Decision Procedures for Term Algebras with Integer Constraints

Ting Zhang, Henny B. Sipma, and Zohar Manna [1]

Computer Science Department, Stanford University

Abstract

Term algebras can model recursive data structures which are widely used in programming languages. To verify programs we must be able to reason about these structures. However, as programming languages often involve multiple data domains, in program verification decision procedures for a single theory are usually not applicable. An important class of *mixed* constraints consists of combinations of data structures with integer constraints on the size of data structures. Such constraints can express memory safety properties such as absence of memory overflow and out-of-bound array access, which are crucial for program correctness. In this paper we extend the theory of term algebras with the length function which maps a term to its size, resulting in a combined theory of term algebras and Presburger arithmetic. This arithmetic extension provides a natural but tight coupling between the two theories, and hence the general purpose combination methods like Nelson-Oppen combination are not applicable. 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.

Key words: decision procedures, recursive data structures, term algebras, Presburger arithmetic, quantifier elimination, combination of satisfiability procedures

Email address: `tingz,sipma,zm@cs.stanford.edu` (Ting Zhang, Henny B. Sipma, and Zohar Manna).

[1] 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.

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 paper 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 paper 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 $\mathrm{Th}^{\forall}(\mathrm{TA}^{\omega})$) [1]. 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.

For the first-order theory, we first present a new quantifier elimination procedure for $\mathrm{Th}(\mathrm{TA})$ and then extend it to an elimination procedure for $\mathrm{Th}(\mathrm{TA}_{\mathbb{Z}})$. Our elimination procedure for $\mathrm{Th}(\mathrm{TA})$ is based on the elimination procedure in [2], 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 [3]. The improved version that allows elimination of blocks of quantifiers was published in [4]. 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 paper 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 [5]. A more efficient algorithm was later discovered by Cooper [6] and further improved by Reddy and Loveland [7].

It is well-known that recursive data structures can be modeled as term algebras which were shown to be decidable by Mal'cev using quantifier elimination [8]. This result was proved again several times in different settings [9,10,2,11–15,3].

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 [9]. Comon and Delor presented an elimination procedure for term algebras with membership predicate in the regular tree language [11]. Backofen presented an elimination procedure for structures of feature trees with arity constraints [12]. Rybina and Voronkov showed the decidability of term algebras with queues [13]. Kuncak and Rinard showed the decidability of term powers, which are term algebras augmented with coordinate-wise defined predicates [14].

Decision procedures for the quantifier-free theory of recursive data structures were discovered by Nelson, Oppen et al. [16,1,17]. Oppen gave a linear algorithm for acyclic structures [1] and (with Nelson) a quadratic algorithm for cyclic structures [16]. If the values of the selector functions on constants are specified, then the problem is NP-complete [1].

A general combination method for decision procedures for quantifier-free theories was developed by Nelson and Oppen in 1979 [18]. The method requires that component theories be loosely coupled, that is, have disjoint signatures, and are stably infinite [2] [19]. Tinelli and Ringeissen presented a general theoretical framework for combining satisfiability procedures of theories with non-disjoint signatures [20]. Tinelli and Zarba generalized Nelson-Oppen's method to theories in multisorted languages [21]. Armando, Ranise and Rusinowitch presented a uniform framework using superposition for deriving decision procedures for certain combined theories [22]. Ghilardi presented a set of model-theoretical conditions for the existence of Nelson-Oppen combination schema on theories having non-disjoint signatures [23]. However, 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 [24] and a theory of multisets and integers [25]. The integration of Presburger arithmetic with recursive data structures was discussed by Bjørner [26] and an incomplete procedure was implemented in STeP (Stanford Temporal Prover) [27].

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 [28]. It readily follows from the reduction technique that the first order theory of sets with cardinality constraints in Presburger arithmetic is decidable [29]. Recently, Revesz [30], and Kuncak and Rinard [31] independently presented decision procedures for this

[2] 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.

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 [32,33]. Along this line of investigation we proved the decidability of the first-order theory of Knuth-Bendix orders [34] using quantifier elimination. The elimination procedure makes extensive use of Presburger arithmetic in the reduction of quantifiers on term variables to quantifiers on integer variables.

Paper Organization. Section 2 presents the notation and terminology. Section 3 introduces the language and structure of term algebras. Section 4 describes Oppen's algorithm for recursive data structures. Section 5 presents our decision procedures for the quantifier-free theory of term algebras augmented with a length function and Presburger arithmetic. In Section 5.1 we first describe the theory and then in Section 5.2 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 5.3 we specialize this procedure for structures with infinite constant domain, and in Section 5.4 we refine it further for structures with finite constant domain. Section 5.5 outlines the approach to obtain decision procedures for structures whose constant domain contains relations besides equality. Section 5.6 discusses the complexity of the decision procedures for the quantifier-free theories.

Section 6 presents a new decision procedure for the first-order theory of term algebras. Section 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 7.2, we extend the elimination procedure for term algebras presented in Section 6 to term algebras with integers. Section 7.3 further generalizes the result to structures whose constant domain has an internal structure and admits quantifier elimination. In Section 7.4 we discuss how this procedure can be adapted for theories with infinite languages. Section 8 concludes with some ideas for future work. Most proofs are provided in the Appendix.

2 Preliminaries

We assume the first-order syntactic notions of variables, parameters and quantifiers, and semantic notions of structures, satisfiability and validity as in [5].

A *signature* 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. 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, $\mathcal{V}(\varphi)$ 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. A formula $\psi(\bar{\boldsymbol{x}})$ can be put into *prenex form* $Q_1 y_1, \ldots, Q_n y_n\ \varphi(\bar{\boldsymbol{x}}, y_1, \ldots, y_n)$, where Q_i's are either $\exists$ or $\forall$ and $\varphi(\bar{\boldsymbol{x}}, y_1, \ldots, y_n)$ is quantifier-free, called the *matrix* of ψ.

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 $\mathrm{Th}(\mathfrak{A})$.

We use $\bar{\boldsymbol{x}}$ to denote a sequence of variables, say, $x_1, \ldots, x_n$, and $\exists\bar{\boldsymbol{x}}$ (resp. $\forall\bar{\boldsymbol{x}}$) as an abbreviation of $\exists x_1, \ldots, \exists x_n$ (resp. $\forall x_1, \ldots, \forall x_n$). We write $\varphi(\bar{\boldsymbol{x}})$ to indicate that all variables occurring freely in φ are among $\bar{\boldsymbol{x}}$. $(\exists\bar{\boldsymbol{x}})\varphi(\bar{\boldsymbol{x}})$ and $(\forall\bar{\boldsymbol{x}})\varphi(\bar{\boldsymbol{x}})$ are called *existential closure* and *universal closure* of $\varphi(\bar{\boldsymbol{x}})$, respectively. If $\varphi(\bar{\boldsymbol{x}})$ is quantifier-free, then $(\exists\bar{\boldsymbol{x}})\varphi(\bar{\boldsymbol{x}})$ is called $\exists_1$ and $(\forall\bar{\boldsymbol{x}})\varphi(\bar{\boldsymbol{x}})$ 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. 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.

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.

All above notions naturally generalize to multi-sorted logics. 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 $1 \leq i \leq n$, 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 $F : A_{s_1} \times \cdots \times A_{s_n} \to A_{s_{n+1}}$; (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".

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

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 $(m..n]$.

