}}_{(m',p')} \tag{4.150}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.151}$$

$$1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.152}$$

$$1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \tag{4.153}$$

$$1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{p}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.154}$$

$$1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{l}}_{n} 0^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.155}$$

$$1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \trianglelefteq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \trianglelefteq^{\mathsf{w}}_{n_3} 0^{\mathsf{p}}_{(m',p')} \tag{4.156}$$

$$1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{p}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} n = 1 \ \rightarrow \ \left(m = m' \ \wedge \ p' < p \ \wedge \ \bigwedge\limits_{p'<i<p} \neg \, \mathrm{Tree}^{\alpha_i}(m) \right) \\ \wedge \\ n > 1 \ \rightarrow \left(\begin{array}{c} m = m' \ \wedge \ p' < p-1 \ \wedge \\ \bigvee\limits_{n_{p'+1}+\cdots+n_{p-1}=n-1} \ \bigwedge\limits_{p'<i<p} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m) \right) \end{array} \right) \end{array} \right) \tag{4.157}$$

$$1^{\mathsf{p}}_{(m,p)} \trianglelefteq^{\mathsf{l}}_{n} 0^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.158}$$

$$1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \leqq^{\mathsf{pl}}_{n_3} 1^{\mathsf{w}}_{(m')} \tag{4.159}$$

$$1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad m = m' \ \wedge \bigvee_{n_1+\cdots+n_{p-1}=n} \bigwedge_{0<i<p} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m)\right) \tag{4.160}$$

$$1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m')} \quad \mapsto \quad \text{false} \tag{4.161}$$

$$1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_2>0}} 1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{pl}}_{n_1} 1^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m')} \leqq^{\mathsf{pl}}_{n_3} 0^{\mathsf{w}}_{(m',p')} \leqq^{\mathsf{pl}}_{n_4} 1^{\mathsf{w}}_{(m',p')} \tag{4.162}$$

$$1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \left(\begin{array}{c} m = m' \ \wedge \ p' < p \\ \wedge \\ \bigvee_{n_{p'}+\cdots+n_{p-1}=n} \bigwedge_{p'\leq i<p} \left(n_i > 0 \ \rightarrow \ \widehat{\mathrm{CNT}}^{(\alpha_i)}_{n_i-1}(m)\right) \end{array} \right) \tag{4.163}$$

$$1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m',p')} \quad \mapsto \quad \text{false} \tag{4.164}$$

4.8.4 Reduction of Half Open Stretchable Gap Orders

Let $u = \alpha_q(u_1, \ldots, u_k)$.

$$0^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} <^{\mathsf{pl}}_{n_2} u \tag{4.165}$$

$$0^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m,q)} <^{\mathsf{l}}_{n_2} u \tag{4.166}$$

$$0^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n} u \quad \mapsto \quad \mathrm{Is}_{\alpha_q}(u) \ \wedge \ m = u^{\mathsf{w}} \ \wedge \bigwedge_{q<i\leq|\Sigma|} \neg \ \mathrm{Tree}^{(\alpha_i)}(u^{\mathsf{w}}) \ \wedge \ \overline{0}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} <^{k;\mathrm{kb}}_{n} \langle u_1, \ldots, u_k \rangle \tag{4.167}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} <^{\mathsf{pl}}_{n_2} u \tag{4.168}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m,q)} <^{\mathsf{l}}_{n_2} u \tag{4.169}$$

$$0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n} u \quad \mapsto \quad p = q \ \wedge \ \mathrm{Is}_{\alpha_q}(u) \ \wedge \ m = u^{\mathsf{w}} \ \wedge \ \overline{0}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} <^{k;\mathrm{kb}}_{n} \langle u_1, \ldots, u_k \rangle \tag{4.170}$$

$$1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} <^{\mathsf{pl}}_{n_2} u \tag{4.171}$$

$$1^{\mathsf{p}}_{(m,p)} \prec^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{p}}_{(m,p)} \prec^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathrm{pl}}_{n_2} u \tag{4.172}$$

$$u \prec^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \prec^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathsf{w}}_{n_2} 1^{\mathsf{w}}_{(m)} \tag{4.173}$$

$$u \prec^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \prec^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m,q)} \prec^{\mathsf{p}}_{n_2} 1^{\mathsf{w}}_{(m)} \tag{4.174}$$

$$u \prec^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m)} \quad \mapsto \quad \mathrm{Is}_{\alpha_q}(u) \wedge m = u^{\mathsf{w}} \wedge \bigwedge_{0<i<q} \neg\, \mathrm{Tree}^{(\alpha_i)}(u^{\mathsf{w}}) \wedge \langle u_1, \ldots, u_k \rangle \prec^{k;\mathrm{kb}}_{n} \overline{1}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} \tag{4.175}$$

$$u \prec^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \prec^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathsf{w}}_{n_2} 1^{\mathsf{p}}_{(m,p)} \tag{4.176}$$

$$u \prec^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \prec^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m,q)} \prec^{\mathsf{p}}_{n_2} 1^{\mathsf{p}}_{(m,p)} \tag{4.177}$$

$$u \prec^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad p = q \wedge \mathrm{Is}_{\alpha_q}(u) \wedge m = u^{\mathsf{w}} \wedge \langle u_1, \ldots, u_k \rangle \prec^{k;\mathrm{kb}}_{n} \overline{1}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} \tag{4.178}$$

$$u \prec^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \prec^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m)} \tag{4.179}$$

$$u \prec^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \prec^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \prec^{\mathsf{w}}_{n_2} 0^{\mathsf{p}}_{(m,p)} \tag{4.180}$$

4.8.5 Reduction of Half Open Rigid Gap Orders

Let $u = \alpha_q(u_1, \ldots, u_k)$.

$$0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} \leqq^{\mathrm{pl}}_{n_2} u \tag{4.181}$$

$$0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{p}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m,q)} \leqq^{\mathsf{l}}_{n_2} u \tag{4.182}$$

$$0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{l}}_{n} u \quad \mapsto \quad \mathrm{Is}_{\alpha_q}(u) \wedge m = u^{\mathsf{w}} \wedge \bigwedge_{q<i\leq|\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_i)}(u^{\mathsf{w}}) \wedge \overline{0}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} \leqq^{k;\mathrm{kb}}_{n} \langle u_1, \ldots, u_k \rangle \tag{4.183}$$

$$0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} \leqq^{\mathrm{pl}}_{n_2} u \tag{4.184}$$

$$0^{\mathsf{p}}_{(m,p)} \mathrel{\leqq}^{\mathsf{p}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 0^{\mathsf{p}}_{(m,p)} \mathrel{\leqq}^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m,q)} \mathrel{\leqq}^{\mathsf{l}}_{n_2} u \tag{4.185}$$

$$0^{\mathsf{p}}_{(m,p)} \mathrel{\leqq}^{\mathsf{l}}_{n} u \quad \mapsto \quad p = q \;\wedge\; \mathrm{Is}_{\alpha_q}(u) \;\wedge\; m = u^{\mathsf{w}} \;\wedge\; \overline{0}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} \mathrel{\leqq}^{k;\mathrm{kb}}_{n} \langle u_1, \ldots, u_k \rangle \tag{4.186}$$

$$1^{\mathsf{w}}_{(m)} \mathrel{\leqq}^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{w}}_{(m)} \mathrel{\leqq}^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} \mathrel{\leqq}^{\mathrm{pl}}_{n_2} u \tag{4.187}$$

$$1^{\mathsf{p}}_{(m,p)} \mathrel{\leqq}^{\mathsf{w}}_{n} u \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} 1^{\mathsf{p}}_{(m,p)} \mathrel{\leqq}^{\mathsf{w}}_{n_1} 0^{\mathsf{w}}_{(u^{\mathsf{w}})} \mathrel{\leqq}^{\mathrm{pl}}_{n_2} u \tag{4.188}$$

$$u \mathrel{\leqq}^{\mathsf{w}}_{n} 1^{\mathsf{w}}_{(m)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \mathrel{\leqq}^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \mathrel{\leqq}^{\mathsf{w}}_{n_2} 1^{\mathsf{w}}_{(m)} \tag{4.189}$$

$$u \mathrel{\leqq}^{\mathsf{p}}_{n} 1^{\mathsf{w}}_{(m)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \mathrel{\leqq}^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m,q)} \mathrel{\leqq}^{\mathsf{p}}_{n_2} 1^{\mathsf{w}}_{(m)} \tag{4.190}$$

$$u \mathrel{\leqq}^{\mathsf{l}}_{n} 1^{\mathsf{w}}_{(m)} \quad \mapsto \quad \mathrm{Is}_{\alpha_q}(u) \;\wedge\; m = u^{\mathsf{w}} \;\wedge\; \bigwedge_{0<i<q} \neg\, \mathrm{Tree}^{(\alpha_i)}(u^{\mathsf{w}}) \;\wedge\; \langle u_1, \ldots, u_k \rangle \mathrel{\leqq}^{k;\mathrm{kb}}_{n} \overline{1}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} \tag{4.191}$$

$$u \mathrel{\leqq}^{\mathsf{w}}_{n} 1^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \mathrel{\leqq}^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \mathrel{\leqq}^{\mathsf{w}}_{n_2} 1^{\mathsf{p}}_{(m,p)} \tag{4.192}$$

$$u \mathrel{\leqq}^{\mathsf{p}}_{n} 1^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \mathrel{\leqq}^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m,q)} \mathrel{\leqq}^{\mathsf{p}}_{n_2} 1^{\mathsf{p}}_{(m,p)} \tag{4.193}$$

$$u \mathrel{\leqq}^{\mathsf{l}}_{n} 1^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad p = q \;\wedge\; \mathrm{Is}_{\alpha_q}(u) \;\wedge\; m = u^{\mathsf{w}} \;\wedge\; \langle u_1, \ldots, u_k \rangle \mathrel{\leqq}^{k;\mathrm{kb}}_{n} \overline{1}^{k;\mathrm{kb}}_{(m-\mathsf{w}(\alpha_q))} \tag{4.194}$$

$$u \mathrel{\leqq}^{\mathsf{w}}_{n} 0^{\mathsf{w}}_{(m)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \mathrel{\leqq}^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \mathrel{\leqq}^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(m)} \tag{4.195}$$

$$u \mathrel{\leqq}^{\mathsf{w}}_{n} 0^{\mathsf{p}}_{(m,p)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} u \mathrel{\leqq}^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \mathrel{\leqq}^{\mathsf{w}}_{n_2} 0^{\mathsf{p}}_{(m,p)} \tag{4.196}$$

