Decision Procedures for Recursive Data Structures with Integer Constraints

Ting Zhang, Henny B. Sipma, Zohar Manna*

Computer Science Department
Stanford University
Stanford, CA 94305-9045
{tingz,sipma,zm}@theory.stanford.edu

Abstract. This paper is concerned with the integration of recursive data structures with Presburger arithmetic. The integrated theory includes a length function on data structures, thus providing a tight coupling between the two theories, and hence the general Nelson-Oppen combination method for decision procedures is not applicable to this theory, even for the quantifier-free case. We present four decision procedures for the integrated theory depending on whether the language has infinitely many constants and whether the theory has quantifiers. Our decision procedures for quantifier-free theories are based on Oppen's algorithm for acyclic recursive data structures with infinite atom domain.

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.

In this paper we consider the integration of Presburger arithmetic with an important subclass of recursively defined data structures known as *recursive data structures*. This class of structures satisfies the following 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

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134 and CCR-02-09237, by ARO grant DAAD19-01-1-0723, by ARPA/AF contracts F33615-00-C-1693 and F33615-99-C-3014, and by NAVY/ONR contract N00014-03-1-0939.

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 data (term) sort λ. Intuitively, the language is the set-theoretic union of the language of recursive data structures and the language of Presburger arithmetic plus the additional length function $|.| : \lambda \to \mathbb{Z}$. Formulae are formed from *data literals* and *integer literals* using logical connectives and quantifications. Data literals are exactly those literals in the theory of recursive data structures. Integer literals are those that can be built up from integer variables (including the length function applied to data terms), addition and the usual arithmetic functions and relations.

We present four decision procedures for different variants of the theory depending on whether the language has infinitely many atoms and whether the theory is quantifier-free. Our decision procedures for quantifier-free theories are based on Oppen's algorithm for acyclic recursive data structures with infinite atom domain [17]. When integer constraints in the input are absent, our decision procedures can be viewed as an extension of Oppen's original algorithm to cyclic structures and to the structures with finite atom domain.

Related Work Our component theories are both decidable. Presburger arithmetic was first shown to be decidable in 1929 by the quantifier elimination method [6]. Efficient algorithms were later discovered by Cooper et al [4, 18]. It is well-known that recursive data structures can be modeled as term algebras which were shown to be decidable by the quantifier elimination method [13, 11, 8]. Decision procedures for the quantifier-free theory of recursive data structures were discovered by Nelson, Oppen et al [15, 17, 5]. In [17] Oppen gave a linear algorithm for acyclic structures and a quadratic algorithm was given in [15] for cyclic structures. If the values of the selector functions on atoms are specified, then the problem is NP-complete [17].

A general combination method for decision procedures for quantifier-free theories was developed by Nelson and Oppen in 1979 [14]. However, this method is not applicable to the combination of our component theories. The method requires that component theories be loosely coupled, that is, have disjoint signatures, and are stably infinite[1]. The method is not applicable to tightly coupled theories such as single-sorted theories with shared signatures or, as in our case, multisorted theories with functions mapping elements in one sort to another.

The integration of Presburger arithmetic with recursive data structures was discussed by Bjørner [1] and an incomplete procedure was implemented in STeP (Stanford Temporal Prover) [2]. Zarba constructed decision procedures for the combined theory of sets and integers [22] and multisets and integers [21] by extending the Nelson-Oppen combination method.

Paper Organization Section 2 provides the preliminaries: it introduces the notation and terminology. Section 3 defines recursive data structures and ex-

[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.

plains Oppen's algorithm, the basis for our decision algorithm for quantifier-free theories. Section 4 introduces the combined theory of recursive data structures and Presburger arithmetic and outlines our approach for constructing the decision procedures. Sections 5-8 describe the four classes of decision procedures for the different variants of the theory in detail. Section 9 discusses complexity issues and Section 10 concludes with some ideas for future work. Because of space limitations most proofs have been omitted. They are available for reference in the extended version of this paper at `http://theory.stanford.edu/~tingz/papers/ijcar04_extended.pdf`.

2 Preliminaries

We assume the first-order syntatic notions of variables, parameters and quantifiers, and semantic notions of structures, satisfiability and validity as in [6].

A *signature* Σ is a set of *parameters* (*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.

An *atomic formula* (*atom*) 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 (equality is treated as a binary predicate symbol). A *literal* is an atomic formula or its negation. A *ground formula* is a formula with no variables. A variable occurs *free* in a formula if it is not in the scope of a quantifier. A *sentence* is a formula in which no variable occurs free. A formula without quantifiers is called *quantifier-free*. Every quantifier-free formula can be put into *disjunctive normal form*, that is, a disjunction of conjunctions of literals.

A Σ*-structure* (or Σ*-interpretation*) $\mathfrak{A}$ 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 usually denote $\mathfrak{A}$ by $\langle A; \Sigma \rangle$ which is called the *signature* of $\mathfrak{A}$. We use Gothic letters (like $\mathfrak{A}$) for structures and Roman letters (like A) for the underlying domain. A *variable valuation* (or *variable assignment*) ν (w.r.t. $\mathfrak{A}$) is a function that assigns each variable an element of A. The truth value of a formula is determined when an interpretation and a variable assignment is given.

A *term* algebra of Σ with basis $\mathcal{X}$ is the structure $\mathfrak{A}$ whose domain is $\mathcal{T}(\Sigma, \mathcal{X})$ and for any n-place function symbol $f \in \Sigma$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$, $f^{\mathfrak{A}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$. We assume that Σ does not contain any predicate symbols except equality.

Presburger arithmetic is the first-order theory of addition in the arithmetic of integers. The corresponding structure is denoted by $\mathfrak{A}_{\mathbb{Z}} = \langle \mathbb{Z}; 0, +, < \rangle$.

A formula φ is *satisfiable* (or *consistent*) if it is true under some variable valuation; it is *unsatisfiable* (or *inconsistent*) otherwise. A formula φ is *valid* if it is true under every variable valuation. A formula φ is valid if and only if $\neg\varphi$ is unsatisfiable. We say that $\mathfrak{A}$ 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 formulae. A *theory* T is a set of sentences that is closed under logical implication, that is, if $T \models \varphi$, then $\varphi \in T$. By a *theory of structure* $\mathfrak{A}$, written $\mathsf{Th}(\mathfrak{A})$, we shall mean the set of all valid sentences in $\mathfrak{A}$. We write $\mathsf{Th}^{\forall}(\mathfrak{A})$ for the quantifier-free fragment of $\mathsf{Th}(\mathfrak{A})$.

In this paper all decision procedures for quantifier-free theories are *refutation-based*; to determine the validity of a formula φ it suffices to 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, an *input formula* always refers to a conjunction of literals.

3 Recursive Data Structures and Oppen's Algorithm

We present a general language of recursive data structures. For simplicity, we do not distinguish syntactic terms in the language from semantic terms in the corresponding interpretation. The meaning should be clear from the context.

Definition 1. *A recursive data structure* $\mathfrak{A}_\lambda : \langle \lambda; \mathcal{A}, \mathcal{C}, \mathcal{S}, \mathcal{T} \rangle$ *consists of*

1. *λ: The data domain, which consists of all terms built up from constants by applying constructors. Elements in λ are called* λ-terms *(or* data terms*).*
2. *$\mathcal{A}$: A set of atoms (constants): a, b, c, ...*
3. *$\mathcal{C}$: A finite set of constructors: α, β, γ, ... The arity of α is denoted by $\mathsf{ar}(\alpha)$. We say that an object is α-typed (or an α-term) if its outmost constructor is α.*
4. *$\mathcal{S}$: A finite set of selectors. For each constructor α with arity $k > 0$, there are k selectors $\mathsf{s}^\alpha_1, \ldots, \mathsf{s}^\alpha_k$ in $\mathcal{S}$. We call s^α_i $(1 \le i \le k)$ the i^{th} α-selector. For a term x, $\mathsf{s}^\alpha_i(x)$ returns the i^{th} component of x if x is an α-term and x itself otherwise.*
5. *$\mathcal{T}$: A finite 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. In addition there is a special tester Is_A such that $\mathsf{Is}_A(x)$ is true if and only if x is an atom. Note that there is no need for individual atom testers as $x = a$ serves as $\mathsf{Is}_a(x)$.*

The theory of recursive data structures is essentially the theory of term algebras (with the empty variable basis) which is axiomatizable as follows.

Proposition 1 (Axiomatization of Recursive Data Structures [8]). *Let $\bar{z}_\alpha$ abbreviate $z_1, \ldots, z_{\mathsf{ar}(\alpha)}$. The following formula schemes, in which variables are implicitly universally quantified over λ, axiomatize $\mathsf{Th}(\mathfrak{A}_\lambda)$.*

A. $t(x) \neq x$, *if t is built solely by constructors and t properly contains x.*
B. $a \neq b$, $a \neq \alpha(x_1 \ldots, x_{\mathsf{ar}(\alpha)})$, *and* $\alpha(x_1 \ldots, x_{\mathsf{ar}(\alpha)}) \neq \beta(y_1, \ldots, y_{\mathsf{ar}(\beta)})$, *if a and b are distinct atoms and if α and β are distinct constructors.*
C. $\alpha(x_1, \ldots, x_{\mathsf{ar}(\alpha)}) = \alpha(y_1, \ldots, y_{\mathsf{ar}(\alpha)}) \rightarrow \bigwedge_{1 \le i \le \mathsf{ar}(\alpha)} x_i = y_i$.
D. $\mathsf{Is}_\alpha(x) \leftrightarrow \exists\ \bar{z}_\alpha \alpha(\bar{z}_\alpha) = x$, $\mathsf{Is}_A(x) \leftrightarrow \bigwedge_{\alpha \in \mathcal{C}} \neg \mathsf{Is}_\alpha(x)$.