3 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 1 *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* $\text{type}(t)$*, is the outermost constructor symbol of* t*. We say that* t *is* α*-typed (or is an* α*-term) if* $\text{type}(t) = \alpha$*.*
(2) $\mathcal{C}$*: A set of constructors:* α*,* β*,* γ*, ... The arity of* α *is denoted by* $\text{ar}(\alpha)$*.*
(3) $\mathcal{A}$*: A set of constants:* a*,* b*,* c*, ... We require* $\mathcal{A} \neq \emptyset$ *and* $\mathcal{A} \subseteq \mathcal{C}$*. For* $a \in \mathcal{A}$*,* $\text{ar}(a) = 0$ *and* $\text{type}(a) = a$*.*
(4) $\mathcal{S}$*: A set of selectors. For a constructor* α *with arity* $k > 0$*, there are* k *selectors* $\text{s}_1^\alpha, \ldots, \text{s}_k^\alpha$ *in* $\mathcal{S}$*. We call* s_i^α $(1 \leq i \leq k)$ *the* i^{th} α*-selector. For a term* x*,* $\text{s}_i^\alpha(x)$ *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*,* $\text{Is}_\alpha(x)$ *is true if and only if* x *is an* α*-term. For a constant* a*,* $\text{Is}_a(x)$ *is just* $x = a$*. In addition there is a special tester* Is_A *such that* $\text{Is}_\mathsf{A}(x)$ *is true if and only if* x *is a constant.*

We use $\mathscr{L}_\mathbb{T}$ *to denote the language of term algebras.*

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 in the language consisting of only constructors. 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 paper we assume that $\mathscr{L}_\mathbb{T}$ is finite except in Section 5.3 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 7.4.

Example 2 *Consider the LISP list structure*

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

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 $z_1, \ldots, z_{\text{ar}(\alpha)}$. 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. $\alpha(x_1, \ldots, x_{\text{ar}(\alpha)}) \neq \beta(y_1, \ldots, y_{\text{ar}(\beta)})$, if $\alpha, \beta \in \mathcal{C}$ and $\alpha \not\equiv \beta$.
A3. $\alpha(x_1, \ldots, x_{\text{ar}(\alpha)}) = \alpha(y_1, \ldots, y_{\text{ar}(\alpha)}) \rightarrow \bigwedge_{1 \leq i \leq \text{ar}(\alpha)} x_i = y_i$.
A4. $\text{Is}_\alpha(x) \leftrightarrow \exists \bar{z}_\alpha \alpha(\bar{z}_\alpha) = x$ for $\alpha \in \mathcal{C}$. In particular, $\text{Is}_a(x) \leftrightarrow x = a$ for $a \in \mathcal{A}$.
A5. $\text{Is}_{\mathsf{A}}(x) \leftrightarrow \bigvee_{a \in \mathcal{A}} \text{Is}_a(x)$.
A6. $\text{s}_i^\alpha(x) = y \leftrightarrow \exists \bar{z}_\alpha\Big(\alpha(\bar{z}_\alpha) = x \wedge y = z_i)\Big) \vee \Big(\forall \bar{z}_\alpha(\alpha(\bar{z}_\alpha) \neq x) \wedge x = y\Big)$.

This set of axioms is a variant of the axiomatization given in [2]. 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 $\bigwedge_{i=1}^{k} \text{s}_i^\alpha(x) = x_i \wedge \text{Is}_\alpha(x)$ and $x = \alpha(x_1, \ldots, x_k)$.

We write $\alpha = (\text{s}_1^\alpha, \ldots, \text{s}_k^\alpha)$ to indicate that α is a non-nullary constructor with $\text{ar}(\alpha) = k$ and $\text{s}_1^\alpha, \ldots, \text{s}_k^\alpha$ 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, $\text{s}_i^\alpha(\alpha(x_1, \ldots, x_k))$ simplifies to x_i $(1 \leq i \leq k)$ and $\text{s}_j^\beta(\alpha(x_1, \ldots, x_k))$ simplifies to $\alpha(x_1, \ldots, x_k)$ for $\alpha \not\equiv \beta$. As a consequence, a selector term has the form $\text{s}_1(\ldots(\text{s}_n(x)\ldots))$ for $n \geq 0$. We use $L, F, G, H, \ldots$ to denote (possibly empty) selector sequences. So $\text{s}_1(\ldots(\text{s}_n(x)\ldots))$ can be abbreviated as Lx for $L = \text{s}_1, \ldots, \text{s}_n$. 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 $\text{depth}_\varphi(x)$. We use $\text{Is}_\alpha(t_1, \ldots, t_n)$ as an abbreviation for $\bigwedge_{i=1}^{n} \text{Is}_\alpha(t)$. We say a selector term $\text{s}_i^\alpha(t)$ is *proper* in a formula φ if $\text{Is}_\alpha(t)$ is a conjunct of φ. We can make selector terms proper with type information.

Definition 3 (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* $\mathrm{Is}_\alpha(t)$ *($\alpha \in \mathcal{C} \setminus \mathcal{A} \cup \{\mathsf{A}\}$) 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 4 *Let us consider in* List *the type complete constraint*

$$y \neq \mathrm{cons}(x, \mathrm{car}(x)) \ \wedge \ \mathrm{Is}_{\mathsf{A}}(x, \mathrm{car}(x)) \ \wedge \ \mathrm{Is}_{\mathrm{cons}}(y) \ . \tag{1}$$

Thanks to the type information, it can be simplified to

$$y \neq \mathrm{cons}(x, x) \ \wedge \ \mathrm{Is}_{\mathsf{A}}(x) \ \wedge \ \mathrm{Is}_{\mathrm{cons}}(y) \ . \tag{2}$$

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 $\mathrm{Is}_\alpha(t)$ ($\alpha \in \mathcal{C}$) for each term t occurring in φ. We call φ' so obtained a *type completion* of φ.

Example 5 *Let us revisit Example 4. It is easily seen that* (1) *is a type completion of* $y \neq \mathrm{cons}(x, \mathrm{car}(x))$.

Example 6 *Let* $\alpha, \beta \in \mathcal{C}$, $\alpha \not\equiv \beta$ *and* $\alpha = (\mathrm{s}_1^\alpha)$. *A possible type completion for* $y = \mathrm{s}_1^\alpha(x)$ *is*

$$y = \mathrm{s}_1^\alpha(x) \ \wedge \ \mathrm{Is}_\beta(x, \mathrm{s}_1^\alpha(x), y) \ , \tag{3}$$

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

$$y = \mathrm{s}_1^\alpha(x) \ \wedge \ \mathrm{Is}_\alpha(x) \ \wedge \ \mathrm{Is}_\beta(\mathrm{s}_1^\alpha(x), y) \ , \tag{4}$$

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

$$y = \mathrm{s}_1^\alpha(x) \ \wedge \ \mathrm{Is}_\alpha(x, \mathrm{s}_1^\alpha(x)) \ \wedge \ \mathrm{Is}_\beta(y) \ , \tag{5}$$

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

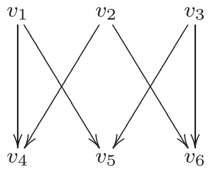
$$x \neq \alpha(t_1, \ldots, t_{\mathrm{ar}(\alpha)}) \;\wedge\; x \neq \beta(t'_1, \ldots, t'_{\mathrm{ar}(\beta)})$$

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.

4 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 [1] 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



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

Fig. 1. The DAG of (6).

without selectors. In fact, the convexity was shown implicitly in the correctness proof of Oppen's algorithm [1].

Definition 7 (Convexity) *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 $s \neq t \in \neg\Psi$, $\Phi \wedge \mathrm{s} \neq t$ 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 8 (DAG Representation of Terms) *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 1 shows the DAG for

$$\mathrm{cons}(y, z) = \mathrm{cons}(x, z) \ \wedge \ \mathrm{cons}(x, y) \neq z \ . \tag{6}$$

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.

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

Definition 9 (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 10 (Congruence Closure) *We say that R' is the* congruence closure *of R (denoted by $R{\upharpoonleft}$) 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 $R{\Updownarrow}$.

Let R be the set of all pairs asserted equal in Φ. It was shown that $R{\Updownarrow}$ represents all equalities logically implied by Φ [1]. Therefore, Φ is unsatisfiable if and only if there exist t and s such that $t \neq s \in \Phi$ and $(t,s) \in R{\Updownarrow}$.