4.8.6 Reductions of Open Stretchable Gap Orders

$$u \mathrel{\prec}^{\mathsf{w}}_{n} v \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} u \mathrel{\prec}^{\mathrm{pl}}_{n_1} 1^{\mathsf{w}}_{(u^{\mathsf{w}})} \mathrel{\prec}^{\mathsf{w}}_{n_2} 0^{\mathsf{w}}_{(v^{\mathsf{w}})} \mathrel{\prec}^{\mathrm{pl}}_{n_3} v \tag{4.197}$$

$$u \mathrel{\prec}^{\mathsf{p}}_{n} v \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} u \mathrel{\prec}^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(u^{\mathsf{w}},p)} \mathrel{\prec}^{\mathsf{p}}_{n_2} 0^{\mathsf{p}}_{(v^{\mathsf{w}},q)} \mathrel{\prec}^{\mathsf{l}}_{n_3} v \tag{4.198}$$

$$u \prec_n^{\mathsf{l}} v \quad \mapsto \quad \begin{array}{c} \textit{where } \mathrm{type}(u) = \alpha_p \textit{ and } \mathrm{type}(v) = \alpha_q \\ \langle u_1, \ldots, u_k \rangle \prec_n^{k;\mathrm{kb}} \langle v_1, \ldots, v_k \rangle \\ \textit{where } u = \alpha(u_1, \ldots, u_k) \textit{ and } v = \alpha(v_1, \ldots, v_k) \end{array} \tag{4.199}$$

4.8.7 Reductions of Open Rigid Gap Orders

$$u \leqq_n^{\mathsf{w}} v \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} u \leqq_{n_1}^{\mathrm{pl}} 1_{(u^{\mathsf{w}})}^{\mathsf{w}} \leqq_{n_2}^{\mathsf{w}} 0_{(v^{\mathsf{w}})}^{\mathsf{w}} \leqq_{n_3}^{\mathrm{pl}} v \tag{4.200}$$

$$u \leqq_n^{\mathsf{p}} v \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} u \leqq_{n_1}^{\mathsf{l}} 1_{(u^{\mathsf{w}},p)}^{\mathsf{p}} \leqq_{n_2}^{\mathsf{p}} 0_{(v^{\mathsf{w}},q)}^{\mathsf{p}} \leqq_{n_3}^{\mathsf{l}} v \tag{4.201}$$

$$u \leqq_n^{\mathsf{l}} v \quad \mapsto \quad \begin{array}{c} \textit{where } \mathrm{type}(u) = \alpha_p \textit{ and } \mathrm{type}(v) = \alpha_q \\ \langle u_1, \ldots, u_k \rangle \leqq_n^{k;\mathrm{kb}} \langle v_1, \ldots, v_k \rangle \\ \textit{where } u = \alpha(u_1, \ldots, u_k) \textit{ and } v = \alpha(v_1, \ldots, v_k) \end{array} \tag{4.202}$$

4.8.8 Reduction of Closed Tuple Equalities

$$0_{(sum,m)}^{k;\mathsf{w}} = 0_{(sum',m')}^{k;\mathsf{w}} \quad \mapsto \quad sum = sum' \wedge m = m' \tag{4.203}$$

$$0_{(sum,m)}^{k;\mathsf{w}} = 0_{(sum',m',p')}^{k;\mathsf{p}} \quad \mapsto \quad sum = sum' \wedge m = m' \wedge \bigwedge_{p'<i\leq|\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_i)}(m) \tag{4.204}$$

$$0_{(sum,m)}^{k;\mathsf{w}} = 1_{(sum',m')}^{k;\mathsf{w}} \quad \mapsto \quad \left(\begin{array}{r} sum = sum' \wedge m = m' \wedge \neg\, \mathrm{CNT}_1(m) \\ \wedge\ 0_{(sum-m)}^{k-1;\mathrm{kb}} = 1_{(sum'-m')}^{k-1;\mathrm{kb}} \end{array} \right) \tag{4.205}$$

$$0_{(sum,m)}^{k;\mathsf{w}} = 1_{(sum',m',p')}^{k;\mathsf{p}} \quad \mapsto \quad \left(\begin{array}{r} sum = sum' \wedge m = m' \wedge \bigwedge_{p'<i\leq|\Sigma|} \neg\, \mathrm{Tree}^{(\alpha_i)}(m) \\ \wedge\ \neg\, \mathrm{CNT}_1^{(\alpha_{p'})}(m) \wedge 0_{(sum-m)}^{k-1;\mathrm{kb}} = 1_{(sum'-m')}^{k-1;\mathrm{kb}} \end{array} \right) \tag{4.206}$$

$$0_{(sum,m,p)}^{k;\mathsf{p}} = 0_{(sum',m',p')}^{k;\mathsf{p}} \quad \mapsto \quad sum = sum' \wedge m = m' \wedge p = p' \tag{4.207}$$

$$0_{(sum,m,p)}^{k;\mathsf{p}} = 1_{(sum',m')}^{k;\mathsf{w}} \quad \mapsto \quad \left(\begin{array}{r} sum = sum' \wedge m = m' \wedge \bigwedge_{0<i<p} \neg\, \mathrm{Tree}^{(\alpha_i)}(m) \\ \wedge\ \neg\, \mathrm{CNT}_1^{(\alpha_p)}(m) \wedge 0_{(sum-m)}^{k-1;\mathrm{kb}} = 1_{(sum'-m')}^{k-1;\mathrm{kb}} \end{array} \right) \tag{4.208}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} = \overline{1}^{k;\mathsf{p}}_{(sum',m',p')} \quad \mapsto \quad \left(\begin{array}{l} sum = sum' \ \wedge \ m = m' \ \wedge \ p = p' \\ \qquad \wedge \ \neg \, \mathrm{CNT}_1^{(\alpha_p)}(m) \ \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} = \overline{1}^{k-1;\mathsf{kb}}_{(sum'-m')} \end{array} \right) \tag{4.209}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} = \overline{1}^{k;\mathsf{w}}_{(sum',m')} \quad \mapsto \quad sum = sum' \ \wedge \ m = m' \tag{4.210}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} = \overline{1}^{k;\mathsf{p}}_{(sum',m',p')} \quad \mapsto \quad sum = sum' \ \wedge \ m = m' \ \wedge \bigwedge_{0<i<p'} \neg \, \mathrm{Tree}^{(\alpha_i)}(m) \tag{4.211}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} = \overline{1}^{k;\mathsf{p}}_{(sum',m',p')} \quad \mapsto \quad sum = sum' \ \wedge \ m = m' \ \wedge \ p = p' \tag{4.212}$$

4.8.9 Reduction of Half Open Tuple Equalities.

$$\overline{0}^{k;\mathsf{kb}}_{(sum)} = \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \overline{0}^{k;\mathsf{w}}_{(sum,\, \mathrm{MinCW}^k(sum))} = \langle u_1, \ldots, u_k \rangle \tag{4.213}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} = \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad 0^{\mathsf{w}}_{(m)} = u_1 \ \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} = \langle u_2, \ldots, u_k \rangle \tag{4.214}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} = \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad 0^{\mathsf{p}}_{(m,p)} = u_1 \ \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} = \langle u_2, \ldots, u_k \rangle \tag{4.215}$$

$$\overline{1}^{k;\mathsf{kb}}_{(sum)} = \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \overline{1}^{k;\mathsf{w}}_{(sum,\, \mathrm{MaxCW}^k(sum))} = \langle u_1, \ldots, u_k \rangle \tag{4.216}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} = \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad 1^{\mathsf{w}}_{(m)} = u_1 \ \wedge \ \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} = \langle u_2, \ldots, u_k \rangle \tag{4.217}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} = \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad 1^{\mathsf{w}}_{(m,p)} = u_1 \ \wedge \ \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} = \langle u_2, \ldots, u_k \rangle \tag{4.218}$$

4.8.10 Reductions of Closed Stretchable Tuple Gap Orders

All reductions shown from this section to Section 4.8.15 are reductions of *proper* gap orders between tuples of the same weight. This amounts to assuming that in each redex (reduction target) we have an implicit side condition $u_1 \prec^\sharp v_1$ or $u_1 \leqq^\sharp v_1$ where u_1 (resp. v_1) is the leftmost immediate subterm of the left (resp. the right) operand.

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} \overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \tag{4.219}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \text{false} \tag{4.220}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \text{false} \tag{4.221}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.222}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)\geq n\\ n_1>0, n_1,n_2\leq n}} 0^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.223}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.224}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \left(\begin{array}{l} m = m' \;\rightarrow\; \displaystyle\bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1,n_2\leq(n+1)}} \left(\begin{array}{l} 0^{\mathsf{w}}_{(m)} <^{\mathsf{w}}_{n_1} 1^{\mathsf{w}}_{(m)} \\ \qquad \wedge\; \overline{0}^{k-1;\mathsf{kb}}_{sum-m} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{sum-m} \end{array}\right) \\ \qquad\qquad \wedge \\ m < m' \;\rightarrow \\ \displaystyle\bigvee_{1<r\leq n+1} \left(\begin{array}{c} l < \mathrm{CW}^k_1(sum,l,l') < \cdots < \mathrm{CW}^k_r(sum,l,l') < l' \\ \wedge \\ \displaystyle\bigvee_{\substack{\sum_{i=1}^r n_i = n+1\\ n_i>0}} \bigwedge_{0<i\leq r} \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,l,l'))} <^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,l,l'))} \end{array}\right) \end{array}\right) \tag{4.225}$$

where $l \equiv m-1$ *and* $l' \equiv m'+1$.