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

In general, selectors and testers can be defined by constructors and vice versa. One direction has been shown by (D) and (E), which are pure definitional axioms.

Example 1. Consider the LISP list structure $\mathfrak{A}_{\mathsf{List}} = \langle \mathsf{List};\ \mathsf{cons}, \mathsf{car}, \mathsf{cdr}\rangle$ where *list* denotes the domain, *cons* is the 2-place constructor (pairing function) and *car* and *cdr* are the corresponding left and right selectors (projectors) respectively. Let $\{\mathsf{car}, \mathsf{cdr}\}^+$ denote any nonempty sequence of car and cdr. The axiom schemas in Proposition 1 reduce to the following.

$(i)\ \mathsf{Is}_A(x) \leftrightarrow \neg\mathsf{Is}_{\mathsf{cons}}(x), \quad (ii)\ \mathsf{car}(\mathsf{cons}(x,y)) = x, \quad (iii)\ \mathsf{cdr}(\mathsf{cons}(x,y)) = y,$

$(iv)\ \mathsf{Is}_A(x) \leftrightarrow \{\mathsf{car}, \mathsf{cdr}\}^+(x) = x, \quad (v)\ \mathsf{Is}_{\mathsf{cons}}(x) \leftrightarrow \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x)) = x.$

Decision Procedures for Recursive Data Structures

In [17] Oppen presented a decision procedure for acyclic data structures $\mathfrak{A}_\lambda$. The basic idea of the decision procedure is to generate all equalities between terms implied by asserted equalities in the formula and check for inconsistencies with the asserted disequalities in the formula.

The decision procedure relies on the fact that $\mathsf{Th}(\mathfrak{A}_\lambda)$ is convex.

Definition 2 (Convexity). *A theory is* convex *if whenever a conjunction of literals implies a disjunction of atoms, it also implies one of the disjuncts.*

Let Φ be a conjunction of literals and Ψ a disjunction of equalities. The convexity of $\mathsf{Th}(\mathfrak{A}_\lambda)$ can be rephrased as follows: if none of the disjuncts in Ψ is implied by Φ, then $\neg\Psi$ is Φ-satisfiable. Hence Φ is satisfiable if and only if for any terms s and t, whenever $s \neq t \in \Phi$, $\Phi \not\models s = t$. The idea of Oppen's algorithm is to discover all logically implied equalities (between terms in Φ) using the DAG representation and the bidirectional closure algorithm, which we introduce below.

Definition 3 (DAG Representation). *A term t can be represented by a tree T_t such that (i) if t is a constant or variable, 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 having the root labeled by t and having $T_{t_1}, \ldots, T_{t_k}$ as its subtrees. A directed acyclic graph (DAG) G_t of t is obtained from T_t by "factoring out" the common subtrees (subterms).*

For a vertex u, let $\pi(u)$ denote the label, $\delta(u)$ the outgoing degree and $u[i]$ $(1 \leq i \leq \delta(u))$ the i^{th} successor of u. The *DAG of a formula* is the DAG representing all terms in the formula. For example, Figure 1 shows the DAG for $\mathsf{cons}(y,z) = \mathsf{cons}(\mathsf{cdr}(x), z) \wedge \mathsf{cons}(\mathsf{car}(x), y) \neq x$ under the assumption that x is not an atom.

Definition 4 (Bidirectional Closure). *Let R be a binary relation on a DAG and let u, v be any two vertices such that $\delta(u) = \delta(v)$. 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 $\pi(u) = \pi(v)$ implies $(u[i], v[i]) \in R'$ for $1 \leq i \leq \delta(u)$. 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 $(u[i], v[i]) \in R'$ $(1 \leq i \leq \delta(u))$ implies $\pi(u) = \pi(v)$. If R' is both unification and congruence closed (w.r.t. R), we call it the* bidirectional *closure, denoted by $R\!\Updownarrow$.*

Let R be the set of all pairs asserted equal in Φ. It has been shown that $R\Updownarrow$ represents all equalities logically implied by Φ [17]. Therefore Φ is unsatisfiable if and only if there exists t and s such that $t \neq s \in \Phi$ and $(t, s) \in R\Updownarrow$.

Algorithm 1 (Oppen's Decision Procedure for Acyclic $\mathfrak{A}_\lambda$ [17]).
Input: $\Phi : q_1 = r_1 \wedge \ldots \wedge q_k = r_k \ \wedge \ s_1 \neq t_1 \wedge \ldots \wedge s_l \neq t_l$

1. *Construct the DAG G of Φ.*
2. *Compute the bidirectional closure $R\Updownarrow$ of $R = \{(q_i, r_i) \mid 1 \leq i \leq k\}$.*
3. *Return* FAIL *if $\exists i(s_i, t_i) \in R\Updownarrow$; return* SUCCESS *otherwise.*

In our setting $\mathfrak{A}_\lambda$ is cyclic 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 problem is NP-complete [15]. 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 immediately inside selector functions before applying Algorithm 1.

Definition 5 (Type Completion). *Φ' is a type completion of Φ if Φ' is obtained from Φ by adding tester predicates such that for any term $\mathsf{s}(t)$ either $\mathsf{Is}_\alpha(t)$ (for some constructor α) or $\mathsf{Is}_A(t)$ is present in Φ'.*

Example 2. A possible type completion for $y = \mathsf{car}(\mathsf{cdr}(x))$ is $y = \mathsf{car}(\mathsf{cdr}(x)) \wedge \mathsf{Is}_{\mathsf{cons}}(x) \wedge \mathsf{Is}_A(\mathsf{cdr}(x))$.

A type completion Φ' is *compatible* with Φ if the satisfiability of Φ implies that Φ' is satisfiable and if any solution of Φ' is a solution of Φ. Obviously Φ is satisfiable if and only if it has a satisfiable compatible completion. This leads to the following nondeterministic algorithm that relies on the successful guess of a satisfiable compatible completion if such completion exists.

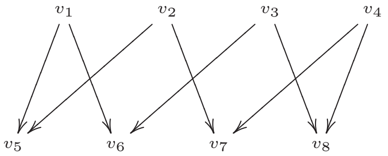
Algorithm 2 (The Decision Procedure for Cyclic $\mathfrak{A}_\lambda$). *Input: Φ.*

1. *Guess a type completion Φ' of Φ and simplify selector terms accordingly.*
2. *Call Algorithm 1 on Φ'.*

Example 3. Figure 1 shows the DAG representation of

$$\mathsf{cons}(y, z) = \mathsf{cons}(\mathsf{cdr}(x), z) \wedge \mathsf{cons}(\mathsf{car}(x), y) \neq x \tag{1}$$

under the guess $\mathsf{Is}_{\mathsf{cons}}(x)$. Initially $R = \{(v_3, v_4)\}$ as v_3 and v_4 are asserted equal in (1). (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 $R\Updownarrow = \{(v_1, v_2), (v_3, v_4), (v_6, v_7)\}$. Obviously this branch fails as $v_1 \neq v_2$ is asserted by (1). The remaining branch (with presence of $\mathsf{Is}_A(x)$) simplifies to $\mathsf{Is}_A(x) \wedge x = y$ which is clearly satisfiable, and therefore so is (1).



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

Fig. 1. The DAG of (1) under the assumption $\mathsf{Is}_{\mathsf{cons}}(x)$.

Note that the correctness of both Algorithm 1 and 2 relies on the (implicit) assumption that the atom domain is infinite, since otherwise the theory is not convex. As a counter example, for the structure $\mathfrak{A}_\lambda$ with only two atoms a and b, we have $\mathsf{Is}_A(x) \models x = a \vee x = b$, but neither $\mathsf{Is}_A(x) \models x = a$ nor $\mathsf{Is}_A(x) \models x = b$. We shall see (in Section 6) that our algorithm extends Oppen's original algorithm to structures with finite atom domain.

4 The Approach for the Integrated Theory

In this section we describe the different variants of the integrated theory and outline our approach for constructing the decision procedures for each of these variants. The details of each of these four decision procedures are presented in the next four sections.

Definition 6. *The structure of the integrated theory is* $\mathfrak{B}_\lambda = (\mathfrak{A}_\lambda; \mathfrak{A}_\mathbb{Z}; |.| : \lambda \to \mathbb{Z})$ *where* $\mathfrak{A}_\lambda$ *is a recursive data structure,* $\mathfrak{A}_\mathbb{Z}$ *is Presburger arithmetic, and* $|.|$ *denotes the length function defined recursively by: (i) for any atom* a, $|a| = 1$, *and (ii) for a term* $\alpha(t_1, \ldots, t_k)$, $|\alpha(t_1, \ldots, t_k)| = \sum_{i=1}^{k} |t_i|$.

Notice that we have chosen a length function that does not count the intermediate vertices. However, the algorithms can easily be modified for an alternative length function. We view terms of the form $|t|$ (where t is a non-ground λ-term) as (*generalized*) integer variables. If ν is an assignment of data variables, let $|\nu|$ denote the corresponding assignment of generalized integer variables. From now on, we use $\Phi_\mathbb{Z}$ (resp. Φ_λ) to denote a conjunction of integer literals (resp. a conjunction of data literals). Whenever it is clear from context, we write $\mathfrak{B}$ for $\mathfrak{B}_\lambda$. By $\mathfrak{B}^\omega$, $\mathfrak{B}^{<\omega}$ and $\mathfrak{B}^{=k}$ ($k > 0$) we denote the structures with infinitely many atoms, with finitely many atoms and with exactly k atoms, respectively.