Algorithm 1 (Oppen's Decision Procedure for $\text{Th}^{\forall}(\text{TA})$ [1]) *Input:*

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

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

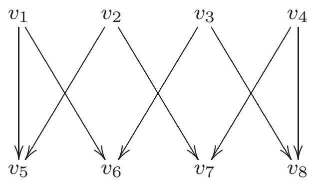
(1) Construct the DAG G of Φ.
(2) Compute on G the bidirectional closure $R{\Updownarrow}$ of $R = \{(q_i, r_i) \mid 1 \leq i \leq k\}$.
(3) Return FAIL *if $\exists i(s_i, t_i) \in R{\Updownarrow}$, or $(\exists (u,v) \in R{\Updownarrow})[\text{type}(u) \neq \text{type}(v)]$.*
Return SUCCESS *otherwise.*

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 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 problem is NP-complete [16]. The complication is that it is not known *a priori* whether $\text{s}(x)$ 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 1.

Algorithm 2 (Decision Procedure for $\text{Th}^{\forall}(\text{TA})$ 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 1 on Φ^{c}.



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

Fig. 2. The DAG of (7) under the assumption $\mathrm{Is}_{\mathrm{cons}}(x)$.

Example 11 *Figure 2 shows the DAG representation of the LISP list formula*

$$\mathrm{cons}(y, z) = \mathrm{cons}(\mathrm{cdr}(x), z) \ \wedge \ \mathrm{cons}(\mathrm{car}(x), y) \neq x \tag{7}$$

under the guess that x is not a constant. Initially $R = \{(v_3, v_4)\}$ as v_3 and v_4 are asserted equal in (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*

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

(Here we used the property that $R{\Downarrow} = (R{\downarrow}){\uparrow}$.) Obviously this branch fails as $v_1 \neq v_2$ is asserted by (7). *The remaining branch (with 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* (7).

Note that the correctness of both Algorithm 1 and 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 5.4) that our algorithm extends Oppen's algorithm to structures with finite constant domain.

5 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 5.1 we describe the theory, and then, in Section 5.2, outline our generic approach for constructing decision procedures for this theory. This approach is refined in Section 5.3 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 5.4. Section 5.5 outlines the approach to obtain decision procedures for structures whose constant domain contains relations besides equality. Section 5.6 discusses the complexity of the decision procedures for the quantifier-free theories.

Definition 12 *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* $|\cdot|$ *denotes the length function defined recursively by: (i) for any constant* a*,* $|a| = 1$*, and (ii) for a term* $\alpha(t_1, \ldots, t_k)$*,* $|\alpha(t_1, \ldots, t_k)| = \sum_{i=1}^{k} |t_i| + 1$*.*

The extended language is denoted by $\mathscr{L}_{\mathbb{T}}^{\mathbb{Z}}$*.*

The length function given in Definition 12 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.

We use subscripts $\mathbb{T}$, $\mathbb{Z}$ (or prefixes TA-, PA-) to denote notions related to term sort and integer sort, respectively. We also use "term" for "TA" when there is no confusion. 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 $\mathscr{L}_{\mathbb{T}}^{\mathbb{Z}}$ 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.

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, $\mathrm{car}(x)$ is a TA-term and $|\mathrm{car}(x)|$ is a PA-term. The first occurrence of $\mathrm{car}(x)$ is a term occurrence and the second one is an integer occurrence.

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

Example 13 *Consider the formula* $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) : x_1 = \mathrm{cons}(x_2, x_3)$*. In the context of this formula* $|\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}}$ *denotes*

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

Example 14 *In the context of the formula* $\Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}) \,:\, |x_1| \,=\, |\mathrm{car}(\mathrm{cdr}(x_2))|$*,*

$|\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}}$ *denotes*

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

Suppose $\Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is in a context in which all occurrences of $\bar{\boldsymbol{x}} \in \mathcal{V}_{\mathbb{T}}$ are integer occurrences. Then by $\Phi(\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}})$ we mean the formula obtained by substituting a true integer variable z ($z \in \bar{\boldsymbol{z}}$) for each pseudo-integer variable $|t(\bar{\boldsymbol{x}})|$. (Here we actually overload the symbol Φ, but the risk of confusion is minimal.) We use $|\bar{\boldsymbol{x}}| \leftarrow \bar{\boldsymbol{z}}$ to denote such a substitution. For example, in Example 14 $\Phi_{\mathbb{Z}}(\bar{\boldsymbol{z}})$ 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 $x_1 = \mathrm{cons}(x_2, x_3)$ and $\sigma_{\mathbb{T}}$ is $\{x_1 := \mathrm{nil}, x_2 := \mathrm{cons}(\mathrm{nil}, \mathrm{nil}), x_3 := \mathrm{nil}\}$, 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 [18] is not directly applicable to $\mathrm{TA}_{\mathbb{Z}}$ due to the presence of the length function.

Example 15 *The constraints*

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

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

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 15, Φ_{list} in fact implies $|x| \geq 2|\mathrm{car}(x)|$, 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 16 *Consider the constraint*

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

in List *and in* PA, *respectively. Clearly, both are satisfiable in the respective structures. In the combined structure* $\mathrm{List}_{\mathbb{Z}}$, *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.

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

$$(\forall \bar{\boldsymbol{x}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \;\rightarrow\; (\exists \bar{\boldsymbol{z}} : \mathbb{Z}) \Big(\Phi_{\Delta}(\bar{\boldsymbol{z}}) \;\wedge\; |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \Big) \Big] \;, \tag{8}$$

$$(\forall \bar{\boldsymbol{z}} : \mathbb{Z}) \Big[\Phi_{\Delta}(\bar{\boldsymbol{z}}) \;\rightarrow\; (\exists \bar{\boldsymbol{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \;\wedge\; |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \Big) \Big] \;. \tag{9}$$

Let Φ_{Δ} be an LCC for $\Phi_{\mathbb{T}}$. Condition (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 (8) is *sound* with respect to $\Phi_{\mathbb{T}}$. On the other hand, condition (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 (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 (8) and (9). Let $\Phi_{\Delta+}$ and $\Phi_{\Delta-}$, respectively, be any two formulas satisfying (8) and (9) (when in place of Φ_{Δ}). If we identify these constraints with their corresponding solution sets, we have

$$\Phi_{\Delta-} \;\subseteq\; \Phi_{\Delta} \;\subseteq\; \Phi_{\Delta+} \;. \tag{10}$$

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 (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 18 *Consider in* List *the formulas* $\Phi_{\mathrm{list}} : \mathrm{cons}(x, y) = z$*. 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 $\text{Th}^{\forall}(\text{TA}_{\mathbb{Z}})$ if Φ_{Δ} can be effectively computed from $\Phi_{\mathbb{T}}$.

Theorem 19 ([3]) *Let* Φ_{Δ} *be an LCC for* $\Phi_{\mathbb{T}}$. *Then* $\text{TA}_{\mathbb{Z}} \models_{\exists} \Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$ *if and only if* $\text{PA} \models_{\exists} \Phi_{\Delta} \wedge \Phi_{\mathbb{Z}}$.

PROOF. Conditions (8) and (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 17 to deal with newly generated integer constraints in the normalization.

Definition 20 (Relativized LCC (RLCC)) *A formula* $\Phi_{\Delta}(\bar{\boldsymbol{x}})$ *is a* length constraint completion *for* $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}})$ *relativized to* $\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}})$, *(in short,* $\Phi_{\Delta}(\bar{\boldsymbol{x}})$ *is an RLCC for* $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}})/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}})$*), if the following formulas are valid:*

$$(\forall \bar{\boldsymbol{x}} : \mathbb{T}) \left[\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}) \ \rightarrow \ (\exists \bar{\boldsymbol{z}} : \mathbb{Z}) \left(\Phi_{\Delta}(\bar{\boldsymbol{z}}) \ \wedge \ |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \right) \right] \ , \qquad (11)$$

$$(\forall \bar{\boldsymbol{z}} : \mathbb{Z}) \left[\Phi_{\Delta}(\bar{\boldsymbol{z}}) \ \rightarrow \ (\exists \bar{\boldsymbol{x}} : \mathbb{T}) \left(\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}) \ \wedge \ |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \right) \right] \ . \qquad (12)$$

Example 21 *Consider* List. *Let*

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

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 $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}})/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}})$. However, neither $\Phi_{\Delta+}$ nor $\Phi_{\Delta-}$ is such an RLCC. Although $\Phi_{\Delta+}$ satisfies (11)*, it does not satisfy* (12)*, as the assignment*

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

can not be realized by any assignment for $\bar{\boldsymbol{x}}$. On the other hand, $\Phi_{\Delta-}$ satisfies (12)*, but not* (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 (8) and (9) with (11) and (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 $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}})/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}})$. In addition, RLCCs have the following easily proved "additive" property.