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.226}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.227}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n\\ n_2>0}} \left(\begin{array}{l} \overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \\ \qquad <^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} <^{k;\mathsf{pl}}_{n_4} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \end{array}\right) \tag{4.228}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.229}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.230}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_2>0}} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \tag{4.231}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.232}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.233}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n\\ n_3>0}} \left(\begin{array}{c} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \\ <^{k;\mathsf{w}}_{n_3} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_4} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \end{array} \right) \tag{4.234}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)\geq n\\ n_1>0, n_1,n_2\leq n}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.235}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.236}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n\\ n_3>0}} \left(\begin{array}{c} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \\ <^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_4} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \end{array} \right) \tag{4.237}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.238}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.239}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4+n+5=n\\ n_3>0}} \left(\begin{array}{c} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m,p)} \\ <^{k;\mathsf{w}}_{n_3} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_4} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \\ <^{k;\mathsf{pl}}_{n_5} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \end{array} \right) \tag{4.240}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.241}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)\geq n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.242}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \left(\begin{array}{c} (n=1 \;\rightarrow\; m<m') \\ \wedge \\ n>1 \;\rightarrow \\ \bigvee_{0<r<n} \left(\begin{array}{c} m<\mathrm{CW}^k_1(sum,m,m')<\cdots<\mathrm{CW}^k_r(sum,m,m')<m' \\ \wedge \\ \bigvee_{\substack{\sum_{i=1}^r n_i=n-1\\ n_i>0}} \bigwedge_{0<i\leq r} \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} <^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \end{array} \right) \end{array} \right) \tag{4.243}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.244}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.245}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_1>0}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_2} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.246}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.247}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.248}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_1>0}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \tag{4.249}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.250}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.251}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_1>0}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_2} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} <^{k;\mathsf{pl}}_{n_3} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.252}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.253}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.254}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_2>0}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \tag{4.255}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.256}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.257}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.258}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1-1)(n_2+1)\geq n-1\\ n_1>0,n_1,n_2<n}} 0^{\mathsf{p}}_{(m)} <^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m)} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.259}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.260}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n\\ n_2>0}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} <^{k;\mathsf{pl}}_{n_3} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \tag{4.261}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1)(n_2+1)\geq n\\ n_1>0,n_1,n_2\leq n}} 1^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.262}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \prec^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \text{false} \tag{4.263}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \prec^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_2>0}} \left(\begin{array}{c} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \prec^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \prec^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \\ \prec^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \prec^{k;\mathsf{pl}}_{n_4} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \end{array} \right) \tag{4.264}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \prec^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1(n_2+1)\geq n \\ n_1>0, n_1,n_2\leq n}} 1^{\mathsf{p}}_{(m,p)} \prec^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \wedge \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \prec^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.265}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \prec^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \text{false} \tag{4.266}$$

4.8.11 Reductions of Closed Rigid Tuple Gap Orders

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} \overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \tag{4.267}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \text{false} \tag{4.268}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \text{false} \tag{4.269}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n \\ n_2>0}} \overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.270}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1(n_2+1)=n \\ n_1>0, n_1,n_2\leq n}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m',p')} \wedge \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.271}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \text{false} \tag{4.272}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \left(\begin{array}{l} m = m' \ \rightarrow \ \bigvee\limits_{\substack{(n_1+1)(n_2+1)=n+1 \\ n_1,n_2 \leq (n+1)}} \left(\begin{array}{l} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{w}}_{n_1} 1^{\mathsf{w}}_{(m)} \\ \qquad \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{sum-m} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{sum-m} \end{array} \right) \\ \qquad\qquad \wedge \\ m < m' \ \rightarrow \\ \bigvee\limits_{1<r\leq n+1} \left(\begin{array}{c} l < \mathrm{CW}^k_1(sum,l,l') < \cdots < \mathrm{CW}^k_r(sum,l,l') < l' \\ \wedge \\ \neg \left(l \leq \mathrm{CW}^k_1(sum,l,l') < \cdots < \mathrm{CW}^k_{r+1}(sum,l,l') < l' \right) \\ \wedge \\ \bigvee\limits_{\substack{\sum_{i=1}^r n_i = n+1 \\ n_i > 0}} \bigwedge\limits_{0<i\leq r} \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,l,l'))} \leqq^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,l,l'))} \end{array} \right) \end{array} \right)$$

where $l \equiv m - 1$ *and* $l' \equiv m' + 1$. (4.273)

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1 \\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(m')} \ \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.274}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1 \\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{l}}_{n_1} 1^{\mathsf{w}}_{(m')} \ \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.275}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_2>0}} \left(\begin{array}{l} \overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \\ \qquad\qquad \leqq^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \leqq^{k;\mathsf{pl}}_{n_4} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \end{array} \right) \tag{4.276}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1 \\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \ \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.277}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1 \\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{w}}_{(m)} \leqq^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \ \wedge \ \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.278}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \tag{4.279}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \text{false} \tag{4.280}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad \mapsto \quad \text{false} \tag{4.281}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_3>0}} \left(\begin{array}{l} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \\ \qquad\qquad \leqq^{k;\mathsf{w}}_{n_3} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_4} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \end{array} \right) \tag{4.282}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)=n\\ n_1>0, n_1,n_2\leq n}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n_1} 0^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.283}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.284}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n\\ n_3>0}} \left(\begin{array}{l} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \\ \qquad\qquad \leqq^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_4} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \end{array} \right) \tag{4.285}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.286}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.287}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4+n+5=n\\ n_3>0}} \left(\begin{array}{l} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m,p)} \\ \qquad\qquad \leqq^{k;\mathsf{w}}_{n_3} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_4} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \\ \qquad\qquad \leqq^{k;\mathsf{pl}}_{n_5} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \end{array} \right) \tag{4.288}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.289}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1+1)(n_2+1)=n+1\\ n_1>0, n_1,n_2\leq n+1}} 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.290}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \left(\begin{array}{l} n=1 \;\rightarrow\; \left(m<m' \;\wedge\; \forall z \left(m<z<m' \;\rightarrow\; \neg\, \mathrm{IsTW}^k(m)\right)\right) \\ \qquad\qquad \wedge \\ n>1 \;\rightarrow \\ \displaystyle\bigvee_{0<r<n} \left(\begin{array}{c} m<\mathrm{CW}^k_1(sum,m,m')<\cdots<\mathrm{CW}^k_r(sum,m,m')<m' \\ \wedge \\ \neg\left(m<\mathrm{CW}^k_1(sum,m,m')<\cdots<\mathrm{CW}^k_{r+1}(sum,m,m')<m'\right) \\ \wedge \\ \displaystyle\bigvee_{\substack{\sum_{i=1}^{r} n_i=n-1\\ n_i>0}} \bigwedge_{0<i\leq r} \overline{0}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \leqq^{k;\mathsf{w}}_{n_i-1} \overline{1}^{k;\mathsf{w}}_{(sum,\mathrm{CW}^k_i(sum,m,m'))} \end{array} \right) \end{array} \right) \tag{4.291}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.292}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.293}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_1>0}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_2} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.294}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.295}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.296}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_1>0}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \tag{4.297}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.298}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.299}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_1>0}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_2} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \leqq^{k;\mathsf{pl}}_{n_3} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.300}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.301}$$

$$\overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.302}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_2>0}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \tag{4.303}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.304}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.305}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \tag{4.306}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \bigvee_{\substack{(n_1-1)(n_2+1)=n-1\\ n_1>0,n_1,n_2<n}} 0^{\mathsf{p}}_{(m)} \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m)} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.307}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \quad\mapsto\quad \text{false} \tag{4.308}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n\\ n_2>0}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \leqq^{k;\mathsf{pl}}_{n_3} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \tag{4.309}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \bigvee_{\substack{(n_1)(n_2+1)=n\\ n_1>0,n_1,n_2\leq n}} 1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(m')} \;\wedge\; \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.310}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m')} \quad\mapsto\quad \text{false} \tag{4.311}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3+n_4=n \\ n_2>0}} \left(\begin{array}{l} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \leqq^{k;\mathsf{w}}_{n_2} \overline{0}^{k;\mathsf{w}}_{(sum,m')} \\ \qquad\qquad \leqq^{k;\mathsf{pl}}_{n_3} \overline{0}^{k;\mathsf{p}}_{(sum,m',p')} \leqq^{k;\mathsf{pl}}_{n_4} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \end{array} \right) \tag{4.312}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \bigvee_{\substack{n_1(n_2+1)=n \\ n_1>0, n_1, n_2 \leq n}} 1^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m',p')} \wedge \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \tag{4.313}$$

$$\overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m',p')} \quad \mapsto \quad \text{false} \tag{4.314}$$

4.8.12 Reductions of Half Open Stretchable Tuple Gap Orders

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n} \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} \overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} <^{k;\mathsf{pl}}_{n_2} \langle u_1, \ldots, u_k \rangle \tag{4.315}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{p}}_{n} \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \bigvee_{\substack{n_1(n_2+1)+n_3 \geq n \\ n_1, n_2, n_3 \leq n}} \left(\begin{array}{lll} & 0^{\mathsf{w}}_{(m)} <^{\mathsf{p}}_{n_1} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_3} \langle u_2, \ldots, u_k \rangle \end{array} \right) \tag{4.316}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} <^{k;\mathsf{l}}_{n} \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \bigvee_{\substack{n_1(n_2+1)+n_3 \geq n \\ n_1, n_2, n_3 \leq n}} \left(\begin{array}{lll} & 0^{\mathsf{w}}_{(m)} <^{\mathsf{l}}_{n_1} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_3} \langle u_2, \ldots, u_k \rangle \end{array} \right) \tag{4.317}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n} \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_1>0}} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{w}}_{n_1} \overline{0}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} <^{k;\mathsf{pl}}_{n_2} \langle u_1, \ldots, u_k \rangle \tag{4.318}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{p}}_{n} \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \bigvee_{\substack{n_1(n_2+1)+n_3 \geq n \\ n_1, n_2, n_3 \leq n}} \left(\begin{array}{lll} & 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{p}}_{n_1} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_3} \langle u_2, \ldots, u_k \rangle \end{array} \right) \tag{4.319}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} <^{k;\mathsf{l}}_{n} \langle u_1, \ldots, u_k \rangle \quad \mapsto \quad \bigvee_{\substack{n_1(n_2+1)+n_3 \geq n \\ n_1, n_2, n_3 \leq n}} \left(\begin{array}{lll} & 0^{\mathsf{p}}_{(m,p)} <^{\mathsf{l}}_{n_1} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} <^{k-1;\mathsf{kb}}_{n_3} \langle u_2, \ldots, u_k \rangle \end{array} \right) \tag{4.320}$$

$$\langle u_1, \ldots, u_k \rangle <^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \quad \mapsto \quad \bigvee_{\substack{n_1+n_2=n \\ n_2>0}} \langle u_1, \ldots, u_k \rangle <^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} <^{k;\mathsf{w}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \tag{4.321}$$

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{p}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3\geq n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \prec_{n_1}^{\mathsf{p}} 1^{\mathsf{w}}_{(m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \prec_{n_2}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \prec_{n_3}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.322}$$

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{l}} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3\geq n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \prec_{n_1}^{\mathsf{l}} 1^{\mathsf{w}}_{(m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \prec_{n_2}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \prec_{n_3}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.323}$$

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{w}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_2>0}} \langle u_1,\ldots,u_k\rangle \prec_{n_1}^{k;\mathsf{pl}} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \prec_{n_2}^{k;\mathsf{w}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \tag{4.324}$$

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{p}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3\geq n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \prec_{n_1}^{\mathsf{p}} 1^{\mathsf{p}}_{(m,p)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \prec_{n_2}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \prec_{n_3}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.325}$$

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{l}} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3\geq n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \prec_{n_1}^{\mathsf{l}} 1^{\mathsf{p}}_{(m,p)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \prec_{n_2}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \prec_{n_3}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.326}$$