As was mentioned before, the general purpose combination method in [14] is not directly applicable due to the presence of the length function. The following example illustrates the problem.

Example 4. Consider $\mathfrak{B}^\omega_{\mathsf{List}}$. The constraints $\Phi_\lambda : x = \mathsf{cons}(\mathsf{car}(y), y)$ and $\Phi_\mathbb{Z} : |x| < 2|\mathsf{car}(x)|$ are clearly satisfiable, respectively, in $\mathfrak{A}_{\mathsf{List}}$ and $\mathfrak{A}_\mathbb{Z}$. However, since Φ_λ implies that $\mathsf{car}(x) = \mathsf{car}(y)$, x contains two copies of $\mathsf{car}(y)$ and so its length should be at least two times the length of $\mathsf{car}(x)$. Therefore, $\Phi_\mathbb{Z} \wedge \Phi_\lambda$ is unsatisfiable.

A simple but crucial observation is that constraints of data structures impose "hidden" constraints on the lengths of those structures. Here we distinguish two types of integer constraints; "external integer constraints" that occur explicitly in input formulae, and "internal integer constraints" that are "induced" by satisfying assignments for the pure data structure constraints. In the following we will show that if we can express sound and complete length constraints (in the sense defined below) in Presburger arithmetic, we can derive decision procedures by utilizing the decision procedures for Presburger arithmetic and for recursive data structures.

Definition 7 (Induced Length Constraint). *A* length constraint Φ_Δ *with respect to* Φ_λ *is a Presburger formula in which only generalized integer variables occur free.* Φ_Δ *is* sound *if for any satisfying assignment* ν_λ *of* Φ_λ, $|\nu_\lambda|$ *is a satisfying assignment for* Φ_Δ. Φ_Δ *is* complete *if, whenever* Φ_λ *is satisfiable, for any satisfying assignment* ν_Δ *of* Φ_Δ *there exists a satisfying assignment* ν_λ *of* Φ_λ *such that* $|\nu_\lambda| = \nu_\Delta$. *We also say that* Φ_Δ *is* realizable *w.r.t.* Φ_λ *provided that* Φ_λ *is satisfiable. We say that* Φ_Δ *is* induced *by* Φ_λ *if* Φ_Δ *is both sound and complete.*

Example 5. Consider the formula $\Phi : \mathsf{cons}(x, y) = z$. The length constraint $|x| < |z| \wedge |y| < |z|$ is sound but it is not complete for Φ, as the integer assignment $\nu_\Delta : \{|x| = 3, |y| = 3, |z| = 4\}$ can not be realized. On the other hand, $|x| + |y| = |z| \wedge |x| > 5 \wedge |y| > 0$ is complete for Φ, but it is not sound because it does not satisfy the data assignment $\nu_\lambda : \{x = a, y = a, z = \mathsf{cons}(a, a)\}$. Finally, $|x| + |y| = |z| \wedge |x| > 0 \wedge |y| > 0$ is both sound and complete, and hence is the induced length constraint of Φ.

Main Theorem. *Let* Φ *be in the form* $\Phi_\mathbb{Z} \wedge \Phi_\lambda$. *Let* Φ_Δ *be the induced length constraint with respect to* Φ_λ. *Then* Φ *is satisfiable in* $\mathfrak{B}$ *if and only if* $\Phi_\Delta \wedge \Phi_\mathbb{Z}$ *is satisfiable in* $\mathfrak{A}_\mathbb{Z}$ *and* Φ_λ *is satisfiable in* $\mathfrak{A}_\lambda$.

Proof. ("$\Rightarrow$") Suppose that Φ is satisfiable and let ν be a satisfying assignment. ν divides into two disjoint parts: ν_λ for data variables and $\nu_\mathbb{Z}$ for integer variables. Φ_λ is obviously satisfiable under ν_λ. Let $\nu_\Delta = |\nu_\lambda|$. By soundness of Φ_Δ we know that ν_Δ satisfies Φ_Δ. Hence $\Phi_\Delta \wedge \Phi_\mathbb{Z}$ is satisfiable under the joint assignment of ν_Δ and $\nu_\mathbb{Z}$.
("$\Leftarrow$") Suppose that both $\Phi_\Delta \wedge \Phi_\mathbb{Z}$ and Φ_λ are satisfiable. Let ν be a satisfying assignment for $\Phi_\Delta \wedge \Phi_\mathbb{Z}$. ν divides into two disjoint parts: $\nu_\mathbb{Z}$ for pure integer variables and ν_Δ for the generalized integer variables. By the completeness of Φ_Δ, ν_Δ can be realized by a satisfying assignment ν_λ such that $\nu_\Delta = |\nu_\lambda|$. Hence the joint assignment $\nu_\lambda \cup \nu_\mathbb{Z} \cup \nu_\Delta$ will satisfy Φ. □

By the main theorem the decision problem for quantifier-free theories reduces to computing the induced length constraints in Presburger arithmetic. In the next two sections we show that this is possible for the two quantifier-free variants of our theory.

5 The Decision Procedure for $\mathsf{Th}^\forall(\mathfrak{B}^\omega)$

The first and easiest of the four variants considered is the quantifier-free combination with an infinite atom domain. In the structure $\mathfrak{B}^\omega$ the induced length constraints of a formula can be derived directly from the DAG for the formula. Before we present the algorithm we define the following predicates on terms in the DAG:

$$\begin{array}{ll} \mathsf{Tree}(t) & : \exists x_1, \ldots, x_n \geq 0 \;\; (\; |t| = (\sum_{i=1}^{n}(d_i - 1)x_i) + 1) \\ \mathsf{Node}^{\alpha}(t, \bar{t}_\alpha) & : |t| = \sum_{i=1}^{\delta(\alpha)} |t_i| \\ \mathsf{Tree}^{\alpha}(t) & : \exists \bar{t}_\alpha \left(\mathsf{Node}^{\alpha}(t, \bar{t}_\alpha) \wedge \bigwedge_{i=1}^{\delta(\alpha)} \mathsf{Tree}(t_i) \right) \end{array}$$

where $\bar{t}_\alpha$ stands for $t_1, \ldots, t_{\delta(\alpha)}$ and and $d_1, \ldots, d_n$ are the distinct arities of the constructors. The predicate $Tree(t)$ is true iff $|t|$ 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 $d-1$. The second predicate forces the length of an α-typed node with known children to be the sum of the lengths of its children. The last predicate states the length constraint for an α-typed leaf. With these predicates the construction of the induced length constraint is given by the following algorithm.

Algorithm 3 (Construction of Φ_Δ in $\mathfrak{B}^\omega$). *Let Φ_λ be a (type-complete) data constraint, G_λ the DAG of Φ_λ and $R\Updownarrow$ the bidirectional closure obtained by Algorithm 1. Initially set $\Phi_\Delta = \emptyset$. For each term t add the following to Φ_Δ.*

- $|t| = 1$, *if t is an atom;*
- $|t| = |s|$, *if $(t, s) \in R\Updownarrow$.*
- $\mathsf{Tree}(t)$ *if t is an untyped leaf vertex*
- $\mathsf{Node}^{\alpha}(t, \bar{t}_\alpha)$ *if t is an α-typed vertex with children $\bar{t}_\alpha$.*
- $\mathsf{Tree}^{\alpha}(t)$ *if t is an α-typed leaf vertex.*

Proposition 2. *Φ_Δ obtained by Algorithm 3 is expressible in a quantifier-free Presburger formula linear in the size of Φ.*

Theorem 1. *Φ_Δ obtained by Algorithm 3 is the induced length constraint of Φ_λ.*

Algorithm 4 (Decision Procedure for $\mathsf{Th}^{\forall}(\mathfrak{B}^\omega)$). *Input: $\Phi_\lambda \wedge \Phi_{\mathbb{Z}}$.*

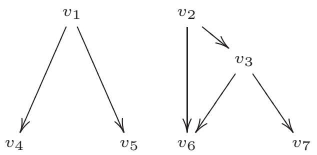
1. *Guess a type completion Φ'_λ of Φ_λ.*
2. *Call Algorithm 1 on Φ'_λ.*
   - *Return* FAIL *if Φ'_λ is unsatisfiable; continue otherwise.*
3. *Construct Φ_Δ from G'_λ using Algorithm 3.*
   - *Return* SUCCESS *if Φ_Δ is satisfiable and $\Phi_{\mathbb{Z}}$ is satisfiable;*
   - *Return* FAIL *otherwise.*

The correctness of the algorithm follows from Main Theorem and Theorem 1.

Example 6. Figure 2 shows the DAG of

$$x = \mathsf{cons}(\mathsf{car}(y), y) \wedge |\mathsf{cons}(\mathsf{car}(y), y)| < 2|\mathsf{car}(x)|, \tag{2}$$

assuming that y is not an atom. The computed $R\Updownarrow$ is $\{(v_1, v_2), (v_3, v_5), (v_4, v_6)\}$. By Algorithm 3 Φ_Δ includes the conjunction $|\mathsf{cons}(\mathsf{car}(y), y)| = |\mathsf{car}(y)| + |y| \wedge |\mathsf{car}(x)| = |\mathsf{car}(y)| \wedge |\mathsf{cdr}(x)| = |y|$ which implies $|\mathsf{cons}(\mathsf{car}(y), y)| \geq 2|\mathsf{car}(x)|$, contradicting $|\mathsf{cons}(\mathsf{car}(y), y)| < 2|\mathsf{car}(x)|$. If y is an atom, v_3, v_6, v_7 are merged. But still we have $|\mathsf{cons}(\mathsf{car}(y), y)| \geq 2|\mathsf{car}(x)|$. Therefore (2) is unsatisfiable.