Proposition 22 *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 $\theta_{\mathbb{Z}} := \mathrm{true}$, 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 19 is in fact a special case of the following theorem.

Theorem 23 *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 (11) and (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 22, $\Phi^{(i)}_\Delta \wedge \Phi_{\mathbb{Z}}$ 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 23 (set $\Phi_{\mathbb{T}} := \Phi^{(i)}_{\mathbb{T}}$, $\Phi_\Delta := \Phi^{(i)}_\Delta \wedge \Phi_{\mathbb{Z}}$,

$\Phi_{\mathbb{Z}} := \Phi_{\mathbb{Z}} \wedge \theta_{\mathbb{Z}}^{(i)}$), reduces to determining whether PA $\models_{\exists} \Phi_{\Delta}^{(i)} \wedge \Phi_{\mathbb{Z}}$. Note that $\Phi_{\mathbb{Z}}$ is not involved in computing an RLCC. Therefore, we can assume that $\theta_{\mathbb{Z}}^{(i)}$ includes constraints relevant to the corresponding partitioning and other constraints generated during the normalization procedure have been merged into $\Phi_{\mathbb{Z}}$.

Algorithm 3 (Generic Decision Procedure) *Input:* $\Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$.

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

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

The easiest arithmetic extension of term algebras is $\mathrm{Th}^{\forall}(\mathrm{TA}_{\mathbb{Z}}^{\omega})$, the quantifier-free theory of term algebras with integers and with an infinite constant domain. In $\mathrm{TA}_{\mathbb{Z}}^{\omega}$, 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:

$$\mathrm{Tree}(x) \quad : \quad \exists x_1, \ldots, x_n \geq 0 \left(x = 1 + \sum_{i=1}^{n} d_i x_i \right) , \tag{13}$$

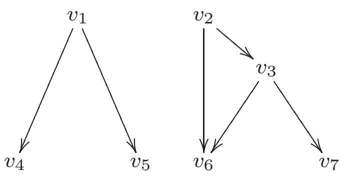
$$\mathrm{Node}^{\alpha}(x, \bar{\boldsymbol{x}}_{\alpha}) \quad : \quad x = 1 + \sum_{i=1}^{\mathrm{ar}(\alpha)} x_i \ , \tag{14}$$

$$\mathrm{Tree}^{\alpha}(x) \quad : \quad \exists \bar{x}_{\alpha} \Big(\mathrm{Node}^{\alpha}(x, \bar{\boldsymbol{x}}_{\alpha}) \wedge \bigwedge_{i=1}^{\mathrm{ar}(\alpha)} \mathrm{Tree}(x_i) \Big) \ , \tag{15}$$

where $\bar{\boldsymbol{x}}_{\alpha}$ 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 4 (Computation of LCC in $\mathrm{TA}_{\mathbb{Z}}^{\omega}$**)** *Let* $\Phi_{\mathbb{T}}$ *be a type-complete term constraint,* $G_{\mathbb{T}}$ *the DAG of* $\Phi_{\mathbb{T}}$ *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 a constant or asserted to be a constant (i.e.,* $\mathrm{Is}_{\mathsf{A}}(t)$ *is in* $\Phi_{\mathbb{T}}$*);*



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

Fig. 3. The DAG of (16) under the assumption $\mathrm{Is}_{\mathrm{cons}}(y)$

- $|t| = |s|$, *if* $(t, s) \in R\upharpoonright$*;*
- $\mathrm{Node}^{\alpha}(|t|, |t_1|, \ldots, |t_{\mathrm{ar}(\alpha)}|)$*, 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 24 Φ_{Δ} *obtained by Algorithm 4 is expressible in a quantifier-free Presburger formula linear in the size of* $\Phi_{\mathbb{T}}$*.*

Theorem 25 Φ_{Δ} *obtained by Algorithm 4 is an LCC for* $\Phi_{\mathbb{T}}$*.*

Algorithm 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 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 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 19 and Theorem 25.

Example 26 *Figure 3 shows the DAG of*

$$x = \mathrm{cons}(\mathrm{car}(y), y) \;\wedge\; |\mathrm{cons}(\mathrm{car}(y), y)| < 2|\mathrm{car}(x)| \;, \tag{16}$$

assuming that y *is not a constant. The computed* $R\upharpoonright$ *is* $\{ (v_1, v_2), (v_3, v_5), (v_4, v_6) \}$*.*

By Algorithm 4 Φ_Δ *is*

$$
\begin{gathered}
|x| = |\mathrm{cons}(\mathrm{car}(y), y)| \ \wedge\ |\mathrm{car}(x)| = |\mathrm{car}(y)| \ \wedge\ |\mathrm{cdr}(x)| = |y| \\
\wedge \\
|x| = |\mathrm{car}(x)| + |\mathrm{cdr}(x)| + 1 \ \wedge\ |y| = |\mathrm{car}(y)| + |\mathrm{cdr}(y)| + 1 \\
\wedge \\
|\mathrm{cons}(\mathrm{car}(y), y)| = |\mathrm{car}(y)| + |y| + 1 \\
\wedge \\
2 \nmid |\mathrm{car}(x)| \ \wedge\ 2 \nmid |\mathrm{cdr}(x)| \ \wedge\ 2 \nmid |\mathrm{car}(y)| \ \wedge\ 2 \nmid |\mathrm{cdr}(y)| \\
\wedge \\
|\mathrm{car}(x)| \geq 1 \ \wedge\ |\mathrm{cdr}(x)| \geq 1 \ \wedge\ |\mathrm{car}(y)| \geq 1 \ \wedge\ |\mathrm{cdr}(y)| \geq 1
\end{gathered}
\tag{17}
$$

which implies

$$
|\mathrm{cons}(\mathrm{car}(y), y)| \geq 2|\mathrm{car}(x)| \ . \tag{18}
$$

contradicting $|\mathrm{cons}(\mathrm{car}(y), y)| < 2|\mathrm{car}(x)|$. *If* y *is a constant,* v_3, v_6, v_7 *are merged. In this case also* Φ_Δ *implies* (18), *and therefore* (16) *is unsatisfiable.*

Note that the last two lines of (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.*

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

Algorithm 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 27 *Consider* List *with* $\mathcal{A} = \{\mathrm{nil}\}$. *The constraint*

$$
|x| = 5 \ \wedge\ \mathrm{Is}_{\mathsf{A}}(y) \ \wedge\ x \neq \mathrm{cons}(\mathrm{cons}(y, y), y) \ \wedge\ x \neq \mathrm{cons}(y, \mathrm{cons}(y, y)) \tag{19}
$$

is unsatisfiable while Φ_Δ *obtained by Algorithm 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 28 (Counting Constraint) *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 29 *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 $m \geq n > 0$,

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

In general we have the following result.

Proposition 30 $\mathrm{CNT}^\alpha_{k,n}(x)$ *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 31 (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 32 *Let* Φ_{list} *be* $y \neq \mathrm{cons}(x, z) \wedge \mathrm{Is}_{\mathrm{cons}}(x, y, z)$*. A possible equality* *completion of* Φ_{list} $(\theta_{\mathbb{Z}} = \emptyset)$ *is*

$$\begin{aligned} |y| = |\text{cons}(x,z)| \ \wedge \ |x| = |z| \ \wedge \ |y| \neq |x| \\ \wedge \ \text{Is}_{\text{cons}}(x,y,z) \ \wedge \bigwedge_{t,t' \in \mathcal{S}; t \not\equiv t'} t \neq t' \ , \end{aligned} \tag{20}$$

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

Strictly speaking, the formula (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. However, it 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 are pairwise distinct.

Algorithm 6 (Computation of an 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 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 33 Φ_{Δ} *obtained by Algorithm 6 is expressible in a quantifier-free Presburger formula of size linear in the size of* $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$.