4.8.13 Reductions of Half Open Rigid Tuple Gap Orders

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq_n^{k;\mathsf{w}} \langle u_1,\ldots,u_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_1>0}} \overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq_{n_1}^{k;\mathsf{w}} \overline{0}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \leqq_{n_2}^{k;\mathsf{pl}} \langle u_1,\ldots,u_k\rangle \tag{4.327}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq_n^{k;\mathsf{p}} \langle u_1,\ldots,u_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & 0^{\mathsf{w}}_{(m)} \leqq_{n_1}^{\mathsf{p}} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq_{n_2}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq_{n_3}^{k-1;\mathsf{kb}} \langle u_2,\ldots,u_k\rangle \end{array}\right) \tag{4.328}$$

$$\overline{0}^{k;\mathsf{w}}_{(sum,m)} \leqq_n^{k;\mathsf{l}} \langle u_1,\ldots,u_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & 0^{\mathsf{w}}_{(m)} \leqq_{n_1}^{\mathsf{l}} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq_{n_2}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq_{n_3}^{k-1;\mathsf{kb}} \langle u_2,\ldots,u_k\rangle \end{array}\right) \tag{4.329}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq_n^{k;\mathsf{w}} \langle u_1,\ldots,u_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_1>0}} \overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq_{n_1}^{k;\mathsf{w}} \overline{0}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \leqq_{n_2}^{k;\mathsf{pl}} \langle u_1,\ldots,u_k\rangle \tag{4.330}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{p}}_{n} \langle u_1,\ldots,u_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{p}}_{n_1} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_3} \langle u_2,\ldots,u_k\rangle \end{array}\right) \tag{4.331}$$

$$\overline{0}^{k;\mathsf{p}}_{(sum,m,p)} \leqq^{k;\mathsf{l}}_{n} \langle u_1,\ldots,u_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & 0^{\mathsf{p}}_{(m,p)} \leqq^{\mathsf{l}}_{n_1} u_1 \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_3} \langle u_2,\ldots,u_k\rangle \end{array}\right) \tag{4.332}$$

$$\langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_2>0}} \langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \leqq^{k;\mathsf{w}}_{n_2} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \tag{4.333}$$

$$\langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{w}}_{(m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \leqq^{k-1;\mathsf{kb}}_{n_3} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.334}$$

$$\langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{w}}_{(sum,m)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \leqq^{\mathsf{l}}_{n_1} 1^{\mathsf{w}}_{(m)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \leqq^{k-1;\mathsf{kb}}_{n_3} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.335}$$

$$\langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{w}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \quad\mapsto\quad \bigvee_{\substack{n_1+n_2=n\\ n_2>0}} \langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{pl}}_{n_1} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \leqq^{k;\mathsf{w}}_{n_2} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \tag{4.336}$$

$$\langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{p}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \leqq^{\mathsf{p}}_{n_1} 1^{\mathsf{p}}_{(m,p)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \leqq^{k-1;\mathsf{kb}}_{n_3} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.337}$$

$$\langle u_1,\ldots,u_k\rangle \leqq^{k;\mathsf{l}}_{n} \overline{1}^{k;\mathsf{p}}_{(sum,m,p)} \quad\mapsto\quad \bigvee_{\substack{n_1(n_2+1)+n_3=n\\ n_1,n_2,n_3\leq n}} \left(\begin{array}{lll} & u_1 \leqq^{\mathsf{l}}_{n_1} 1^{\mathsf{p}}_{(m,p)} \\ \wedge & \overline{0}^{k-1;\mathsf{kb}}_{(sum-m)} \leqq^{k-1;\mathsf{kb}}_{n_2} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \\ \wedge & \langle u_2,\ldots,u_k\rangle \leqq^{k-1;\mathsf{kb}}_{n_3} \overline{1}^{k-1;\mathsf{kb}}_{(sum-m)} \end{array}\right) \tag{4.338}$$

4.8.14 Reductions of Open Stretchable Tuple Gap Orders

Let $sum = \sum_{i=1}^{k} u_i^{\mathsf{w}} = \sum_{i=1}^{k} v_i^{\mathsf{w}}$ and $rem = \sum_{i=2}^{k} u_i^{\mathsf{w}} = \sum_{i=2}^{k} v_i^{\mathsf{w}}$. Let $\mathrm{type}(u_1) = \alpha_p$ and $\mathrm{type}(v_1) = \alpha_{p'}$.

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{w}} \langle v_1,\ldots,v_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \left(\begin{array}{r} \langle u_1,\ldots,u_k\rangle \prec_{n_1}^{k;\mathsf{pl}} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \\ \prec_{n_2}^{k;\mathsf{w}} \overline{0}^{k;\mathsf{w}}_{(sum,v_1^{\mathsf{w}})} \prec_{n_3}^{k;\mathsf{pl}} \langle v_1,\ldots,v_k\rangle \end{array} \right) \tag{4.339}$$

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{p}} \langle v_1,\ldots,v_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \left(\begin{array}{r} \langle u_1,\ldots,u_k\rangle \prec_{n_1}^{k;\mathsf{l}} \overline{1}^{k;\mathsf{p}}_{(sum,u_1^{\mathsf{w}},p)} \\ \prec_{n_2}^{k;\mathsf{p}} \overline{0}^{k;\mathsf{p}}_{(sum,v_1^{\mathsf{w}},p')} \prec_{n_3}^{k;\mathsf{l}} \langle v_1,\ldots,v_k\rangle \end{array} \right) \tag{4.340}$$

$$\langle u_1,\ldots,u_k\rangle \prec_n^{k;\mathsf{l}} \langle v_1,\ldots,v_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_3>0}} \left(\begin{array}{c} \langle u_2,\ldots,u_k\rangle \prec_{n_1}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(rem)} \\ \wedge \\ \overline{0}^{k-1;\mathsf{kb}}_{(rem)} \prec_{n_2}^{k-1;\mathsf{kb}} \langle v_2,\ldots,v_k\rangle \\ \wedge \\ \displaystyle\bigvee_{\substack{(m_1+1)(m_2-1)\geq(n_3-1)\\ m_2>0,m_1,m_2<n_3}} \left(\begin{array}{c} \overline{0}^{k-1;\mathsf{kb}}_{(rem)} \prec_{m_1}^{k-1;\mathsf{kb}} \overline{1}^{k-1;\mathsf{kb}}_{(rem)} \\ \wedge \\ u_1 \prec_{m_2}^{\mathsf{l}} v_1 \end{array} \right) \end{array} \right) \tag{4.341}$$

4.8.15 Reductions of Open Rigid Tuple Gap Orders

Let $sum = \sum_{i=1}^k u_i^{\mathsf{w}} = \sum_{i=1}^k v_i^{\mathsf{w}}$ and $rem = \sum_{i=2}^k u_i^{\mathsf{w}} = \sum_{i=2}^k v_i^{\mathsf{w}}$. Let $\mathrm{type}(u_1) = \alpha_p$ and $\mathrm{type}(v_1) = \alpha_{p'}$.

$$\langle u_1,\ldots,u_k\rangle \leqq_n^{k;\mathsf{w}} \langle v_1,\ldots,v_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \left(\begin{array}{r} \langle u_1,\ldots,u_k\rangle \leqq_{n_1}^{k;\mathsf{pl}} \overline{1}^{k;\mathsf{w}}_{(sum,u_1^{\mathsf{w}})} \\ \leqq_{n_2}^{k;\mathsf{w}} \overline{0}^{k;\mathsf{w}}_{(sum,v_1^{\mathsf{w}})} \leqq_{n_3}^{k;\mathsf{pl}} \langle v_1,\ldots,v_k\rangle \end{array} \right) \tag{4.342}$$

$$\langle u_1,\ldots,u_k\rangle \leqq_n^{k;\mathsf{p}} \langle v_1,\ldots,v_k\rangle \quad\mapsto\quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_2>0}} \left(\begin{array}{r} \langle u_1,\ldots,u_k\rangle \leqq_{n_1}^{k;\mathsf{l}} \overline{1}^{k;\mathsf{p}}_{(sum,u_1^{\mathsf{w}},p)} \\ \leqq_{n_2}^{k;\mathsf{p}} \overline{0}^{k;\mathsf{p}}_{(sum,v_1^{\mathsf{w}},p')} \leqq_{n_3}^{k;\mathsf{l}} \langle v_1,\ldots,v_k\rangle \end{array} \right) \tag{4.343}$$

$$\langle u_1,\ldots,u_k\rangle \leqq_n^{k;\mathsf{l}} \langle v_1,\ldots,v_k\rangle \quad \mapsto \quad \bigvee_{\substack{n_1+n_2+n_3=n\\ n_3>0}} \left(\begin{array}{c} \langle u_2,\ldots,u_k\rangle \leqq_{n_1}^{k-1;\mathsf{kb}} \overline{1}_{(rem)}^{k-1;\mathsf{kb}} \\ \wedge \\ \overline{0}_{(rem)}^{k-1;\mathsf{kb}} \leqq_{n_2}^{k-1;\mathsf{kb}} \langle v_2,\ldots,v_k\rangle \\ \wedge \\ \bigvee\limits_{\substack{(m_1+1)(m_2-1)=(n_3-1)\\ m_2>0, m_1,m_2<n_3}} \left(\begin{array}{c} \overline{0}_{(rem)}^{k-1;\mathsf{kb}} \leqq_{m_1}^{k-1;\mathsf{kb}} \overline{1}_{(rem)}^{k-1;\mathsf{kb}} \\ \wedge \\ u_1 \leqq_{m_2}^{\mathsf{l}} v_1 \end{array} \right) \end{array} \right) \tag{4.344}$$

Chapter 5

Conclusion

02:07:59:900

> Neo : I know you're out there. I can feel you now. I know that you're afraid. You're afraid of us. You're afraid of change. I don't know the future. I didn't come here to tell you how this is going to end. I came here to tell you how it's going to begin. I'll hang up this phone. And then I'll show these people what you don't want them to see. I'm going to show them a world without you. A world without rules and controls, without borders or boundaries. A world where anything is possible. Where we go from there is a choice I leave to you.
>
> —*The Matrix*.

In the previous chapters we presented techniques to construct decision procedures for theories of data structures with integer arithmetic. In this chapter, we conclude this thesis with discussion of future research.