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

Fig. 2. The DAG of (2) under the assumption $\mathsf{Is}_{\mathsf{cons}}(y)$

6 The Decision Procedures for $\mathsf{Th}^{\forall}(\mathfrak{B}^{=k})$ and $\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega})$

Algorithm 3 cannot be used to construct the induced length constraints in structures with a finite number of atoms, $\mathfrak{B}^{=k}$, as illustrated by the following example.

Example 7. Consider $\mathfrak{B}^{=1}_{\mathsf{List}}$ (with atom a). The constraint

$$|x| = 3 \wedge \mathsf{Is}_A(y) \wedge x \neq \mathsf{cons}(\mathsf{cons}(y, y), y) \wedge x \neq \mathsf{cons}(y, \mathsf{cons}(y, y)) \tag{3}$$

is unsatisfiable while Φ_Δ obtained by Algorithm 3 is

$$|y| = 1 \wedge |\mathsf{cons}(y, y)| = 2 \wedge |\mathsf{cons}(\mathsf{cons}(y, y), y)| = 3 \wedge |\mathsf{cons}(y, \mathsf{cons}(y, y)| = 3$$

which is obviously satisfiable together with $|x| = 3$.

The reason is that if the atom domain is finite 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 8 (Counting Constraint). *A* counting constraint *is a predicate* $\mathsf{CNT}^{\alpha}_{k,n}(x)$ *that is* true *if and only if there are at least $n+1$ different α-terms of length x in the language with exactly $k > 0$ distinct atoms.* $\mathsf{CNT}_{k,n}(x)$ *is similarly defined with α-terms replaced by λ-terms.*

The following two *monotonicity* properties are easily proven: for any $l \geq k > 0$ and $m \geq n > 0$, (i) $\mathsf{CNT}^{\alpha}_{k,n}(x) \rightarrow \mathsf{CNT}^{\alpha}_{l,n}(x)$ and (ii) $\mathsf{CNT}^{\alpha}_{k,m}(x) \rightarrow \mathsf{CNT}^{\alpha}_{k,n}(x)$.

Example 8. For $\mathfrak{B}^{=1}_{\mathsf{List}}$, $\mathsf{CNT}^{\mathsf{cons}}_{1,n}(x)$ is $x \geq m$ 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 leaf vertices.

In general we have the following result.

Proposition 3. $\mathsf{CNT}^{\alpha}_{k,n}(x)$ *is expressible by a quantifier-free Presburger formula that can be computed in time $O(n)$.*

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

Definition 9 (Equality Completion). *An* equality completion Φ'_λ *of* Φ_λ *is formula obtained from* Φ_λ *such that for any two terms u and v in* Φ_λ *either $u = v$ or $u \neq v$, and either $|u| = |v|$ or $|u| \neq |v|$ are in* Φ'_λ*.*

As before we present a nondeterministic algorithm; Φ is satisfiable if and only if at least one of the compatible (type and equality) completions of Φ is. By $\mathsf{NEQ}_n(x_0, x_1, \ldots, x_n)$ we shall mean that $x_0, \ldots, x_n$ have the same length but are pairwise distinct.

Algorithm 5 (Construction of Φ_Δ in $\mathfrak{B}^{=k}$).
Input: Φ_λ (type and equality complete), G_λ and $R\Updownarrow$.

1. *Call Algorithm 3 to obtain Φ_Δ.*
2. *Add $\mathsf{CNT}^{\alpha}_{k,n}(|t|)$ to Φ_Δ for each t occurring in $\mathsf{NEQ}_n(t, t_1, \ldots, t_n)$.*

Proposition 4. *Φ_Δ obtained by Algorithm 5 is expressible in a quantifier-free Presburger formula and the size of such a formula is linear in the size of Φ.*

Theorem 2. *Φ_Δ obtained by Algorithm 5 is the induced length constraint of Φ_λ.*

Algorithm 6 (Decision Procedure for $\mathsf{Th}^{\forall}(\mathfrak{B}^{=k})$). *Input : $\Phi_\lambda \wedge \Phi_{\mathbb{Z}}$.*

1. *Guess a type and equality completion Φ'_λ of Φ_λ.*
2. *Call Algorithm 1 on Φ'_λ.*
   – *Return* FAIL *if Φ'_λ is unsatisfiable; continue otherwise.*
3. *Construct Φ_Δ from G'_λ using Algorithm 5.*
   – *Return* SUCCESS *if Φ_Δ is satisfiable and $\Phi_{\mathbb{Z}}$ is satisfiable.*
   – *Return* FAIL *otherwise.*

The correctness of Algorithm 6 follows from the Main Theorem and Theorem 2. Notice that, when $\Phi_{\mathbb{Z}}$ is empty, the algorithm can be viewed as an extension of Oppen's original algorithm for structures with finite atom domain.

Example 9. Let us return to Example 7. Constraint (3) has exactly one compatible completion, namely $\mathsf{NEQ}_3(x, \mathsf{cons}(\mathsf{cons}(y, y), y), \mathsf{cons}(y, \mathsf{cons}(y, y)))$. This results in the counting constraint $|x| \geq 4$, contradicting $|x| = 3$.

Algorithm 4 is also a decision procedure for $\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega})$ according to the following theorem.

Theorem 3. $\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega}) = \mathsf{Th}^{\forall}(\mathfrak{B}^{\omega})$ *in the languages with no constants.*

7 The Decision Procedure for $\mathsf{Th}(\mathfrak{B}^{\omega})$

In this section and the next one we show the decidability of $\mathsf{Th}(\mathfrak{B}^{\omega})$, $\mathsf{Th}(\mathfrak{B}^{=k})$ and $\mathsf{Th}(\mathfrak{B}^{<\omega})$ by extending the quantifier elimination procedure for the theory of term algebras [13, 11, 8] to our combined theory. A structure is said to "admit quantifier elimination" if any formula can be equivalently (and effectively) transformed into a quantifier-free formula. We demonstrate a procedure by which a sentence of $\mathfrak{B}$ is reduced to a ground quantifier-free formula (i.e., with no variable occurrence) whose validity can be easily checked. Our elimination procedures induce decision procedures for quantifier-free theories as satisfiability of a quantifier-free formula is the same as validity of its existential closure. However, unfortunately, the complexity lower bound of the theory of term algebras is *non-elementary* [3, 7].

It is well-known that eliminating arbitrary quantifiers reduces to eliminating existential quantifiers from formulae in the form $\exists x(A_1(x) \wedge \ldots \wedge A_n(x))$, where $A_i(x)$ ($1 \leq i \leq n$) are literals [8]. We may also assume that these literals are not of the form $x = t$ as $\exists x(x = t \wedge \varphi(x, \bar{y}))$ simplifies to $\varphi(t, \bar{y})$ if x does not occur in t, and to false by Axiom (A) if t properly contains x. For our two-sorted language, we need to show how to eliminate integer quantifiers as well as data quantifiers. As before, we present nondeterministic algorithms, but now a formula is valid if and only if it is true in every existential branch. It is easy to see the soundness of transformations in the two elimination procedures presented in this section and the next one. We leave the termination proofs to the extended version of this paper.

Eliminate Quantifiers on Integer Variables

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

$$\exists x : \mathbb{Z}\ (\Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z}) \wedge \Phi_{\lambda}(\bar{z})), \tag{4}$$

where $\bar{y}$ (resp. $\bar{z}$) denotes a sequence of integer variables (resp. data variables). Since $\Phi_{\lambda}(\bar{z})$ does not contain x, we can move it out of the scope of $\exists x$, and obtain

$$\exists x : \mathbb{Z}\ (\Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z})) \wedge \Phi_{\lambda}(\bar{z}). \tag{5}$$

Notice that in $\Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z})$, $\bar{z}$ only occurs inside generalized integer variables of the form $|t|$. Therefore $\exists x : \mathbb{Z}(\Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z}))$ is essentially a Presburger formula and we can proceed to remove the quantifier using Cooper's method [4].