Theorem 34 Φ_{Δ} *obtained by Algorithm 6 is an RLCC for* $\Phi_{\mathbb{T}}/\theta_{\mathbb{Z}}$.

Algorithm 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 1 on $\Phi_{\mathbb{T}}$. *Return* FAIL *if* $\text{TA} \not\models_{\exists} \Phi_{\mathbb{T}}$.
(3) Construct Φ_{Δ} *from* $\Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$ *using Algorithm 6.*
- *Return* SUCCESS *if* $\text{PA} \models_{\exists} \Phi_{\Delta} \wedge \Phi_{\mathbb{Z}}$.
- *Return* FAIL *otherwise.*

The correctness of Algorithm 7 follows from Theorems 23 and 34. 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 35 *Let us return to Example 27. Constraint* (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 the counting constraint $\mathrm{CNT}_{1,2}^{\mathrm{cons}}(|x|) : \ |x| > 5 \wedge 2 \nmid |x|$, *contradicting* $|x| = 5$ *in* (19).

5.5 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 Nelson-Oppen combination method that $\mathrm{TA}_{\mathbb{Z}}^{+} \models_{\exists} \Phi^{+}$ if and only if $\mathrm{TA}_{\mathbb{Z}} \models_{\exists} \Phi$ and $\mathfrak{A}_c \models_{\exists} \Phi_c$.

5.6 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 [1], [35, pp. 336-340]. Second, Algorithm 4 computes Φ_{Δ} in $O(n)$ (see Proposition 24) and so does Algorithm 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 5 and 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.

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.

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

$$(\exists \bar{\boldsymbol{x}}) \left[A_1(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \dots \ \wedge \ A_n(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \right] \ , \tag{21}$$

where $A_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ $(1 \leq i \leq n)$ are literals [2]. We can assume that the literals A_i are not of the form $x = t$ when x does not appear inside selectors. For $\exists x(x = t \wedge \Phi(x, \bar{\boldsymbol{y}}))$ simplifies to $\Phi(t, \bar{\boldsymbol{y}})$ if x does not occur in t, to $\exists x \Phi(x, \bar{\boldsymbol{y}})$ 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 each step to restore the normal form.

Nondeterminism. In the rest of this paper all transformations are done on formulas of the form (21). 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 (21). 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 (21) "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 (21) 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: $\bar{\boldsymbol{x}}$ denote the set of existentially quantified variables; $\bar{\boldsymbol{y}}$ 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 $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ 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 $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ 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 $\exists \bar{\boldsymbol{x}}$ can be eliminated by removing all literals containing $\bar{\boldsymbol{x}}$ in the matrix.

Definition 36 (Solved Form) *We say $\Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is* solved *in $\bar{\boldsymbol{x}}$, if $\bar{\boldsymbol{x}}$ do not appear in equalities, are not asserted to be constants and are not inside selector terms. We say $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is in solved form if $\Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is solved in $\bar{\boldsymbol{x}}$.*

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 8 (Normalization in TA**)** *Input: $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$.*

(1) Type Completion. Guess a type completion of $\Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ and simplify every selector term to a proper one.
(2) Selector Elimination. Replace all selector terms containing $\bar{\boldsymbol{x}}$ by the corresponding equivalent constructor terms according to Axiom (A6).
(3) Decomposition. Call Algorithm 9 to decompose equalities and disequalities between constructor terms and equalities containing $\bar{\boldsymbol{x}}$.
(4) Constant Elimination. If some $x \in \bar{\boldsymbol{x}}$ 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 8 terms can have three forms:

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

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 $t_i(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}})$ of the form (i), (ii) or (iii). These three forms give rise to the following six types of equality literals:

$$Lx = L'x'\ , \tag{22}$$
$$Lx = L'y\ , \tag{23}$$
$$Lx = \alpha(t_1(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}),\ldots,t_{\mathrm{ar}(\alpha)}(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}))\ , \tag{24}$$
$$Ly = L'y'\ , \tag{25}$$
$$Ly = \alpha(t_1(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}),\ldots,t_{\mathrm{ar}(\alpha)}(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}))\ , \tag{26}$$
$$\alpha(t_1(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}),\ldots,t_{\mathrm{ar}(\alpha)}(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}})) = \alpha(t'_1(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}),\ldots,t'_{\mathrm{ar}(\alpha)}(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}))\ . \tag{27}$$

Similarly, we have six types of disequalities, the negations of (22)-(27).

Step (2) transforms equalities of the forms (22-24), and similarly for disequalities of the same form. Thus after application of this step, we can assume that $\bar{\boldsymbol{x}}$ does not appear inside selector terms, that is, equality literals have the forms

$$Ly = L'y'\ , \tag{28}$$
$$Ly = \alpha(t_1(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}),\ldots,t_{\mathrm{ar}(\alpha)}(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}))\ , \tag{29}$$
$$\alpha(t_1(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}),\ldots,t_{\mathrm{ar}(\alpha)}(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}})) = \alpha(t'_1(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}),\ldots,t'_{\mathrm{ar}(\alpha)}(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}}))\ , \tag{30}$$

and disequality literals are in the forms of the negations of (28)-(30) and in the forms of

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

Step (2) may generate literals like $x = t(\bar{\boldsymbol{x}},\bar{\boldsymbol{y}})$. The reinstatement of normal form, however, does not put any of $\bar{\boldsymbol{x}}$ 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 37 *Step 2, selector elimination, first converts the formula*

$$(\exists x)\Big[\ \mathrm{car}(x) = y_2\ \wedge\ \mathrm{cdr}(x) \neq y_2\ \wedge x\ \neq y_3\ \Big] \tag{31}$$

into

$$(\exists x_1 \exists x_2)\Big[\ x_1 = y_2\ \wedge\ x_2 \neq y_2\ \wedge\ \mathrm{cons}(x_1,x_2) \neq y_3\ \Big]\ , \tag{32}$$

which, by substitution of x_1 for y_2, simplifies to

$$(\exists x_2)\left[\ x_2 \neq y_2 \ \wedge\ \text{cons}(y_2, x_2) \neq y_3\ \right]\ . \tag{33}$$

Step (3) converts equalities and disequalities of form (30) and equalities of form (29). After step (3) the matrix, if it did not simplify to false, contains only literals of the form

$$x \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})\ , \qquad Ly \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})\ , \qquad Ly = L'y'\ . \tag{34}$$

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

Algorithm 9 (Decomposition)

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

$$\alpha(t_1, \ldots, t_{\text{ar}(\alpha)}) = \alpha(t'_1, \ldots, t'_{\text{ar}(\alpha)}) \tag{35}$$

by $\bigwedge_{1 \leq i \leq \text{ar}(\alpha)} t_i = t'_i$*. Repeat until no equality of the form* (35) *appears.*

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

$$\alpha(t_1, \ldots, t_{\text{ar}(\alpha)}) \neq \alpha(t'_1, \ldots, t'_{\text{ar}(\alpha)}) \tag{36}$$

by $\bigvee_{1 \leq i \leq \text{ar}(\alpha)} t_i \neq t'_i$*. Repeat until no disequality of the form* (36) *appears.*

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

In step (1) of Algorithm 9, recall that literals like $x = t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ 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:

$$Ly = L'y'\ , \tag{37}$$
$$Ly \neq L'y'\ , \tag{38}$$
$$x \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})\ , \tag{39}$$
$$Ly \neq \alpha(t_1, \ldots, t_{\text{ar}(\alpha)})\ , \tag{40}$$
$$Ly = \alpha(t_1, \ldots, t_{\text{ar}(\alpha)})\ . \tag{41}$$

Step (3) solves equalities of the form (41), and thus we are left only with literals of the forms (37)-(40). Notice that these are the same as those in (34), where $Ly \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ represents both (38) and (40).

The following example illustrates how equalities are solved.