This thesis offers novel solutions to an important class of decision problems, the mixed constraints on data structures with quantitative properties. We developed the reduction technique, namely, extraction of accurate integer constraints from data constraints, and in case of quantified theories, reduction of quantifiers on data objects to quantifiers on integers. From the construction of accurate integer constraints that precisely characterize data constraints, we can derive decision procedures for the combined constraints by utilizing decision procedures for data structures and decision procedures for integer arithmetic. We presented decision procedures for term algebras with integers and decision procedures for queues with integers. Using our reduction technique and quantifier elimination, we proved the decidability of the first-order theory of *Knuth-Bendix Order*, thereby solving a long-standing open problem in term rewriting (officially listed as RTA open problem 99 since 2000).

We envisage that decision procedures will play a bigger role in formal methods, model checking and program analysis. They will render more valuable tools for specifying and analyzing security applications, and embedded and reactive systems. We plan to expand the thesis work in the following directions.

Security Verification. A vast majority of security problems of software systems is caused by memory access violations such as stack or heap overflow and out-of-bound array access. This brings unprecedented demands in reasoning about *memory safety* properties, that is, memory accesses, in terms of various data manipulations, always stay within designated boundaries. In fact, memory safety properties are a subclass of the more general *quantitative* properties of *resource reallocation* which can be expressed in the language of data structures with integer constraints. We believe work in this thesis can be used as the basis for specifying and verifying such quantitative properties.

High-level Static Analysis. Many advanced data structures are widely used in industry-sized applications such as Java Runtime Library and C++ Standard Template Library. They include linked lists, heaps, priority queues, hash tables, skip lists, splay trees, etc. Program reliability and efficiency rely on *high-level* properties of these data structures. The traditional *low-level* logic representation of these structures easily leads to undecidability. Here the challenge is to strike the right balance between expressive power and complexity. A specification language should be well-designed so that it can model the *core* properties of a data structure while retaining decidability or even low complexity. We believe more new decision procedures for advanced data structures will make important contributions to high-level static analysis.

Verification of Embedded and Reactive Systems. It is of essential importance to develop techniques for designing and analyzing *embedded* and *reactive* systems, as they are ubiquitous in our daily lives, particularly in many safety-critical devices that we use. One of the challenges that we would like to focus on is to carry out symbolic exploration of the state-space efficiently. Such

symbolic computation unavoidably involves quantified formulas, while many first-order theories either are undecidable or intractable due to high complexity. As many have observed, however, we hardly deal with formulas with a large quantifier alternation depth, and hence it is worthwhile to investigate the class of formulas that can have arbitrarily long sequences of quantifiers of the same kind while the total number of quantifier alternations is bounded by a constant number. In the search of quantifier elimination procedures for the theory of term algebras with integers and for the theory of queues with integers, we already aimed at and successfully obtained *block-wise* quantifier eliminations which are practically more efficient than *single-variable* quantifier eliminations. We propose to continue the development of more efficient quantifier eliminations, in particular elimination procedures for the combined theory of integer and real arithmetic, which finds applications in the verification of hybrid systems and real-time systems.

More Powerful Decision Theories. On the theoretical front, we plan to search for new powerful tools to prove decision problems. Currently we are investigating the decidability of the theory of queues with integers and with subsequence relations including *subqueue*, *prefix* and *suffix* relations. This theory has a very strong expressive power; it can interpret the theory of word concatenation with length function, the theory of Presburger arithmetic with divisibility predicate, the theory of arrays, etc. Determining the decidability of this theory will have far-reaching consequences for solving other decision problems. In particular, it could give us a better understanding and classification of solutions to word equations. Besides the theoretical importance, the decidability of this theory may give us more powerful algorithms in pattern matching, which has numerous applications in computer science. It can precisely characterize the semantics of common string operations in the C language, and hence would be a powerful tool to reason about memory safety properties. It may also lead to a decision procedure for the theory of unbounded bit-vectors which potentially has many applications in hardware verification.

00:44:07:932

> Oracle : . . . Because you didn't come here to make the choice. You've already made it. You're here to try to understand why you made it.
>
> —*The Matrix Reloaded*.

Bibliography

[1] Alessandro Armando, Silvio Ranise, and Michaël Rusinowitch. Uniform derivation of decision procedures by superposition. In *Proceedings of the 15th International Workshop on Computer Science Logic (CSL'01)* volume 2142 of *Lecture Notes in Computer Science*, pages 513–527. Springer-Verlag, 2001.

[2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, Cambridge, UK, 1999.

[3] Rolf Backofen. A complete axiomatization of a theory with feature and arity constraints. *Journal of Logical Programming*, 24(1&2):37–71, 1995.

[4] Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. A model-theoretic approach to regular string relations. In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science*, pages 431–440. IEEE Computer Society Press, 2001.

[5] Nikolaj S. Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Computer Science Department, Stanford University, November 1998.

[6] Nikolaj S. Bjørner, Anca Browne, Michael Colón, Bernd Finkbeiner, Zohar Manna, Henny B. Sipma, and Tomás E. Uribe. Verifying temporal properties of reactive systems: A STeP tutorial. *Formal Methods in System Design*, 16(3):227–270, June 2000.

[7] Hubert Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.

[8] Hubert Comon and Catherine Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, 1994.

[9] Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.

[10] Hubert Comon and Ralf Treinen. Ordering constraints on trees. In Sophie Tison, editor, *Proceedings of the 19th International Colloquium on Trees in Algebra and Programming (CAAP'94)*, volume 787 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 1994.

[11] Hubert Comon and Ralf Treinen. The first-order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science*, 176(1-2):67–87, 1997.

[12] David C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, volume 7, pages 91–99. American Elsevier, 1972.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 2001.

[14] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 7:279–301, 1982.

[15] Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. Variations of the common subexpression problem. *Journal of ACM*, 27:758–771, 1980.

[16] Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2001.

[17] Solomon Feferman and Robert L. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.

[18] Silvio Ghilardi. Model-theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 33(3-4):221–249, 2005.

[19] Wilfrid Hodges. *Model Theory*. Cambridge University Press, Cambridge, UK, 1993.

[20] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[21] Jean-Pierre Jouannaud and Mitsuhiro Okada. Satisfiability of systems of ordinal notation with the subterm property is decidable. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 455–468. Springer-Verlag, 1991.

[22] H. Jerome Keisler and Chen C. Chang. *Model Theory*. Elsevier Science, Netherlands, 1990.

[23] Claude Kirchner, Hélène Kirchner, and Michaël Rusinowitch. Deduction with symbolic constraints. *Revue Francaise d' Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automated deduction.

[24] Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Proceedings of the 30th International Colloquium on Automata, Languages and Programming, ICALP'2003*, volume 2719 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[25] Donald E. Knuth and Peter Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970. Reprinted in *Automation of Reasoning, Vol. 2* Jürgen Siekmann and G. Wrightson, editors, pages 342-376, Springer-Verlag, 1983.

[26] Konstantin Korovin and Andrei Voronkov. A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering. In *Proceedings of the 15th IEEE Symposium on Logic in Computer Science*, pages 291 – 302. IEEE Computer Society Press, 2000.

[27] Konstantin Korovin and Andrei Voronkov. Knuth-Bendix constraint solving is NP-complete. In *Proceedings of 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *Lecture Notes in Computer Science*, pages 979–992. Springer-Verlag, 2001.

[28] Konstantin Korovin and Andrei Voronkov. The decidability of the first-order theory of the Knuth-Bendix order in the case of unary signatures. In *Proceedings of the 22th Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS'02)*, volume 2556 of *Lecture Notes in Computer Science*, pages 230–240. Springer-Verlag, 2002.

[29] Viktor Kuncak and Martin Rinard. On the theory of structural subtyping. Technical Report MIT-LCS-TR-879, Massachusetts Institute of Technology, January 2003.

[30] Viktor Kuncak and Martin Rinard. The structural subtyping of non-recursive types is decidable. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science*, pages 96–107. IEEE Computer Society Press, 2003.

[31] Viktor Kuncak and Martin Rinard. An algorithm for deciding BAPA: Boolean algebra with Presburger arithmetic. In *Proceedings of the 20th International Conference on Automated Deduction (CADE'05)*, volume 3632 of *Lecture Notes in Computer Science*, pages 260–277. Springer-Verlag, 2005.

[32] M. Lothaire. *Combinatorics on Words*. Addison-Wesley, Massachusetts, USA, 1983. M. Lothaire is a joint pseudonym for the following: Robert Cor, Dominque Perrin, Jean Berstel, Christian Choffrut, Dominque Foata, Jean Eric Pin, Guiseppe Pirillo, Christophe Reutenauer, Marcel P. Schutzenberger, Jadcques Sakaroovitch, and Imre Simon.

[33] László Lovász. *Combinatorial Problems and Exercises*. Elsevier, Horth-Holland, 1993.

[34] Michael J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite tree. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science*, pages 348–357. IEEE Computer Society Press, 1988.

[35] Gennady S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977.

[36] Anatoly I. Malcev. Axiomatizable classes of locally free algebras of various types. In *The Metamathematics of Algebraic Systems, Collected Papers*, chapter 23, pages 262–281. North Holland, 1971.

[37] Paliath Narendran and Michaël Rusinowitch. The theory of total unary RPO is decidable. In *Proceedings of the 1st International Conference on Computational Logic (CL 2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 660–672. Springer-Verlag, 2000.

[38] Paliath Narendran, Michaël Rusinowitch, and Rakesh M. Verma. RPO constraint solving is in NP. In *Proceedings of the 12th International Workshop on Computer Science Logic (CSL 98)*, volume 1584 of *Lecture Notes in Computer Science*, pages 385 – 398. Springer-Verlag, 1999.

[39] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Transaction on Programming Languages and Systems*, 1(2):245–257, October 1979.

[40] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of ACM*, 27(2):356–364, April 1980.

[41] Robert Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47(2):65–69, 1993.

[42] Robert Nieuwenhuis and José M. Rivero. Solved forms for path ordering constraints. In *Proceeding of 10th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1631 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1999.

[43] Robert Nieuwenhuis and Albert Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computation*, 19(4):321–351, 1995.

[44] Derek C. Oppen. Elementary bounds for Presburger arithmetic. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 34–37. ACM Press, 1973.

[45] Derek C. Oppen. Reasoning about recursively defined data structures. *Journal of ACM*, 27(3), July 1980.

[46] Cattamanchi R. Reddy and Donald W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 320–325. ACM Press, 1978.

[47] Peter Revesz. Quantifier-elimination for the first-order theory of boolean algebras with linear cardinality constraints. In *Proceedings of the 18th Conference on Advances in Databases and Information Systems (ADBIS'04)*, volume 3255 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, 2004.

[48] Tatiana Rybina and Andrei Voronkov. A decision procedure for term algebras with queues. In *Proceedings of the 15th IEEE Symposium on Logic in Computer Science*, pages 279 – 290. IEEE Computer Society Press, 2000.