Eliminate Quantifiers on Data Variables

We assume that formulae with existential quantifiers on data variables are in the form

$$\exists x : \lambda\ (\Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z}) \wedge \Phi_{\lambda}(x, \bar{z})), \tag{6}$$

where $\bar{y}$ (resp. $\bar{z}$) denotes a sequence of integer variables (resp. data variables). As before in $\Phi_{\mathbb{Z}}(\bar{y}, x, \bar{z})$, $x, \bar{z}$ only occur inside integer terms. We may assume that (6) does not contain constructors (see Section 3). First we make sure that x does not appear properly inside any terms. Suppose otherwise that $s(x)$ occurs for some selector s. As in [8], by guessing that x is α-typed, (6) becomes

$$\exists x, x_1, \ldots, x_{\mathsf{ar}(\alpha)} : \lambda \left[\mathsf{Is}_{\alpha}(x) \wedge \bigwedge_{1 \leq i \leq \mathsf{ar}(\alpha)} s_i^{\alpha}(x) = x_i \wedge \Phi_{\lambda}(x, \bar{z}) \wedge \Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z})\right]. \tag{7}$$

We simplify (7) as follows: replace $x \neq t$ by $\bigvee_{1 \leq i \leq \mathsf{ar}(\alpha)} s_i^{\alpha}(t) \neq x_i \vee \neg \mathsf{Is}_{\alpha}(t)$ (which will cause disjunctive splittings), replace $s_i^{\alpha}(x)$ by x_i, replace $s_j^{\beta}(x)$ (for $\alpha \not\equiv \beta$) by x and replace $|x|$ by $\Sigma_{i=1}^{\mathsf{ar}(\alpha)} |x_i|$. After the simplification no x occurs in Φ_{λ} and $\Phi_{\mathbb{Z}}$, so we can remove $\bigwedge_{1 \leq i \leq \mathsf{ar}(\alpha)} s_i^{\alpha}(x) = x_i$, $\mathsf{Is}_{\alpha}(x)$ and the quantifier $\exists x$. Although the transformation from (6) to (7) introduces new quantified variables, those new variables appear in smaller terms (compared with x). Hence the process will eventually terminate, producing formulae of the form

$$\exists x : \lambda\ \big(\bigwedge_{i<n} x \neq t_i \wedge \Phi_{\lambda}(\bar{z}) \wedge \Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z})\big), \tag{8}$$

where x does not appear in $\Phi_\lambda(\bar{z})$ and t_i $(i < n)$. (8) says that there exists an x which is not equal to any terms of $t_0, \ldots t_n$ and whose length is constrained by $\Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z})$. But in $\mathfrak{B}^\omega$ for any $n > 0$ there are infinitely many terms of length n. It follows that $\bigwedge_{i<n} x \neq t_i$ can be ignored and thus (8) is equivalent to

$$\exists n : \mathbb{Z} \left[\mathsf{Tree}(x)\big[|x|/n\big] \wedge \Phi_\lambda(\bar{z}) \wedge \Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z})\big[|x|/n\big]\right], \tag{9}$$

where $\varphi[|x|/n]$ stands for the formula obtained from φ by substituting all occurrences of $|x|$ for n. Now (9) can be handled by the elimination procedure for integer quantifiers. Note that if $\mathsf{Is}_\alpha(x)$ is in (8) we use $\mathsf{Tree}^\alpha(x)$ instead of $\mathsf{Tree}(x)$ in (9). Also, if we had guessed that x is an atom, we would have directly arrived at (8) by substituting x for $s(x)$, and then to (9) with n instantiated to 1.

8 The Decision Procedures for $\mathsf{Th}(\mathfrak{B}^{=k})$ and $\mathsf{Th}(\mathfrak{B}^{<\omega})$

In $\mathfrak{B}^{=k}$ the reduction from (8) to (9) is not sound for the same reason as in Section 6. However, the technique (of adding counting constraints) is still applicable here. We first introduce some new notations.

Definition 10 (Partitioning Formula). *Let x be a data variable and S be the set of data terms $\{t_1, \ldots, t_n\}$. Let P be a partition of S and $Q \subseteq P$. By a partitioning formula, written $\mathsf{PART}[x, S, P, Q]$, we shall mean*

$$\mathsf{NE}_\lambda[x, S] \wedge \underbrace{\mathsf{NE}'_\lambda[S, P] \wedge \mathsf{EQ}_\lambda[S, P] \wedge \mathsf{NE}_{\mathbb{Z}}[x, S, Q] \wedge \mathsf{EQ}_{\mathbb{Z}}[x, S, Q]}_{\mathsf{PART}'[x,S,P,Q]}, \tag{10}$$

where $\mathsf{NE}_\lambda[x, S] \equiv \bigwedge_{t \in S} x \neq t,$ $\quad \mathsf{NE}_{\mathbb{Z}}[x, S, Q] \equiv \bigwedge_{t \in S_i \notin Q} |x| \neq |t|,$

$\mathsf{NE}'_\lambda[S, P] \equiv \bigwedge_{t \in S_i \in P, t' \in S_j \in P, S_i \neq S_j} t \neq t',$ $\quad \mathsf{EQ}_{\mathbb{Z}}[x, S, Q] \equiv \bigwedge_{t \in S_i \in Q} |x| = |t|,$

$\mathsf{EQ}_\lambda[S, P] \equiv \bigwedge_{t, t' \in S_i \in P} t = t'.$

Example 10. Let $S = \{t_1, t_2, t_3\}$, $P = \{\{t_1, t_2\}, \{t_3\}\}$ and $Q = \{\{t_1, t_2\}\}$. Then

$\mathsf{NE}_\lambda[x, S] : x \neq t_1 \wedge x \neq t_2 \wedge x \neq t_3$ $\quad \mathsf{NE}_{\mathbb{Z}}[x, S, Q] : |x| \neq |t_3|$
$\mathsf{NE}'_\lambda[S, P] : t_1 \neq t_3 \wedge t_2 \neq t_3$ $\quad \mathsf{EQ}_{\mathbb{Z}}[x, S, Q] : |x| = |t_1| \wedge |x| = |t_2|$
$\mathsf{EQ}_\lambda[S, P] : t_1 = t_2$

In fact $\mathsf{PART}[x, S, P, Q]$ is an equality completion of terms $\{x\} \cup S$. Starting from (8) we guess a partition P of $S = \{t_1, \ldots, t_n\}$ and a set $Q \subseteq P$, resulting in

$$\exists x : \lambda \left[\mathsf{NE}_\lambda[x, S] \wedge \mathsf{PART}'[x, S, P, Q] \wedge \Phi_\lambda(\bar{z}) \wedge \Phi_{\mathbb{Z}}(x, \bar{y}, \bar{z})\right], \tag{11}$$

which says that there exists an x such that (i) x is not equal to any terms in S, and (ii) S is partitioned into $|P|$ equivalence classes among which there are $|Q|$

classes whose members have the same length as x. But this is exactly what the counting constraint $\mathsf{CNT}_{k,|Q|}(|x|)$ states, and so we can transform (11) to

$$\exists n : \mathbb{Z}\ \Big[\mathsf{Tree}(x)\big[|x|/n\big] \wedge \mathsf{CNT}_{k,|Q|}(n) \wedge \\ \mathsf{PART}'[x,S,P,Q]\big[|x|/n\big] \wedge \Phi_\lambda(\bar{z}) \wedge \Phi_\mathbb{Z}(|x|/n, \bar{y}, \bar{z})\Big]. \qquad (12)$$

Again, we are left with the known task of eliminating integer quantifiers. As before, in case of presence of $\mathsf{Is}_\alpha(x)$, we use $\mathsf{Tree}^\alpha(x)$ and $\mathsf{CNT}^\alpha_{k,|Q|}(n)$ instead of $\mathsf{Tree}(x)$ and $\mathsf{CNT}_{k,|Q|}(n)$, respectively, in (12).

Example 11. Consider, in $\mathfrak{B}^{=1}_{\mathsf{List}}$ (with atom a), $\exists x : \lambda(x \neq \mathsf{cons}(a,a) \wedge |x| = 2)$. The only compatible completion is $\exists x : \lambda(\mathsf{Is}_{\mathsf{cons}}(x) \wedge |x| = |\mathsf{cons}(a,a)| \wedge x \neq \mathsf{cons}(a,a) \wedge |x| = 2)$. This produces the counting constraint $\mathsf{CNT}^{\mathsf{cons}}_{1,1}(|x|)$ which is $|x| > 2$, contradicting $|x| = 2$.

We can also derive a decision procedure for $\mathsf{Th}(\mathfrak{B}^{<\omega})$ using the above elimination procedure.

Theorem 4. $\mathsf{Th}(\mathfrak{B}^{<\omega})$ *(in the languages with no constants) is decidable.*

9 Complexity

In this section we briefly discuss complexity of the quantifier theories. Let n be the input size of Φ. First it is not hard to see that both $\mathsf{Th}^\forall(\mathfrak{B}^\omega)$ and $\mathsf{Th}^\forall(\mathfrak{B}^{=k})$ are NP-hard as they are super theories of $\mathsf{Th}^\forall(\mathfrak{A}_\lambda)$ and $\mathsf{Th}^\forall(\mathfrak{A}_\mathbb{Z})$, either of which is NP-complete. Second, Algorithm 3 computes Φ_Δ in $O(n)$ (see Proposition 2) and so does Algorithm 5 (see Proposition 4). 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 Algorithm 4 and Algorithm 6 respectively) is in P. Therefore both $\mathsf{Th}^\forall(\mathfrak{B}^\omega)$ (or $\mathsf{Th}^\forall(\mathfrak{B}^{<\omega})$) and $\mathsf{Th}^\forall(\mathfrak{B}^{=k})$ are NP-complete.

10 Conclusion

We presented four classes of decision procedures for recursive data structures integrated with Presburger arithmetic. Our technique is based on the extraction of sound and complete integer constraints from data constraints. We believe that this technique may apply to arithmetic integration of various other theories.

We plan to extend our results in two directions. The first is to reason about the combination of recursive data structures with integers in richer languages such as the theory of recursive data structures with subterm relation $\preceq$ [20]. The second is to relax the restriction of unique construction of data objects to enable handling of structures in which a data object can be constructed in more than one way such as in the theory of queues [1, 19] and word concatenation [12].

Recently it comes to our attention that the combination of Presburger arithmetic and term algebras has been used in [9, 10] to show that quantifier-free theory of term algebras with Knuth-Bendix is NP-complete. We will compare the work with ours in the extended version of this paper.