Example 38 *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 $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, the normalization procedure in Algorithm 8 lets us effectively eliminate all quantifiers $(\exists \bar{\boldsymbol{x}})$ in one step. Indeed, after application of Algorithm 8 we can assume, according to (34), that $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is in the form

$$\begin{aligned} (\exists \bar{\boldsymbol{x}}) \left[\bigwedge_i x_{p(i)} \neq t_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \right] \\ \wedge \ \bigwedge_k F_k y_{f(k)} \neq u_k(\bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_l G_l y_{g(l)} = H_l y_{h(l)} \ . \end{aligned} \tag{42}$$

Here t_i, s_j are:

(i) existentially quantified variables $\bar{\boldsymbol{x}}$,
(ii) implicitly universally quantified parameters $\bar{\boldsymbol{y}}$,
(iii) selector terms of parameters in the form Ly ($y \in \bar{\boldsymbol{y}}$),
(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 $\bar{\boldsymbol{x}}$ (otherwise, the corresponding literal should have been moved out of the scope of $(\exists \bar{\boldsymbol{x}})$). The term $u_k(\bar{\boldsymbol{y}})$ can be one of (ii)-(v) above, where in (v) recursion is limited to (ii)-(v).

We claim that the first part of (42),

$$(\exists \bar{\boldsymbol{x}}) \left[\bigwedge_i x_{p(i)} \neq t_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \right] , \tag{43}$$

is valid, and hence (42) is equivalent to

$$\bigwedge_k F_k y_{f(k)} \neq u_k(\bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_l G_l y_{g(l)} = H_l y_{h(l)} \ . \tag{44}$$

Thus the algorithm for elimination of quantifiers can be given as

Algorithm 10 (Elimination of Quantifiers) *Input:* $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$.

(1) Call Algorithm 8 to normalize $(\exists \bar{\boldsymbol{x}})\, \Phi(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$.
(2) Remove $(\exists \bar{\boldsymbol{x}})$ *and all literals containing* $\bar{\boldsymbol{x}}$.

Theorem 39 *All transformations in Algorithm 10 preserve equivalence.*

Theorem 40 *Algorithm 10 removes a block of quantifiers in time* $2^{O(n)}$.

Example 41 *Let us look at an example in* List. *Consider a type complete formula*

$$(\exists x) \left[\text{ cons}(\text{car}(x), y_1) = y_2 \ \wedge \ y_2 \neq x \right] . \tag{45}$$

(Because of the assumption of type completeness, we know $\text{Is}_{\text{cons}}(y_2)$ *and* $\text{Is}_{\text{cons}}(x)$ *must be present in the side condition. We omit type information of* y_1*, because it is irrelevant here.) Step* (2) *of Algorithm 8 removes the selectors on* x *by converting* (45) *into*

$$(\exists x_1 \exists x_2) \left[\text{ cons}(x_1, y_1) = y_2 \ \wedge \ y_2 \neq \text{cons}(x_1, x_2) \right] . \tag{46}$$

Step (3) *of Algorithm 8 (Algorithm 9) solves the equality* $\text{cons}(x_1, y_1) = y_2$*, with solution set*

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

resulting in

$$(\exists x_1 \exists x_2) \left[\text{ car}(y_2) = x_1 \ \wedge \ \text{cdr}(y_2) = y_1 \ \wedge \ y_2 \neq \text{cons}(x_1, x_2) \right] , \tag{47}$$

which, by standard substitution and quantifier manipulation, reduces to

$$(\exists x_2) \left[\ y_2 \neq \text{cons}(\text{car}(y_2), x_2) \right] \ \wedge \ \text{cdr}(y_2) = y_1 . \tag{48}$$

As $(\exists x_2)[\, y_2 \neq \text{cons}(\text{car}(y_2), x_2)\,]$ *is in solved form, it is valid, and hence* (48) *reduces to* $\text{cdr}(y_2) = y_1$*, or more formally, to* $\text{cdr}(y_2) = y_1 \wedge \text{Is}_{\text{cons}}(y_2)$.

7 Decision Procedures for Quantified Theories

In this section we present decision procedures for the theory $\text{Th}(\text{TA}_{\mathbb{Z}})$ of term algebras with integers and parameters. In Section 7.1 we first refine the notions and techniques from Section 5 for the construction of a quantifier elimination procedure for $\text{Th}(\text{TA}_{\mathbb{Z}})$, and then, in Section 7.2, present the quantifier elimination procedure itself. Throughout Sections 7.1 and 7.2 we assume a finite language. In Section 7.3 we further generalize the result to structures whose constant domain has an internal structure and admits quantifier elimination.

In Section 7.4 we discuss how the procedure can be adapted for infinite languages.

7.1 Term Algebras with Integers and Parameters

In this section we refine the notions and techniques from Section 5 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 13). 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 42 (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,

$$t = s \;\wedge\; s = u \;\wedge\; u \neq v \;\wedge\; |t| = |v| \;\wedge\; \mathrm{Is}_{\alpha}(t, s, u, v), \tag{49}$$

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 13.

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 43 *Consider again the formula from Example 32,*

$$\begin{aligned} |y| = |\mathrm{cons}(x,z)| \ \wedge \ |x| = |z| \ \wedge \ |y| \neq |x| & \\ \wedge \ \mathrm{Is}_{\mathrm{cons}}(x,y,z) \ \wedge \bigwedge_{t,t' \in \mathcal{S}; t \not\equiv t'} t \neq t' \ , & \end{aligned} \tag{50}$$

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 44 *The* $\mathrm{List}_{\mathbb{Z}}$ *formula* Φ :

$$x \neq y \ \wedge \ x \neq z \ \wedge \ x \neq w \ \wedge \ |x| = |y| \ \wedge \ |x| = |z| \ \wedge \ \mathrm{Is}_{\mathrm{cons}}(x,y,z,w) \tag{51}$$

gives three maximal cons*-clusters*

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

C_3 *is closed, but neither* C_1 *nor* C_2 *is. If we conjoin* (51) *with* $y \neq z$, C_1 *and* C_2 *are merged, making the resulting cluster* $\{\, x, \ y, \ z \,\}$ *maximal. If we conjoin* (51) *with* $y = z$, *then* C_1 *and* C_2 *become identical. Neither strengthening, however, results in a cluster completion of* (51)*, 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 43, 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,

$$\bigwedge_{i=1}^{n} \Big(|x_i| = |y_i| \ \wedge \ x_i \neq y_i \ \wedge \ \mathrm{Is}_\alpha(x_i, y_i)\Big),$$

is not equality complete, but induces n maximal and mutually independent α-clusters, namely $\{\,\{x_i, y_i\} \mid 1 \le i \le n\,\}$. 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 45 (RLCC with parameters) *Consider*

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

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

$$(\forall \bar{\boldsymbol{x}}, \bar{\boldsymbol{y}} : \mathbb{T}) \left[\, \Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \rightarrow (\exists \bar{\boldsymbol{z}} : \mathbb{Z}) \Big(\Phi_\Delta(\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}}) \wedge |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}}\Big) \,\right] \ , \tag{52}$$

$$\begin{aligned}(\forall \bar{\boldsymbol{y}} : \mathbb{T})(\forall \bar{\boldsymbol{z}} : \mathbb{Z}) \Big[\, & \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \ \wedge \ \Phi_\Delta(\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}}) \\ & \rightarrow (\exists \bar{\boldsymbol{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}}\Big) \,\Big] \ .\end{aligned} \tag{53}$$

In the viewpoint of logical equivalence, we can assume $\Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}})$ is empty as it can be moved out of the scope of $(\exists \bar{\boldsymbol{x}})$. 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 $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ be cluster complete and in *strongly solved form*.

Definition 46 (Strongly Solved Form) *We say that $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is* strongly solved *in $\bar{\boldsymbol{x}}$ if $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is solved in $\bar{\boldsymbol{x}}$ and all literals of the form $Ly \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, where $y \in \bar{\boldsymbol{y}}$ and $t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is a constructor term (properly) containing $\bar{\boldsymbol{x}}$, are redundant. We say that $(\exists \bar{\boldsymbol{x}} : \mathbb{T})[\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})]$ is in* strongly solved form *if $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is strongly solved in $\bar{\boldsymbol{x}}$.*

Example 47 *The formula* $\Phi_{\text{list}} \wedge \theta_{\mathbb{Z}}$

$$\begin{aligned} \text{Is}_{\text{cons}}(y) \;\wedge\; y \neq \text{cons}(x,z) \;\wedge \bigwedge_{t,t' \in \mathcal{S}; t \not\equiv t'} t \neq t' \\ \wedge\; |y| = |\text{cons}(x,z)| \;\wedge\; |x| = |z| \;\wedge\; |y| \neq |x| \;, \end{aligned} \tag{54}$$

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

Recall that $\text{CLS}^{\alpha}_{n}(x_0, x_1, \ldots, x_n)$ states that $x_0, \ldots, x_n$ is a α-cluster of $n+1$ elements. We claim that Algorithm 6 computes an RLCC (with parameters $\bar{\boldsymbol{y}}$ being treated as ordinary variables).