[49] Tatiana Rybina and Andrei Voronkov. A decision procedure for term algebras with queues. *ACM Transactions on Computational Logic*, 2(2):155–181, 2001.

[50] Tatiana Rybina and Andrei Voronkov. Upper bounds for a theory of queues. In *Proceedings of 30th International Colloquium on Automata, Languages and Programming (ICALP'03)*, volume 2719 of *Lecture Notes in Computer Science*, pages 714–724. Springer-Verlag, 2003.

[51] Thoralf A. Skolem. Untersuchungen über die Axiome des Klassenkalküls und über Produktations- und Summationsprobleme, welche gewisse Klassen von Aussagen betreffen. In Jens Erik Fenstad, editor, *Selected works in logic*, pages 67–101. Universitetsforlaget, 1970.

[52] Wolfgang Thomas. Infinite trees and automaton-definable relations over ω-words. *Theoretical Computer Science*, 103:143–159, 1992.

[53] Cesare Tinelli and Mehdi T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and Klaus U. Schulz, editors, *Proceedings of the 1st International Workshop on Frontiers of Combining Systems (FroCos'96)*, Applied Logic Series, Vol. 3, pages 103–120. Kluwer Academic Publishers, 1996.

[54] Cesare Tinelli and Christophe Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theoretical Computer Science*, 290(1):291–353, January 2003.

[55] Cesare Tinelli and Calogero G. Zarba. Combining decision procedures for sorted theories. In José Júlio Alferes and João Alexandre Leite, editors, *Proceedings of the 9th European Conference on Logic in Artificial Intelligence (JELIA'04)*, volume 3229 of *Lecture Notes in Computer Science*, pages 641–653. Springer-Verlag, 2004.

[56] Ralf Treinen. A new method for undecidability proofs of first order theories. *Journal of Symbolic Computation*, 14:437–457, 1992.

[57] K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebras. *Journal of ACM*, 34(2):492–510, 1987.

[58] Calogero G. Zarba. Combining multisets with integers. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 363–376. Springer-Verlag, 2002.

[59] Calogero G. Zarba. Combining sets with integers. In Alessandro Armando, editor, *Proceedings of the 4th International Workshop on Frontiers of Combining Systems (FroCoS'02)*, volume 2309 of *Lecture Notes in Artificial Intelligence*, pages 103–116. Springer-Verlag, 2002.

[60] Ting Zhang, Henny B. Sipma, and Zohar Manna. Decision procedures for recursive data structures with integer constraints. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning (IJCAR'04)*, volume 3097 of *Lecture Notes in Computer Science*, pages 152–167. Springer-Verlag, 2004.

[61] Ting Zhang, Henny B. Sipma, and Zohar Manna. Term algebras with length function and bounded quantifier alternation. In *Proceedings of the 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'04)*, volume 3223 of *Lecture Notes in Computer Science*, pages 321–336. Springer-Verlag, 2004.

[62] Ting Zhang, Henny B. Sipma, and Zohar Manna. The decidability of the first-order theory of term algebras with Knuth-Bendix order. In Robert Nieuwenhuis, editor, *the* 20^{th} *International Conference on Automated Deduction (CADE'05)*, volume 3632 of *Lecture Notes in Computer Science*, pages 131–148. Springer-Verlag, 2005.

[63] Ting Zhang, Henny B. Sipma, and Zohar Manna. Decision procedures for queues with integer constraints. In R. Ramanujam and Sandeep Sen, editors, *Proceedings of the 25th International Conference on the Foundatations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 225–237. Springer-Verlag, 2005.

[64] Ting Zhang, Henny B. Sipma, and Zohar Manna. Decision procedures for term algebras with integer constraints. *Information and Computation*, 204(10):1526–1574, October 2006.

[65] Zohar Manna, Henny B. Sipma, and Ting Zhang. Verifying Balanced Trees. *Symposium on Logical Foundations of Computer Science*, 2007. To appear.

List of Definitions

Definition 2.1 **Term Algebras** . . . 13
Definition 2.2 **Type Completeness** . . . 15
Definition 2.3 **Convexity Theory** . . . 17
Definition 2.4 **DAG Representation** . . . 17
Definition 2.5 **Unification Closure** . . . 18
Definition 2.6 **Congruence Closure** . . . 18
Definition 2.7 **Term Algebras with Integers** . . . 20
Definition 2.8 **Length Constraint Completion (LCC) in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 22
Definition 2.9 **Relativized LCC (RLCC) in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 24
Definition 2.10 **Counting Constraints** . . . 28
Definition 2.11 **Equality Completeness** . . . 28
Definition 2.12 **Red-black Trees [13]** . . . 32
Definition 2.13 **Colored Trees** . . . 32
Definition 2.14 **Structure of Red-black Trees** . . . 33
Definition 2.15 **Relativized LCC (RLCC) in** $\mathrm{RB}_{\mathbb{Z}}$ . . . 34
Definition 2.16 **Solved Form in** TA . . . 49
Definition 2.17 **Clusters** . . . 54
Definition 2.18 **RLCC with Parameters in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 55
Definition 2.19 **Strongly Solved Form** . . . 56
Definition 3.1 **Queues** . . . 85
Definition 3.2 **Conjugacy [32]** . . . 87
Definition 3.3 **Primitive Words** . . . 87
Definition 3.4 **Solved Form in** $\mathfrak{Q}$ . . . 88
Definition 3.5 **Normal Form in** $\mathfrak{Q}$ . . . 88
Definition 3.6 **Queues with Integers** . . . 90
Definition 3.7 **Length Constraint Completion (LCC) in** $\mathfrak{Q}_{\mathbb{Z}}$ . . . 91
Definition 3.8 **Relativized LCC (RLCC) in** $\mathfrak{Q}_{\mathbb{Z}}$ . . . 91
Definition 3.9 **Equality Completeness** . . . 92

Definition 3.10 Normal Form in $\mathfrak{Q}_{\mathbb{Z}}$ . . . 92
Definition 3.11 Delimiters in $\mathfrak{Q}$ . . . 95
Definition 3.12 Length Configurations in $\mathfrak{Q}_{\mathbb{Z}}$ . . . 95
Definition 3.13 Queues with Prefix . . . 97
Definition 3.14 Queues with Prefix and Integers . . . 97
Definition 3.15 Normal Form in $\mathfrak{Q}^{+}_{\mathbb{Z}}$ . . . 97
Definition 3.16 Prefix Completeness . . . 98
Definition 3.17 Delimiters in $\mathfrak{Q}^{+}$ . . . 106
Definition 3.18 Length Configurations in $\mathfrak{Q}^{+}_{\mathbb{Z}}$ . . . 109
Definition 3.19 RLCC with Parameters in $\mathfrak{Q}_{\mathbb{Z}}$ . . . 111
Definition 3.20 Normal Form with Parameters in $\mathfrak{Q}_{\mathbb{Z}}$ . . . 111
Definition 4.1 Knuth-Bendix Order [25] . . . 126
Definition 4.2 Term Algebras with Knuth-Bendix Order . . . 126
Definition 4.3 Decomposition of KBO . . . 128
Definition 4.4 Gap Orders . . . 128
Definition 4.5 Boundary Functions . . . 129
Definition 4.6 Term Algebras with KBO and Its Extensions . . . 130
Definition 4.7 KBO on Tuples . . . 130
Definition 4.8 Suborders on Tuples . . . 130
Definition 4.9 Tuple Gap Orders . . . 130
Definition 4.10 Tuple Boundary Functions . . . 131
Definition 4.11 Gap Order Completion . . . 132
Definition 4.12 Delineated Gap Order Completion . . . 133

List of Algorithms

Algorithm 2.1 **Oppen's Decision Procedure for** $\mathrm{Th}^{\forall}(\mathrm{TA})$ **[45]** . . . 18
Algorithm 2.2 **Decision Procedure for** $\mathrm{Th}^{\forall}(\mathrm{TA})$ **with Selectors** . . . 19
Algorithm 2.3 **Generic Decision Procedure** . . . 25
Algorithm 2.4 **Computation of LCC in** $\mathrm{TA}_{\mathbb{Z}}^{\omega}$ . . . 26
Algorithm 2.5 **Decision Procedure for** $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}}^{\omega})$ . . . 26
Algorithm 2.6 **Computation of RLCC in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 29
Algorithm 2.7 **Decision Procedure for** $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}})$ . . . 29
Algorithm 2.8 **Left-Rotation of Trees** . . . 31
Algorithm 2.9 **Computation of RLCC in** $\mathrm{RB}_{\mathbb{Z}}$ . . . 34
Algorithm 2.10 **RB-Insertion with Left-Rotation and Right-Rotation Integrated** . . . 36
Algorithm 2.11 **Normalization in** TA . . . 49
Algorithm 2.12 **Decomposition of Constructor Terms** . . . 51
Algorithm 2.13 **Quantifier Elimination in** TA . . . 52
Algorithm 2.14 **Normalization with Parameters** $\mathrm{TA}_{\mathbb{Z}}$ . . . 57
Algorithm 2.15 **Cluster Completion** . . . 58
Algorithm 2.16 **Decomposition of Disequalities Containing** $\bar{\mathrm{x}}$ . . . 59
Algorithm 2.17 **Elimination of Term Quantifiers in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 61
Algorithm 3.1 **Normalization in** $\mathfrak{Q}$**, cf. [5]** . . . 89
Algorithm 3.2 **Decision Procedure for Queues [5]** . . . 90
Algorithm 3.3 **Generic Decision Procedure** . . . 92
Algorithm 3.4 **Equality Completion** . . . 92
Algorithm 3.5 **Normalization in** $\mathfrak{Q}_{\mathbb{Z}}$ . . . 92
Algorithm 3.6 **Computation of** $\Phi_{\Delta+}$ **in** $\mathfrak{Q}_{\mathbb{Z}}$ . . . 94
Algorithm 3.7 **Decision Procedures for** $\mathfrak{Q}_{\mathbb{Z}}$ . . . 96
Algorithm 3.8 **Prefix Completion** . . . 98
Algorithm 3.9 **Elimination of Single Variable Literals** . . . 99
Algorithm 3.10 **Simplification of Negative** $\prec$ **Literals** . . . 99
Algorithm 3.11 **Upward Alignment** . . . 100