References

1. Nikolaj S. Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Computer Science Department, Stanford University, November 1998.
2. 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.
3. K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Annals of Pure and Applied Logic*, 48:1–79, 1990.
4. D. C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, volume 7, pages 91–99. American Elsevier, 1972.
5. J. Downey, R. Sethi, and R. E. Tarjan. Variations of the common subexpression problem. *J. ACM*, 27:758–771, 1980.
6. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, second edition, 2001.
7. J. Ferrante and C. W. Rackoff. *The Computational Complexity of Logical Theories*. Springer-Verlag, 1979.
8. Wilfrid Hodges. *Model Theory*. Cambridge University Press, Cambridge, UK, 1993.
9. Konstantin Korovin and Andrei Voronkov. A decision procedure for the existential theory of term algebras with the knuth-bendix ordering. In *Proc. 15th IEEE Symp. Logic in Comp. Sci.*, pages 291 – 302, 2000.
10. Konstantin Korovin and Andrei Voronkov. Knuth-Bendix constraint solving is NP-complete. In *Proceedings of 28th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of *Lecture Notes in Computer Science*, pages 979–992. Springer, 2001.
11. M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite tree. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 348–357. IEEE Computer Society Press, 1988.
12. G. S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977.
13. A. I. Mal'cev. 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.
14. Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Prog. Lang. Sys.*, 1(2):245–257, October 1979.
15. Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, April 1980.
16. Derek C. Oppen. Elementary bounds for presburger arithmetic. In *Proc. 5th ACM Symp. on Theory of Computing*, pages 34–37, 1973.
17. Derek C. Oppen. Reasoning about recursively defined data structures. *J. ACM*, 27(3), July 1980.
18. C. R. Reddy and D. W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *Proceedings of the 10th Annual Symposium on Theory of Computing*, pages 320–325. ACM Press, 1978.
19. Tatiana Rybina and Andrei Voronkov. A decision procedure for term algebras with queues. *ACM Transactions on Computational Logic*, 2(2):155–181, 2001.
20. K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebras. *J. ACM*, 34,2:492–510, 1987.

21. Calogero G. Zarba. Combining multisets with integers. In Andrei Voronkov, editor, *Proc. of the* 18^{th} *Intl. Conference on Automated Deduction*, volume 2392 of *LNAI*, pages 363–376. Springer, 2002.
22. Calogero G. Zarba. Combining sets with integers. In Alessandro Armando, editor, *Frontiers of Combining Systems*, volume 2309 of *LNAI*, pages 103–116. Springer, 2002.

A Example 1 Revisited

We need to show that LISP lists can be axiomatized by the following axioms.

$$\mathsf{car}(\mathsf{cons}(x, y)) = x \tag{13}$$

$$\mathsf{cdr}(\mathsf{cons}(x, y)) = y \tag{14}$$

$$\mathsf{Is}_A(x) \leftrightarrow \neg\mathsf{Is}_{\mathsf{cons}}(x) \tag{15}$$

$$\mathsf{Is}_A(x) \leftrightarrow \{\mathsf{car}, \mathsf{cdr}\}^+(x) = x \tag{16}$$

$$\mathsf{Is}_{\mathsf{cons}}(x) \leftrightarrow \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x)) = x \tag{17}$$

It follows from Proposition 1 that LISP lists is axiomatizable by the following axioms.

$$t(x) \neq x \textit{ if } t \textit{ is built solely by the constructor } \mathsf{cons} \textit{ and } t(x) \textit{ is not } x. \tag{18}$$

$$\mathsf{Is}_A(x) \rightarrow x \neq \mathsf{cons}(y, z) \tag{19}$$

$$\mathsf{cons}(x, y) = \mathsf{cons}(x', y') \rightarrow x = x' \wedge y = y' \tag{20}$$

$$\mathsf{Is}_{\mathsf{cons}}(x) \leftrightarrow \exists y, z(\mathsf{cons}(y, z) = x) \tag{21}$$

$$\mathsf{Is}_{\mathsf{cons}}(x) \leftrightarrow \neg\mathsf{Is}_A(x) \tag{22}$$

$$\mathsf{car}(x) = y \leftrightarrow \exists z(\mathsf{cons}(y, z) = x) \vee (\forall z_1, z_2(\mathsf{cons}(z_1, z_2) \neq x) \wedge x = y) \tag{23}$$

$$\mathsf{cdr}(x) = y \leftrightarrow \exists z(\mathsf{cons}(z, y) = x) \vee (\forall z_1, z_2(\mathsf{cons}(z_1, z_2) \neq x) \wedge x = y) \tag{24}$$

So it remains to show that (13)-(17) imply (18)-(24).

- (Proof of (18).) By induction on the structure of $t(x)$.
  Base Case. Without loss of generality we assume $t(x) \equiv \mathsf{cons}(x, y)$. Suppose

  $$\mathsf{cons}(x, y) = x.$$

  Applying car and cdr to both sides of the equation and simplifying the results by (13) and (14), respectively, we obtain

  $$x = \mathsf{car}(x) \textit{ and } y = \mathsf{cdr}(x)$$

  which imply $\mathsf{Is}_A(x)$ by (16) and $\mathsf{Is}_{\mathsf{cons}}(x)$ by (17), contradicting (15).
  Induction Step. If $t(x)$ is of the form $\mathsf{cons}(x, t')$ (or $\mathsf{cons}(t', x)$), the proof is the same as for the base case. Otherwise, as before, we may assume that $t(x) \equiv \mathsf{cons}(t_1(x), t')$ where $t_1(x)$ is not identical to x. Suppose

  $$\mathsf{cons}(t_1(x), t') = x.$$

  By (13) and (14) we know there exists $L \in \{\mathsf{car}, \mathsf{cdr}\}^+$ such that $L(x) = x$. By (16) we have $\mathsf{Is}_A(x)$ and $\mathsf{car}(x) = x$. But then $\mathsf{cons}(t_1(x), t') = x$ implies $t_1(x) = x$ by (13), contradicting the induction hypothesis.
- (Proof of (19).) Suppose that $\mathsf{Is}_A(x)$ and there exists y and z such that $x = \mathsf{cons}(y, z)$. Then by (13) and (14) we have

  $$y = \mathsf{car}(x) \textit{ and } z = \mathsf{cdr}(x)$$

  which imply $\mathsf{Is}_{\mathsf{cons}}(x)$ by (17), contradicting (15).

- (Proof of (20).) (20) can be obtained by first applying car and cdr, respectively, to the both sides of the antecedent

$$\mathsf{cons}(x, y) = \mathsf{cons}(x', y')$$

and then doing simplification using (13) and (14).
- (Proof of (21).) (21) is equivalent to (19) by (15).
- (Proof of (22).) (22) is just (15).
- (Proof of (23).) We consider two cases.
Case 1: $\mathsf{Is}_A(x)$. By (16) the left hand side of (23) simplifies to $x = y$. And by (19) the right hand side of (23) also simplifies to $x = y$.
Case 2: $\mathsf{Is}_{\mathsf{cons}}(x)$. By (21), (23) simplifies to

$$\mathsf{car}(x) = y \leftrightarrow \exists z(\mathsf{cons}(y, z) = x).$$

With $\mathsf{Is}_{\mathsf{cons}}(x)$ and $\mathsf{car}(x) = y$, (17) gives

$$\mathsf{cons}(y, \mathsf{cdr}(x)) = x.$$

Take $z = \mathsf{cdr}(x)$ we have $\exists z(\mathsf{cons}(y, z) = x)$. On the other hand, let z be such that $\mathsf{cons}(y, z) = x$. Then by (13) we have $\mathsf{car}(x) = y$.
- (Proof of (24).) Similar to the proof of (23).

B An Easy Lemma

The following lemma is a version of the Euclidean algorithm.

Lemma 1. *Let* $d_1, \ldots, d_n$ *be positive integers and* $gcd(d_1, \ldots, d_n)$ *the* greatest common divisor *of them. Then there exists* $N_{\bar{d}}$ *such that for any* $k \geq N_{\bar{d}}$,

$$gcd(d_1, \ldots, d_n) \mid k - N_{\bar{d}},$$

if and only if k *can be expressed as a non-negative linear combination of* $d_1, \ldots, d_n$, *that is, there exist non-negative integers* $x_1, \ldots, x_n$ *such that*

$$k = \sum_{i=1}^{n} x_i d_i.$$

If $n = 2$, then $N_{\bar{d}} = lcm(d_1, d_2)$, the *least common multiple* of d_1 and d_2. As an easy consequence, any number of the form

$$N_{\bar{d}} + gcd(d_1, \ldots, d_n) \cdot t \quad (t \geq 0)$$

can be expressed as a non-negative linear combination of $d_1, \ldots, d_n$. However there is no closed-form formula to decide if a number smaller than $N_{\bar{d}}$ can be put into a non-negative linear combination of $d_1, \ldots, d_n$. Finding the largest such "indecomposable" number (when $gcd(d_1, \ldots, d_n) = 1$) is known as Frobenius Coin Problem which is NP-hard.

C The Proofs of Complexity

Proof of Proposition 2. We show that Φ_Δ obtained by Algorithm 3 is expressible in a quantifier-free Presburger formula linear in the size of Φ.