Theorem 48 *If* $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ *is strongly solved in* $\bar{\boldsymbol{x}}$ *and cluster complete, then* $\Phi_{\Delta}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ *computed by Algorithm 6 is an RLCC for* $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$.

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

In this section we expand Algorithm 8 to an elimination procedure for Th($\text{TA}_{\mathbb{Z}}$), the first-order theory of term algebras with integers. Since $\mathscr{L}^{\mathbb{Z}}_{\mathbb{T}}$ has two sorts, namely $\mathbb{Z}$ and $\mathbb{T}$, we need to show elimination of integer quantifiers as well as term quantifiers.

7.2.1 Elimination of Quantifiers on Integer Variables

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

$$(\exists \bar{\boldsymbol{z}} : \mathbb{Z}) \Big[\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \;\wedge\; \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \Big] \;, \tag{55}$$

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

$$\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \;\wedge\; (\exists \bar{\boldsymbol{z}} : \mathbb{Z})\, \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \;. \tag{56}$$

For this reason, neither $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}})$ nor $\Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ is required to be quantifier-free. Now we can put $(\exists \bar{\boldsymbol{z}} : \mathbb{Z})\, \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ into a quantifier-free form using Cooper's method [6,7]. For the sake of efficiency, however, we can defer the actual elimination on $(\exists \bar{\boldsymbol{z}} : \mathbb{Z})\, \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ 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 (57) to be quantifier-free.

7.2.2 Elimination of Quantifiers on Term Variables

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

$$(\exists \bar{\boldsymbol{x}} : \mathbb{T}) \left[\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \right] , \tag{57}$$

where $\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}$ are term variables, $\bar{\boldsymbol{z}}$ are integer variables, $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is quantifier-free. The following algorithm is based on Algorithm 8.

Algorithm 11 (Normalization with Parameters) *Input:* (57). *Apply the following subprocedures successively.*

(1) Basic Normalization. Apply Algorithm 8 to $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ and update $\Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ accordingly.
(2) Cluster Completion. Normalize (57) *to a cluster completion (Algorithm 12).*
(3) Decomposition of Disequalities. Decompose disequalities of the Form $Ly \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ (Algorithm 13).

The purpose of steps (1-3) is to transform (57) into a formula which is in strongly solved form and cluster complete, such that we can use Algorithm 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) (43) (copied below)

$$(\exists \bar{\boldsymbol{x}} : \mathbb{T}) \left[\bigwedge_i x_{p(i)} \neq t_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \right] . \tag{43}$$

Algorithm 12 produces cluster completions of the input by merging mutually dependent maximal clusters.

Algorithm 12 (Cluster Completion) *Input:* (43). *Apply the following subprocedures 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 9 to $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ and update $\Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ accordingly.

The termination of Algorithm 12 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 9) is called afterwards to restore the normal form. Clearly, steps (1-2) of Algorithm 9 do not introduce new terms. Step (3) of Algorithm 9 solves equalities containing existentially quantified variables $\bar{\boldsymbol{x}}$ in terms of parameters $\bar{\boldsymbol{y}}$. However, it will only introduce new terms into existing equivalence classes. In particular, it removes at least one variable in $\bar{\boldsymbol{x}}$ 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 (43). It is easily seen that the resulting formula is still in the form of (43) (omitting integer literals).

Algorithm 13 decomposes disequalities in $\Phi_{\mathbb{T}}$ that are of the form $Ly \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ such that $|Ly| = |t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})|$ is implied by $\Phi_{\mathbb{T}}$ and $t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is a constructor term (properly) containing $\bar{\boldsymbol{x}}$. The decomposition consists of a sequence of disjunctive splittings, where in each step the matrix of (57) is updated accordingly. It is assumed that when Algorithm 13 is called, the matrix $\Phi_{\mathbb{T}} \wedge \Phi_{\mathbb{Z}}$ of (57) is cluster complete. The algorithm preserves this completeness: all resulting branches are cluster complete.

Algorithm 13 (Decomposition of Disequalities Containing $\bar{\boldsymbol{x}}$) *Input: Set $\mathcal{D}$ of disequalities of the form $Ly \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ such that $|Ly| = |t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})|$, in the context of* (57).

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

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

$$\mathcal{D}' = \{\, Ly \neq \alpha(t_1^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}), \ldots, t_k^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})) \;\mid\; 1 \leq i \leq m \,\}$$

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

$$\bigwedge_{i=1}^{m} \Big(\mathrm{s}_{p(i)}^{\alpha} Ly \neq t_{p(i)}^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \;\wedge \bigwedge_{1 \leq j < p(i)} \mathrm{s}_j^{\alpha} Ly = t_j^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \Big) \;, \tag{58}$$

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*

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

For any $i \in I$, replace

$$\mathrm{s}_{p(i)}^{\alpha} Ly \neq t_{p(i)}^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \qquad by \qquad t_{p(i)}^{(\delta)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \neq t_{p(i)}^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \;.$$

(2) Cluster Completion. *Call Algorithm 12 to restore cluster completeness.*

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

$$\bigvee_{1\leq j\leq k} \Big(\mathrm{s}_j^\alpha Ly \neq t_j^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \bigwedge_{1\leq l<j} \mathrm{s}_l^\alpha Ly = t_l^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \Big)$$

instead of more directly to

$$\bigvee_{1\leq j\leq k} \Big(\mathrm{s}_j^\alpha Ly \neq t_j^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \Big) \ .$$

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 ($\mathrm{s}_{p(\delta)}^\alpha Ly$) is asserted unequal to old terms that may contain $\bar{\boldsymbol{x}}$ 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

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

with a new set of disequalities

$$\{\, \mathrm{s}_{p(\delta)}^\alpha Ly \neq t_{p(\delta)}^{(i)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \mid\ i \in I \,\} \ .$$

Clearly, $\mathrm{s}_{p(\delta)}^\alpha Ly$ 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 13 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 13.

Besides the rewriting in step (1a), we also need to prevent step (2) (Algorithm 12) from generating unwanted disequalities. This can be done by requiring, upon calling Algorithm 12, that newly generated terms are not made representatives for existing equivalence classes.

Example 49 *To illustrate step* (1)*, assume* $\alpha = (\mathrm{s}_1^\alpha, \mathrm{s}_2^\alpha, \mathrm{s}_3^\alpha, \mathrm{s}_4^\alpha)$*, 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:*

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

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

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

In this case only $\mathrm{s}_4^\alpha y$ *(among the newly generated terms), having a lower rank than* y*, is part of a disequality that may need to be split further.*

Proposition 50 *Algorithm 11 produces a formula that is in strongly solved form and cluster complete.*

After normalization (Algorithm 11) and removal of redundant disequalities, the resulting formula is in the form

$$(\exists \bar{\boldsymbol{x}} : \mathbb{T}) \left[\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \;\wedge\; \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \;\wedge\; \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \;\wedge\; \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \right] , \tag{59}$$

where $\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is of the form $\bigwedge_i x_{f(i)} \neq t_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ and $\Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}})$ does not contain $\bar{\boldsymbol{x}}$. $\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is the integer constraint obtained from runs of Algorithm 12 (step (2) of Algorithm 11 and step (2) of Algorithm 13). The resulting formula is guaranteed to be in strongly solved form and cluster complete, and hence by Theorem 48, we can compute an RLCC $\Phi_\Delta(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ for $\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, producing the equivalent

$$(\exists \bar{\boldsymbol{x}} : \mathbb{T}) \left[\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \;\wedge\; \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \;\wedge\; \Phi_\Delta(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \;\wedge\; \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \right] , \tag{60}$$

which reduces to

$$\Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \;\wedge\; (\exists \bar{\boldsymbol{u}} : \mathbb{Z}) \left[\Phi_\Delta(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}) \;\wedge\; \Phi_{\mathbb{Z}}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \right] , \tag{61}$$

because $\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ has been completely characterized by Φ_Δ, and all occurrences of $\bar{\boldsymbol{x}}$ 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 14 (Reduction of Term Quantifiers to Integer Quantifiers)
Input: (57)