Algorithm 3.12 Downward Alignment . . . 100
Algorithm 3.13 Simplification of Positive $<$ Literals . . . 101
Algorithm 3.14 Loop Elimination . . . 101
Algorithm 3.15 Multi-path Elimination . . . 102
Algorithm 3.16 Alignment of Positive $<$ Literals . . . 103
Algorithm 3.17 Chain Normalization . . . 103
Algorithm 3.18 Alignment with Presence of Orbit Literals . . . 103
Algorithm 3.19 Elimination of Chains with Disequalities . . . 105
Algorithm 3.20 Elimination of Chains with Negative $\prec$-Literals . . . 106
Algorithm 3.21 Elimination of Negative $\prec$-Literals with Chains . . . 107
Algorithm 3.22 Normalization in $\mathfrak{Q}^+_{\mathbb{Z}}$ . . . 107
Algorithm 3.23 Computation of Φ_{Δ^+} in $\mathfrak{Q}^+_{\mathbb{Z}}$ . . . 108
Algorithm 3.24 Decision Procedure for $\mathfrak{Q}^+_{\mathbb{Z}}$ . . . 109
Algorithm 3.25 Normalization with Parameters in $\mathfrak{Q}_{\mathbb{Z}}$ . . . 111
Algorithm 3.26 Elimination of Queue Quantifiers . . . 112
Algorithm 4.1 Elimination of Integer Quantifiers in $\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+}$ . . . 136
Algorithm 4.2 Elimination of Term Quantifiers in $\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+}$ . . . 137

List of Propositions

Proposition 2.1 **Additivity of RLCC** . . . 24
Proposition 2.2 **LCC in $\mathrm{TA}^{\omega}_{\mathbb{Z}}$** . . . 26
Proposition 2.3 **RLCC in $\mathrm{TA}_{\mathbb{Z}}$** . . . 29
Proposition 2.4 **Strongly Solved Form** . . . 61
Proposition 2.5 **Linear Combination** . . . 67
Proposition 3.1 **Elimination of Selectors** . . . 86
Proposition 3.2 **Conjugacy [32]** . . . 87
Proposition 3.3 **Exclusiveness of Orbit Literals [5, 48]** . . . 88
Proposition 3.4 **Normal Form in $\mathfrak{Q}$** . . . 89
Proposition 3.5 **Normal Form in $\mathfrak{Q}_{\mathbb{Z}}$** . . . 94
Proposition 3.6 **Normalization in $\mathfrak{Q}^{+}_{\mathbb{Z}}$** . . . 108
Proposition 3.7 **Tags in $\alpha\beta$** . . . 115
Proposition 3.8 **Tags in Chains** . . . 117
Proposition 3.9 **Tags in αX** . . . 119

List of Lemmas

Lemma 4.1 **Elimination of Negative Literals** . . . 133
Lemma 4.2 **Delineated Gap Order Completion** . . . 134
Lemma 4.3 **No Embedding of Boundary Terms** . . . 134
Lemma 4.4 **Reduction of Term Gap Order Literals** . . . 134
Lemma 4.5 **Reduction of Closed Term Literals** . . . 134
Lemma 4.6 **Reduction of Non-closed Term Gap Order Literals** . . . 135
Lemma 4.7 **Reduction of Tuple Literals** . . . 135
Lemma 4.8 **Elimination of Term Quantifiers** . . . 135
Lemma 4.9 **Elimination of Integer Quantifiers** . . . 135
Lemma 4.10 **Soundness of Algorithms 4.1** . . . 138
Lemma 4.11 **Termination** . . . 139
Lemma 4.12 **Open Gap Order Literals in DGOCs** . . . 149

List of Theorems

Theorem 2.1 **Reduction with LCC [60]** . . . 23
Theorem 2.2 **Reduction with RLCC** . . . 25
Theorem 2.3 **LCC in** $\mathrm{TA}^{\omega}_{\mathbb{Z}}$ . . . 26
Theorem 2.4 **Counting Constraints in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 28
Theorem 2.5 **RLCC in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 29
Theorem 2.6 **Counting Constraints in** $\mathrm{RB}_{\mathbb{Z}}$ . . . 34
Theorem 2.7 **RLCC in** $\mathrm{RB}_{\mathbb{Z}}$ . . . 35
Theorem 2.8 **Decidability of** $\mathrm{Th}^{\forall}(\mathrm{RB}_{\mathbb{Z}})$ . . . 35
Theorem 2.9 **Soundness of Algorithm 2.13** . . . 52
Theorem 2.10 **Complexity of Algorithm 2.13** . . . 52
Theorem 2.11 **RLCC with Parameters in** $\mathrm{TA}_{\mathbb{Z}}$ . . . 56
Theorem 2.12 **Soundness of Algorithm 2.17** . . . 61
Theorem 2.13 **Complexity of Algorithm 2.17** . . . 61
Theorem 3.1 **Exclusiveness of Orbit Literals [5, 48]** . . . 88
Theorem 3.2 **RLCC in** $\mathfrak{Q}_{\mathbb{Z}}$ . . . 95
Theorem 3.3 **Complexity of** $\mathrm{Th}^{\forall}(\mathfrak{Q}_{\mathbb{Z}})$ . . . 96
Theorem 3.4 **RLCC in** $\mathfrak{Q}^{+}_{\mathbb{Z}}$ . . . 109
Theorem 3.5 **Complexity of** $\mathrm{Th}^{\forall}(\mathfrak{Q}^{+}_{\mathbb{Z}})$ . . . 110
Theorem 3.6 **RLCC with Parameters in** $\mathfrak{Q}_{\mathbb{Z}}$ . . . 112
Theorem 3.7 **Soundness of Algorithm 3.26** . . . 113
Theorem 4.1 **Decidability** . . . 139

List of Example

Example 2.1 LISP Lists . . . 14
Example 2.2 Type Completeness . . . 15
Example 2.3 Type Completion . . . 16
Example 2.4 Type Completion and Simplification . . . 16
Example 2.5 Oppen's Algorithm . . . 19
Example 2.6 Notation for Pseudo Integer Variables . . . 21
Example 2.7 Hidden Length Constraints . . . 21
Example 2.8 More Hidden Length Constraints . . . 22
Example 2.9 LCC in $\mathrm{List}_{\mathbb{Z}}$ . . . 23
Example 2.10 RLCC in List . . . 24
Example 2.11 LCC in $\mathrm{TA}^{\omega}_{\mathbb{Z}}$ . . . 27
Example 2.12 Hidden Length Constraints in $\mathrm{TA}_{\mathbb{Z}}$ . . . 27
Example 2.13 Counting Constraints . . . 28
Example 2.14 Equality Completion . . . 29
Example 2.15 RLCC in $\mathrm{TA}_{\mathbb{Z}}$ . . . 30
Example 2.16 Elimination of Selectors . . . 50
Example 2.17 Solving Equalities . . . 51
Example 2.18 Quantifier Elimination in TA . . . 53
Example 2.19 Mutually Independent Clusters . . . 54
Example 2.20 Cluster Completeness . . . 55
Example 2.21 Strongly Solved Form . . . 56
Example 2.22 Removal of Disequalities . . . 60
Example 2.23 Quantifier Elimination in $\mathrm{TA}_{\mathbb{Z}}$ . . . 61
Example 2.24 Assignment in $\mathrm{TA}^{\omega}_{\mathbb{Z}}$ . . . 69
Example 3.1 Orbits . . . 87
Example 3.2 Conjugacy . . . 87
Example 3.3 Primitive Words . . . 87
Example 3.4 Exclusiveness of Orbit Literals . . . 88

Example 3.5 **Hidden Length Constraints in $\mathfrak{Q}_{\mathbb{Z}}$** 90
Example 3.6 **LCC in $\mathfrak{Q}_{\mathbb{Z}}$** 91
Example 3.7 **RLCC in $\mathfrak{Q}_{\mathbb{Z}}$** 91
Example 3.8 **Anomaly** 94
Example 3.9 **Computation of L_t** 95
Example 3.10 **Blocks** 96
Example 4.1 **Elimination in Linear Order** 129
Example 4.2 **Extension of Knuth-Bendix Order** 130
Example 4.3 **Gap Order Completion** 132
Example 4.4 **Order Arrangement** 132
Example 4.5 **Delineated Gap Order Completion** 133
Example 4.6 **Quantifier Elimination in $\mathrm{List}^{\mathbb{Z}}_{\mathrm{kb}^+}$** 139

List of Figures

2-1 The DAG of $\text{cons}(y,z) = \text{cons}(x,z) \wedge \text{cons}(x,y) \neq z$. 18
2-2 The DAG of $\text{cons}(y,z) = \text{cons}(\text{cdr}(x),z) \wedge \text{cons}(\text{car}(x),y) \neq x \wedge \text{Is}_{\text{cons}}(x)$. 19
2-3 The DAG of $x = \text{cons}(\text{car}(y),y) \wedge |\text{cons}(\text{car}(y),y)| < 2|\text{car}(x)| \wedge \text{Is}_{\text{cons}}(y)$. 26
2-4 A run of RB-Insertion. 37
2-5 Paths from the root of the tree to x. In each of (a)-(d), the first row shows the sequence of nodes from the root to x; the second row shows whether the node above it is a left ($\leftarrow$) or right ($\rightarrow$) sibling; the third row shows the sibling tree of the node in the top row. 38
2-6 A detailed run of RB-Insertion step (b). 40
2-7 A detailed run of RB-Insertion step (c). 41
2-8 A detailed run of RB-Insertion step (d) with $x-2 = \mathit{root}$. 42
2-9 A detailed run of RB-Insertion step (d) with $x-2 \neq \mathit{root} \wedge T[x-2].\mathit{dir} = \mathit{right}$. 44
2-10 A detailed run of RB-Insertion step (d) with $x-2 \neq \mathit{root} \wedge T[x-2].\mathit{dir} = \mathit{right}$ (continued). 45
2-11 A detailed run of RB-Insertion step (d) with $x-2 \neq \mathit{root} \wedge T[x-2].\mathit{dir} = \mathit{left}$. 46
2-12 A detailed run of RB-Insertion step (d) with $x-2 \neq \mathit{root} \wedge T[x-2].\mathit{dir} = \mathit{left}$ (continued). 47
2-13 The DAG of $\text{Is}_{\text{cons}}(x) \wedge \text{cons}(y,z) = \text{cons}(\text{cdr}(x),z) \wedge \text{car}(x) \neq z$. 69

3-1 *aaba*, *abaa*, *aaab* and *baaa* are pairwise conjugate. 87

4-1 An order arrangement of $x_1 \prec^{\mathsf{w}}_{n_1} x_2 \prec^{\mathsf{p}}_{n_2} x_3 \prec^{\mathsf{l}}_{n_3} x_4$. 132
4-2 Quantifier Elimination in $\text{TA}^{\mathbb{Z}}_{\text{kb}^+}$. 138
4-3 Quantifier Elimination on $(\exists x)\,[\text{car}(x) \prec^{\mathsf{l}}_2 \text{cdr}(\text{cdr}(x)) \wedge \text{cdr}(\text{cdr}(\text{car}(x))) \prec^{\mathsf{l}}_3 y]$. 139