Let $d_1, \ldots, d_n$ be distinct arities of the constructors in a specific language. Since n and d_i $(1 \leq i \leq n)$ are constant values (in a fixed language), it is easily seen that predicates $\mathsf{Tree}(t)$, $\mathsf{Node}^\alpha(t, \bar{t}_\alpha)$ and $\mathsf{Tree}^\alpha(t)$ are of constant length (see Section 5). Note that all generalized integer variables has length 1 as we can abstract $|t|$ to a fresh integer variable v. Though $\mathsf{Tree}(t)$ involves quantifiers, by the quantifier elimination of Presburger arithmetic, there exists an equivalent quantifier-free formula with the length at most triple exponential of the length of $\mathsf{Tree}(t)$ ([16]). So such an quantifier-free formula is still of constant length. The similar result holds for $\mathsf{Tree}^\alpha(t)$ and therefore Φ_Δ obtained by Algorithm 3 can be expressible in quantifier-free Presburger formula linear in the size of Φ.

In fact we don't need to do quantifier elimination at all. We illustrate this for $\mathsf{Tree}(t)$. $\mathsf{Tree}(t)$ states that $|t| - 1$ is a non-negative linear combination of $d_i - 1$ $(1 \leq i \leq n)$. Let gcd be the greatest common divisor of $\{d_i - 1 \mid 1 \leq i \leq n\}$. By Lemma 1 there exists $N_{\bar{d}}$ such that if $|t| - 1 \geq N_{\bar{d}}$, $|t| - 1$ can assume values of the form

$$N_{\bar{d}} + k \cdot gcd \quad (k \geq 0).$$

Thus, $\mathsf{Tree}(t)$ is equivalent to

$$\bigvee_{a \in S} (|t| - 1 = a) \vee gcd \mid (|t| - 1 - N_{\bar{d}}).$$

where S is some subset of $\{1, \ldots, N_{\bar{d}}\}$ and S can be recomputed for a fixed language. Similarly we can obtain a quantifier-free formula equivalent to $\mathsf{Tree}^\alpha(t)$ without going through the quantifier elimination procedure.

Proof of Proposition 3. We show that $\mathsf{CNT}^\alpha_{k,n}(x)$ is expressible in a quantifier-free Presburger formula that can be computed in time $O(n)$.

Assume that L_λ has m constructors and let d_i $(1 \leq i \leq m)$ be the i^{th} arity. Let $f(p)$ be the number of distinct term trees of length p. For $p > 1$, we have the following recurrence relation.

$$f(p) = \sum_{h=1}^{m} \sum_{i_1 + \ldots + i_{d_h} = p} \prod_{j=1}^{d_h} f(i_j). \tag{25}$$

The reason is as follows: there are m possible ways to label the root of a tree; for a root with d children whose lengths are $i_1, \ldots, i_d$, respectively, there are $\prod_{j=1}^{d} f(i_j)$ combinations. By dynamic programming, we can compute $f(1), f(2)$, $\ldots$ in the bottom-up fashion. On the way we will find the first l such that $f(l) > n$ and we take $\mathsf{CNT}_{k,n}(x)$ be $x \geq l$. Similarly we can get $\mathsf{CNT}^\alpha_{k,n}(x)$. Since there are $O(p^{d-1})$ different sequences of positive numbers whose sum is $p-1$, $f(l)$ can be obtained by $O(l^d)$ arithmetic operations. As $f(p)$ grows exponentially in p, l is at the scale of $O(\log n)$. Moreover, as all integers in the computation is less than n, any arithmetic operation costs time $O(\log n)$. Therefore the search for such l can be done in $O(n)$.

We are almost done except one complication. We say that term trees can grow *continuously*, if for any tree T, the following conditions are satisfied.

1. There is a tree T' whose length is the next legitimate tree length greater than the length of T.
2. Such a T' can be obtained by expanding a leaf of T one level.

It is not hard to see that if trees can grow continuously, then for any legitimate tree length x, $x \geq m$ implies $f(x) > n$. (This is the case if we have a binary constructor, or all constructors have the same arity.) Therefore $\mathsf{CNT}^{\alpha}_{k,n}(x)$ is just

$$x \geq m.$$

If trees can not grow continuously, then it is not true that for any $x \geq m$, $f(x) > n$. To see this, let us consider an extreme example; a language with only one atom and with two constructors of arity 4 and 1001 respectively. Let $n' = f(1000)$. With no doubt, there are so many distinct trees of length 1000 (constructed using solely the constructor of arity 4), and so n' is a very huge number. But there is only one tree of length 1001 simply because 1000 can not be divided by 3 and so such a tree has to be constructed using the constructor of arity 1001. So 1000 is the least number such that $f(1000) > n' - 1$ while $f(1001) << n' - 1$. We should remark that such an anomaly very unlikely happens in practice. However, by Lemma 1 we still have $f(x) > n$ for any

$$x \geq m + N_{\bar{d}}.$$

So finally $\mathsf{CNT}^{\alpha}_{k,n}(x)$ assume the form

$$\bigvee_{a \in S} (x = a) \vee x \geq m + N_{\bar{d}},$$

where S is a subset of $\{m, \ldots, m + N_{\bar{d}}\}$. Since $N_{\bar{d}}$ and hence the size of S are constants, we see that $\mathsf{CNT}^{\alpha}_{k,n}(x)$ can be computed in $O(n)$.

Proof of Proposition 4. We show that Φ_{Δ} obtained by Algorithm 5 is expressible in a quantifier-free Presburger formula and the size of such a formula is quadratic in the size of Φ.

By Proposition 3 this amounts to show that the computation of all counting constraints in Φ_{Δ} can be done in quadratic time. Let us call $C = \{t_1, \ldots, t_n\}$ a *cluster* (w.r.t. Φ) if $\mathsf{NEQ}(t_1, \ldots, t_n)$ is in Φ. A cluster C is *maximum* if any proper superset C' of C is not a cluster (w.r.t. Φ). We observe the followings.

1. Since Φ is an equality completion, any term t is in exactly one maximum cluster. It follows that any two distinct maximum clusters are disjoint, the union of all maximum clusters is the set of all terms occurring in Φ and there are at most n maximum clusters.
2. For each maximum cluster $\{t_1, \ldots, t_n\}$, as $|t_i| = |t_j|$ for $1 \leq i < j \leq n$, we only need to add one counting constraint $\mathsf{CNT}^{\alpha}_{k,n}(|t|)$ for some $t \in \{t_1, \ldots, t_n\}$.

By Proposition 3 and the above observations, we need at most n counting constrains each of which can be computed in $O(n)$. Therefore, we can obtained Φ_{Δ} in $O(n^2)$.

D The Proofs of Completeness

Proof of Theorem 1. We show that Φ_Δ obtained by Algorithm 3 is the induced length constraint of Φ_λ.

We may assume that the language only has constructors [2]. This means that leaf vertices of G_λ are labeled either by constants or variables. We say G'_λ is an *extension* of G_λ if G'_λ is obtained from G_λ by "pasting" to every variable leaf a well-formed ground term trees. (We need to make sure that the type of a term tree matches the type of the leaf to be replaced.) It is not difficult to see the one-to-one correspondence between variable assignments of Φ_λ and extensions of G_λ.

By Algorithm 3, Φ_Δ has the following two properties:

1. for any assignment ν_λ of Φ_λ, $|\nu_\lambda|$ is a satisfying assignment of Φ_Δ.
2. for any satisfying assignment ν_Δ of Φ_Δ, there is a corresponding data assignment ν such that $|\nu| = \nu_\Delta$.

The soundness of Φ_Δ follows immediately from the first property. For completeness let ν and ν_Δ be as stated in the second property. (Note that ν may not satisfy Φ_λ, otherwise, we are done.) We transform ν in two steps.

1. Let ν' be obtained from ν by substituting each atom in terms in ν for a fresh constant. This can be done as $\mathfrak{B}^\omega$ has infinitely many atoms.
2. For variables in the same equivalence class we take any arbitrary representative, say z, and assign every other variables in the class $\nu'(z)$. Let us denote by ν_λ the resulting assignment.

We claim that ν_λ obtained by the above transformation is a satisfying assignment for Φ_λ. But before we go into the details of the proof, it is worthwhile to illustrate the above transformation by an example.

Example 12. Figure 3 shows the DAG representation of

$$\mathsf{Is}_{\mathsf{cons}}(x) \wedge \mathsf{cons}(y, z) = \mathsf{cons}(\mathsf{cdr}(x), z) \wedge \mathsf{car}(x) \neq z \tag{26}$$

As in Example 5, the final computed $R\Updownarrow$ is $\{(v_1, v_2), (v_3, v_4), (v_6, v_7)\}$. Before we do the transformation we need to abstract terms $\mathsf{car}(x)$ and $\mathsf{cdr}(x)$ to fresh variables. Let $x_1 = \mathsf{car}(x)$ and $x_2 = \mathsf{cdr}(x)$ (where x_1 and x_2 are new variables). Now $x = \mathsf{cons}(x_1, x_2)$ and then we don't treat it as a variable in the construction of an assignment. Suppose that we take an satisfying integer assignment

$$\nu_\Delta : \{|x_1| = 1, |x_2| = 3, |y| = 3, |z| = 1\}.$$

Let a possible corresponding data assignment ν be

$$\{x_1 = a, x_2 = \mathsf{cons}(\mathsf{cons}(a, a), a), y = \mathsf{cons}(a, \mathsf{cons}(a, a)), z = a\}.$$

The first step of substitution (of every atom for a fresh constant) will render

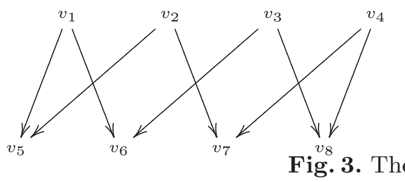
$$\nu' : \{x_1 = a_1, x_2 = \mathsf{cons}(\mathsf{cons}(a_2, b_2), c_2), y = \mathsf{cons}(a_3, \mathsf{cons}(b_3, c_3)), z = a_4\}.$$

Since x_2 and y are in the same equivalence class, the second step will give

$$\nu_\lambda : \{x_1 = a_1, x_2 = \mathsf{cons}(\mathsf{cons}(a_2, b_2), c_2), y = \mathsf{cons}(\mathsf{cons}(a_2, b_2), c_2), z = a_4\}.$$

A quick check will convince us that ν_λ satisfies (26) (while ν does not).