Index

A, *see* Tester
$\mathcal{A}$, *see* Constant
black, *see* Red-black tree
C, *see* Constructor
CLS, *see* Cluster
$\widehat{\mathrm{CNT}}^{\sharp}_{\natural}$, *see* Counting constraint, Integer function
$\mathrm{CNT}^{\sharp}_{\natural}$, *see* Counting constraint, Integer function
$\mathrm{CWL}^{\sharp}_{\natural}$, *see* Integer function
$\mathrm{CWS}^{\sharp}_{\natural}$, *see* Integer function
$\mathrm{CW}^{\sharp}_{\natural}$, *see* Integer function
DGOC, *see* Delineated gap order completion
Is, *see* Tester
$\mathrm{IsCW}^{\sharp}$, *see* Integer function
$\mathrm{IsTW}^{\sharp}$, *see* Integer function
$\mathscr{L}_{\mathcal{A}}$, *see* Queue
$\mathscr{L}_{\mathrm{kb}}$, *see* Term algebra with KBO
$\mathscr{L}^{\mathbb{Z}}_{\mathrm{kb}^+}$, *see* Term algebra with KBO and integers
$\mathscr{L}_{Q}$, *see* Queue
$\mathscr{L}^{\mathbb{Z}}_{Q}$, *see* Queue with integers
$\mathscr{L}_{Q^+}$, *see* Queue with prefix
$\mathscr{L}^{\mathbb{Z}}_{Q^+}$, *see* Queue with prefix and integers
$\mathscr{L}_{\mathrm{RB}}$, *see* Colored tree
$\mathscr{L}^{\mathbb{Z}}_{\mathrm{RB}}$, *see* Red-black tree
$\mathscr{L}_{\mathbb{T}}$, *see* Term algebra
$\mathscr{L}^{\mathbb{Z}}_{\mathbb{T}}$, *see* Term algebra with integers
$\mathscr{L}_{\mathbb{Z}}$, *see* Presburger arithmetic
List, *see* LISP list
$\mathrm{MaxCW}^{\sharp}$, *see* Integer function
$\mathrm{MinCW}^{\sharp}$, *see* Integer function
$\mathrm{Node}^{\sharp}$, 25
PA, *see* Presburger arithmetic
PRE, *see* Delimiter
Q, *see* Queue
$\mathfrak{Q}_{\mathbb{Z}}$, *see* Queue with integers
$\mathfrak{Q}^{+}_{\mathbb{Z}}$, *see* Queue with prefix and integers
$\mathfrak{Q}^{+}$, *see* Queue with prefix
$\mathfrak{Q}$, *see* Queue
RB, *see* Colored tree
$\mathrm{RB}_{\mathbb{Z}}$, *see* Red-black tree
$R\!\downarrow$, *see* Unification closure
$R\!\downarrow\!\uparrow$, *see* Bidirectional closure
red, *see* Red-black tree
$R\!\uparrow$, *see* Congruence closure
SUF, *see* Delimiter
TA, *see* Term algebra
$\mathrm{TA}_{\mathrm{kb}}$, *see* Term algebra with KBO
$\mathrm{TA}^{\mathbb{Z}}_{\mathrm{kb}^+}$, *see* Term algebra with KBO and integers
$\mathrm{TA}_{\mathbb{Z}}$, *see* Term algebra with integers
$\mathrm{TA}^{\omega}_{\mathbb{Z}}$, *see* Term algebra with integers
TA^{ω}, *see* Term algebra
$\mathbb{T}$, *see* Term domain
$\mathbb{T}_{\mathrm{rb}}$, *see* Colored tree
Th, *see* Theory
$\mathrm{Th}^{\forall}$, *see* Quantifier-free theory
$\mathrm{Tree}^{\sharp}$, 25
WD, *see* Well-definedness
$\mathbb{Z}$, *see* Presburger arithmetic
$\epsilon_{\mathcal{A}}$, *see* Queue
ar, *see* Arity, *see* Constructor
$\mathcal{T}$, 5

$\mathcal{V}$, 5
car, *see* LISP list
cdr, *see* LISP list
cons, *see* LISP list
depth, *see* Depth
ϵ_Q, *see* Queue
ext, *see* Orbit
$\mathfrak{A}$, *see* Σ-structure
gen, *see* Orbit
la, *see* Queue
lh, *see* Queue
$\leqq_n^\sharp$, *see* Gap order
$\prec_n^\sharp$, *see* Gap order
list, *see* LISP list
lt, *see* Queue
nil, *see* LISP list
$\not\models_\exists$, *see* Unsatisfiability
$1_{(\ldots)}$, *see* Upper boundary function
orb, *see* Orbit
ra, *see* Queue
rcons, *see* Queue
rh, *see* Queue
rt, *see* Queue
s, *see* Selector
$\models_\exists$, *see* Satisfiability
type, *see* Term algebra
$\leqq_n^{k;\sharp}$, *see* Tuple gap order
$\prec_n^{k;\sharp}$, *see* Tuple gap order
$\overline{1}_{(\ldots)}$, *see* Upper tuple boundary function
$\overline{0}_{(\ldots)}$, *see* Lower tuple boundary function
$(\cdot)^w$, *see* Weight function
$0_{(\ldots)}$, *see* Lower boundary function

$\forall_1$-formula, 6
$\forall_1$-fragment, 6
Arity, 5
Atom, 85
Atomic formula, 5

Bidirectional closure, 18
Block, 96
Boundary function, 129
  lower, 129
  on tuples, 131
    lower, 131
    upper, 131
  upper, 129
Boundary term, 129
  lower, 129
  upper, 129
Boundary tuple, 131
  lower, 131
  upper, 131

Closed term literal, 129
Closed tuple literal, 131
Cluster, 54
Cluster completeness, 55
Cluster completion, 55
Color, 95
Colored Tree, 32
Completeness
  of clusters, 55
  of equality
    of queues with integers, 92
    of term algebras with integers, 28
  of prefix, 98
  of sibling, 17
  of type, 15
Completion
  of clusters, 55
  of delineated gap order, 133
  of equality
    of queues with integers, 92
    of term algebras with integers, 28
  of gap order, 132
  of prefix, 98

of sibling, 17
of type, 15
Congruence closure, 18
Conjugacy, 87
Connected variable, 96
Constant, 5
of term algebras, 13
Constructor
of queue, 85
of term algebras, 13
Constructor term, 15
Convexity theory, 17
Counting constraint, 28
Cut length, 94

d_p, *see* Delimiter
DAG, *see* Directed acyclic graph
Delimiter
of queues with integers, 95
of queues with prefix and integers, 106
Delineated gap order completion, 133
Depth, 15
DGOC, *see* Delineated gap order completion
Directed acyclic graph, 17
of a formula, 17
Disjunctive normal form, 5
Domain, 6
of queue, 85
of term algebras, 13

$\exists_1$-formula, 6
$\exists_1$-fragment, 6
Equality completeness
of queues with integers, 92
of term algebras with integers, 28
Equality completion
of queues with integers, 92
of term algebras with integers, 28
Existential closure, 6
Expression, 6

Free occurrence, 5
Function symbol, 5

Gap order, 128
rigid, 129
stretchable, 129
tuple, 130
Gap order completion, 132
GOC, *see* Gap order completion
Ground formula, 5

Half-open term literal, 129
Half-open tuple literal, 131

Integer function, 175
Integer predicate, 174

KBO, *see* Knuth-Bendix order
Knuth-Bendix order, 126
on tuples, 130
suborders, 130
suborders, 128
lexicographic order, 128
precedence order, 128
weight order, 128

LCC, *see* Length constraint completion
Length configuration, 95
Length constraint completion
of queues with integers, 91
of term algebras with integers, 22
LISP list, 14
Literal, 5
closed, 129
half-open, 129
open, 129
tuple, 131

closed, 131
half-open, 131
open, 131
Lower boundary function, 129
Lower boundary term, 129
Lower boundary tuple, 131
Lower tuple boundary function, 131

Matrix, 6
Model, 6
Multi-sorted logic, 7

Nondeterminism, 8
Normal form
of queues, 88
of queues with integers, 92
with parameters, 111
of queues with prefix and integers, 97

Open term literal, 129
Open tuple literal, 131
Orbit, 87

Precedence, 126
Predicate symbol, 5
Prefix completeness, 98
Prefix completion, 98
Prenex form, 6
Presburger arithmetic, 8
Primitive form, 7
Primitive formula, *see* Primitive form
Primitive word, 87

Quantifier elimination, 7
Quantifier-free formula, 5
Quantifier-free fragment, 6
Quantifier-free theory, 6
Queue, 85
Queue with prefix, 97
Queue with prefix and integers, 97
Queues with integers, 90

Red-black tree, 33
Red-black trees, 32
Redex, 8
Reduct, 8
Refutational decision procedure, 7
Relativized length constraint completion
of queues with integers, 91
with parameters, 111
of red-black trees, 34
of term algebras with integers, 24
with parameters, 55
Rigid gap order, 129
Rigid tuple gap order, 131
RLCC, *see* Relativized length constraint completion

Satisfiability, 6
Satisfiable formula, 6
Selector
of queue, 85
of term algebras, 13
Selector term, 15
proper, 15
Sentence, 5
Sibling completeness, 17
Sibling completion, 17
Σ-structure, 6
Σ-term, 5
Solved form
KBO, 127
of queues, 88
of term algebras, 49
Sort, *see* Multi-sorted logic
Stretchable gap order, 129
Stretchable tuple gap order, 131

Strongly primitive word, 87
Strongly solved form
    of term algebras with integers, 56
Structure, *see* Σ-structure
    multi-sorted, 7
    of colored trees, 32
    of queues, 85
    of queues with integers, 90
    of queues with prefix, 97
    of queues with prefix and integers, 97
    of red-black trees, 33
    of term algebras, 13
    of term algebras with integers, 20
    of term algebras with KBO, 126
    of term algebras with KBO and integers, 130

Tag
    of queues with integers, 95
    of queues with prefix and integers, 108
Term, *see* Σ-term
    constructor, 15
    depth of, 15
    selector, 15
Term algebra, 13
Term algebra with integers, 20
Term algebra with KBO, 126
Term algebra with KBO and integers, 130
Tester, 13
Theory, 6
Tuple gap order
    rigid, 131
    stretchable, 131
Type completeness, 15
Type completion, 15

Unification closure, 18
Universal closure, 6
Unsatisfiability, 6
Unsatisfiable formula, 6
Upper boundary function, 129
Upper boundary term, 129
Upper boundary tuple, 131
Upper tuple boundary function, 131

Valid formula, 6
Validity, 6

Weight function, 126