[2] The generalization can be done with variable abstraction of terms solely built by selectors. We will illustrate this in Example 12



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

Fig. 3. The DAG of (26).

Proof Continued. We show that ν_λ obtained by the above transformation is a satisfying assignment for Φ_λ.

Clearly $|\nu_\lambda| = |\nu'| = |\nu| = \nu_\Delta$ as the transformation won't affect the lengths of terms at all. Moreover, ν_λ respects $R\Updownarrow$, that is, for any terms t and s, $(t, s) \in R\Updownarrow$ implies $\nu_\lambda(t) = \nu_\lambda(s)$ (or $G'_t = G'_s$). (Recall that G'_λ is the extension of G_λ corresponding to ν_λ.) So it suffices to show that $G'_{t_1} \neq G'_{t_2}$ for any two non-equivalent vertices t_1 and t_2 in G_λ. For a term t denote the *height* of G'_t by $\mathsf{ht}(t)$. Suppose the contrary and let t and s be two vertices such that $G'_t = G'_s$ but $t \neq s$ is in Φ_λ. With out loss of generality we assume that $h = \mathsf{ht}(t) = \mathsf{ht}(s)$ is minimal. If $h = 1$, then both t and s should be variables in different equivalence classes. By the construction of ν_λ, $\nu_\lambda(t) \neq \nu_\lambda(s)$, a contradiction. If $r > 1$, then $\nu_\lambda(t) = \nu_\lambda(s)$ implies that $\nu_\lambda(t[1]) = \nu_\lambda(s[1])$. But $\mathsf{ht}(t[1]) = \mathsf{ht}(s[1]) = h-1 < h$, contradicting the minimality of h.

Proof of Theorem 2. The soundness proof is the same as that for Theorem 1. For completeness we assume that after running Algorithm 1 on G_λ, terms (trees) in same equivalence class have been merged, (which amounts to call the second step of the transformation in the previous proof). Since Φ_λ is an equality completion, any two distinct vertices in the resulting G_λ are "necessarily" unequal. Let ν_Δ be a satisfying assignment of Φ_Δ. We order all generalized integer variables according to the values given by ν_Δ as follows.

$$\underbrace{|t_0^{(1)}| = \ldots = |t_{n_1}^{(1)}|}_{cluster\ 1} < \underbrace{|t_0^{(2)}| = \ldots = |t_{n_2}^{(2)}|}_{cluster\ 2} < \ldots < \underbrace{|t_0^{(k)}| = \ldots = |t_{n_k}^{(k)}|}_{cluster\ k}$$

Denote by l_i the length of terms in the i^{th} cluster. Note that for any data term t, $|t|$ appears in Φ_Δ and hence in the above sequence. Beginning with terms in the 1^{th} cluster, namely $t_1^{(1)}, \ldots, t_{n_1}^{(1)}$, we incrementally construct a satisfying assignment ν_λ for Φ_λ. Obviously $t_i^{(1)}$ $(0 \leq i \leq n_1)$ is either a constant or a data variable as its length is the smallest. And we only need to consider $t_i^{(1)}$ which are data variables. By Algorithm 6 we know that $\mathsf{CNT}_{k,n_1}(l_1)$ is in Φ_Δ. As ν_Δ satisfies Φ_Δ, there are at least n_1+1 different terms at length l_1. Therefore we can simply assign each $t_i^{(1)}$ (if it is variable) a distinct term. Now suppose that the terms in the i^{th} cluster have been assigned. Consider the $(i+1)^{th}$ cluster. By this time values of all non-variable terms in the $(i+1)^{th}$ cluster have been fixed simply because variables (if any) in those terms have length $\leq l_i$ and those variables have been assigned by the i^{th} round. By the same argument as before, due to the presence of $\mathsf{CNT}_{k,n_{i+1}}(l_{i+1})$ in Φ_Δ, we are able to assign each variable in $(i+1)^{th}$ cluster a different tree of length l_{i+1}. Note that the assignment in each round won't create any equality between terms in G_λ simply as all terms of the same length are different. By induction we can eventually construct a satisfying assignment ν_λ for Φ_λ such that $|\nu_\lambda| = \nu_\Delta$.

E The Proofs of Decidability

Proof of Theorem 3. We show that $\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega}) = \mathsf{Th}^{\forall}(\mathfrak{B}^{\omega})$ in the language with no constants. First note that it only makes sense to talk about $\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega})$ (or $\mathsf{Th}(\mathfrak{B}^{<\omega})$) if we drop the axiom that $a \neq b$ for $a \not\equiv b$, and hence essentially treat every constant as an atom variable.

By the first monotonicity property of counting constraints, namely, $\mathsf{CNT}^{\alpha}_{k,n}(x) \rightarrow \mathsf{CNT}^{\alpha}_{l,n}(x)$ for $l \geq k > 0$, we know that the larger the atom domain, the less restrictive Φ_{Δ} is, and hence the more (quantifier-free) formulae are satisfiable, which in turn means that the fewer (universally quantified) formulae are valid. (Recall that φ is valid if and only if $\neg\varphi$ is unsatisfiable.) Thus,

$$\mathsf{Th}^{\forall}(\mathfrak{B}^{\omega}) \subseteq \ldots \subseteq \mathsf{Th}^{\forall}(\mathfrak{B}^{=k}) \subseteq \ldots \subseteq \mathsf{Th}^{\forall}(\mathfrak{B}^{=1}),$$

and so

$$\mathsf{Th}^{\forall}(\mathfrak{B}^{\omega}) \subseteq \bigcap_{i>0}^{\omega} \mathsf{Th}^{\forall}(\mathfrak{B}^{=i}).$$

On the other hand, for a formula Φ of size n the number of terms of Φ is bounded by $O(n)$, and so is the size of maximum clusters. Therefore all counting constraints occurring in Φ_{Δ} will have the form $\mathsf{CNT}^{\alpha}_{k,m}(x)$ with m bounded by $O(n)$, and hence for a large enough k all those counting constraints will degenerate to $x > 0$ which is apparently true. This means that Algorithm 6 is the same as Algorithm 4 for $\mathfrak{B}^{=k}$ with any large enough k. Therefore, if Φ is valid in all $\mathfrak{B}^{=i}$ $(i > 0)$, then it is valid in $\mathfrak{B}^{\omega}$, which implies that

$$\bigcap_{i>0}^{\omega} \mathsf{Th}^{\forall}(\mathfrak{B}^{=i}) \subseteq \mathsf{Th}^{\forall}(\mathfrak{B}^{\omega}).$$

Put all together, as $\mathfrak{B}^{<\omega} = \bigcup_{i>0}^{\omega} \mathfrak{B}^{=i}$ and hence $\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega}) = \bigcap_{i>0}^{\omega} \mathsf{Th}^{\forall}(\mathfrak{B}^{=i})$, we have

$$\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega}) = \mathsf{Th}^{\forall}(\mathfrak{B}^{\omega}).$$

Therefore Algorithm 4 is also a decision procedure for $\mathsf{Th}^{\forall}(\mathfrak{B}^{<\omega})$. This result bears resemblance to the *finite model property* in the first-order logic.

Proof of Theorem 4. We show that $\mathsf{Th}(\mathfrak{B}^{<\omega})$ (in the language with no constants) is decidable.

First we note that $\mathsf{Th}(\mathfrak{B}^{<\omega}) \neq \mathsf{Th}(\mathfrak{B}^{\omega})$. This can be seen as we have shown that $\mathsf{Th}(\mathfrak{B}^{<\omega})$ contains fewer existential sentences and more universal sentences than $\mathsf{Th}(\mathfrak{B}^{\omega})$ does. However, similar as before, for a formula Φ of size n, in one-step elimination [3] the maximum number k_n of t_i's in formulae of the form (8) is bounded by $2^{O(n)}$ even if we take into account of the newly generated variables. So for any $i \geq k_n$ the one-step elimination for $\mathsf{Th}(\mathfrak{B}^{=i})$ is the same as the one

[3] By one-step elimination we mean the process of eliminating one quantifier which occurs in the original input formula. Here n refers to the size of the formula currently being processed. Let m be the size of the original input formula. In general, n can not be bounded by any k-fold exponentials in m for any fixed k.

for $\mathsf{Th}(\mathfrak{B}^\omega)$. Then we can run the procedure k_n times. Each time we obtain a reduction Φ_i and finally we have

$$\Phi \leftrightarrow \bigvee_{i}^{k_n} \Phi_i.$$

Note that certain properties on the cardinality of the atom domain can be expressed by formulae with quantifiers. For example,

$$\exists x, y(\mathsf{Is}_A(x) \wedge \mathsf{Is}_A(y) \wedge x \neq y \wedge \forall z(\mathsf{Is}_a(z) \rightarrow (z = x \vee z = y)))$$

states that the atom domain have exactly two elements. So the truth of a formula could rely on some implicit 'cardinality statement" in the formula itself. The above decidability result means that those hidden cardinality statements can always be "discharged" by the quantifier elimination procedure.