(1) Call Algorithm 11 to normalize the input, obtaining (59)*.*
(2) Call Algorithm 6 to get the RLCC $\Phi_\Delta(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ *for* $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ *and add it to the matrix of* (59)*, obtaining* (60)*.*
(3) Remove $\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ *from the matrix of* (60)*, pull* $\Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ *out of* $(\exists \bar{\boldsymbol{x}} : \mathbb{T})$ *and then reduce* $(\exists \bar{\boldsymbol{x}} : \mathbb{T})$ *to* $(\exists \bar{\boldsymbol{z}} : \mathbb{Z})$*, obtaining* (61)*.*

Theorem 51 *All transformations in Algorithm 14 preserve equivalence.*

Theorem 52 *Algorithm 14 eliminates a block of quantifiers in time* $2^{O(n^2 \lg n)}$.

Example 53 *Let us modify Example 41 by conjoining the matrix of* (45) *with an integer constraint* $\varphi_{\mathbb{Z}}(x)$. *(The concrete form of* $\varphi_{\mathbb{Z}}(x)$ *is irrelevant.) Now we have*

$$(\exists x : \text{list}) \left[\text{cons}(\text{car}(x), y_1) = y_2 \ \wedge \ y_2 \neq x \ \wedge \ \varphi_{\mathbb{Z}}(x) \right] . \tag{62}$$

Running Algorithm 8 on (62) *and updating integer constraints accordingly, we arrive at*

$$\begin{aligned} \text{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \text{list}) \Big[\ & y_2 \neq \text{cons}(\text{car}(y_2), x_2) \\ & \wedge \ \varphi_{\mathbb{Z}}(|\text{car}(y_2)| + |x_2| + 1) \Big] \end{aligned} \tag{63}$$

(cf. (48)*). Below we will follow a single branch produced in the disjunctive splittings.*

We run Algorithm 12 to obtain a compatible cluster completion of (63)*, producing*

$$\begin{aligned} \text{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \text{list}) \Big[\ & y_2 \neq \text{cons}(\text{car}(y_2), x_2) \\ & \wedge \ |y_2| = |\text{car}(y_2)| + |x_2| + 1 \ \wedge \ \varphi_{\mathbb{Z}}(|\text{car}(y_2)| + |x_2| + 1) \Big] . \end{aligned} \tag{64}$$

Here we omitted obvious disequalities implied by literals listed. The matrix of (64) *induces three mutually independent clusters:*

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

The formula (64)*, however, is not in strongly solved form as* x_2 *appears in the disequality* $y_2 \neq \text{cons}(\text{car}(y_2), x_2)$*, and hence we need to run Algorithm 13. Choosing* $\text{cdr}(y_2) \neq x_2$ *in step* (1a)*, we obtain*

$$\begin{aligned} \text{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \text{list}) \Big[\ & \text{cdr}(y_2) \neq x_2 \ \wedge \ |y_2| = |\text{car}(y_2)| + |x_2| + 1 \\ & \wedge \ \varphi_{\mathbb{Z}}(|\text{car}(y_2)| + |x_2| + 1) \Big] , \end{aligned} \tag{65}$$

which induces four clusters:

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

that are not mutually independent as neither $\{x_2\}$ *nor* $\{\text{cdr}(y_2)\}$ *is closed. Then step* (2) *of Algorithm 13 (Algorithm 12) is called upon to fix the problem* *of mutual independence. Choosing* $|\mathrm{cdr}(y_2)| = |x_2|$ *we have*

$$\mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \mathrm{list}) \Big[\ \mathrm{cdr}(y_2) \neq x_2 \ \wedge \ |y_2| = |\mathrm{car}(y_2)| + |x_2| + 1 \\ \wedge \ |\mathrm{cdr}(y_2)| = |x_2| \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2) + |x_2| + 1) \ \Big] \ , \quad (66)$$

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 (66) *with* (59), *we have*

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

We can now call Algorithm 6 to get the RLCC $\Phi_{\Delta}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ *for* $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, *which simplifies to*

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

By step (2) *of Algorithm 14,* (66) *is equivalent to*

$$\mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists x_2 : \mathrm{list}) \Big[\ \mathrm{cdr}(y_2) \neq x_2 \\ \wedge \ |y_2| = |\mathrm{car}(y_2)| + |x_2| + 1 \ \wedge \ |\mathrm{cdr}(y_2)| = |x_2| \\ \wedge \ |x_2| > 3 \ \wedge \ 2 \nmid |x_2| \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |x_2| + 1) \ \Big] \ , \quad (67)$$

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

$$\mathrm{cdr}(y_2) = y_1 \ \wedge \ (\exists z : \mathbb{Z}) \Big[\ |y_2| = |\mathrm{car}(y_2)| + z + 1 \ \wedge \ |\mathrm{cdr}(y_2)| = z \\ \wedge \ z > 3 \ \wedge \ 2 \nmid z \ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + z + 1) \ \Big] \ . \quad (68)$$

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*

$$\mathrm{cdr}(y_2) = y_1 \ \wedge \ |\mathrm{cdr}(y_2)| > 3 \ \wedge \ 2 \nmid |\mathrm{cdr}(y_2)| \\ \wedge \ \varphi_{\mathbb{Z}}(|\mathrm{car}(y_2)| + |\mathrm{cdr}(y_2)| + 1) \ . \quad (69)$$

It is easy to verify that (69) *implies* (62). *The reverse direction, however, does not hold because we only showed one branch of the reduction.*

Similar to Section 5.5, 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

$$(\exists \bar{\boldsymbol{x}} : \mathbb{T}) \Big[\Phi_c(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \Big] \ , \tag{70}$$

where $\Phi_c(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is a formula in $\mathscr{L}_c$. Thanks to step (2) in Algorithm 8, we can assume that none of $\bar{\boldsymbol{x}}$ appears inside selectors. Without loss of generality, we also assume $\bar{\boldsymbol{x}}$ are either all asserted to be non-nullary constructor terms or all asserted to be constants. In the former case, we can assume $\Phi_c(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is just $\Phi_c(\bar{\boldsymbol{y}})$ because $\bar{\boldsymbol{x}}$ do not properly appear inside selectors and hence they do not have any place in Φ_c. So we can simply move $\Phi_c(\bar{\boldsymbol{y}})$ out of $\exists \bar{\boldsymbol{x}}$, obtaining a quantified formula in the same form as (57). In the latter case, we can assume that $\bar{\boldsymbol{x}}$ do not properly occur inside constructor terms either, because we can decompose constructor terms properly containing $\bar{\boldsymbol{x}}$ in the same way as in Algorithms 9 and 13. So literals containing $\bar{\boldsymbol{x}}$ in $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ are equalities or disequalities on terms of constant type, and hence we can rewrite $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ as $\Phi_{\mathbb{T}}^{(a)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{T}}^{(b)}(\bar{\boldsymbol{y}})$ with $\Phi_{\mathbb{T}}^{(a)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ being a constraint in $\mathscr{L}_c$. We can then assume that $\Phi_{\mathbb{T}}^{(a)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is a part of $\Phi_c(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. We also simplify $\Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ to $\Phi'_{\mathbb{Z}}(\bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ by instantiating all $|x_i|$ to 1. Now it is clear that (70) is equivalent to

$$\Phi_{\mathbb{T}}^{(b)}(\bar{\boldsymbol{y}}) \ \wedge \ \Phi'_{\mathbb{Z}}(\bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \ \wedge \ (\exists \bar{\boldsymbol{x}} : \mathbb{T})\, \Phi_c(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ , \tag{71}$$

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

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.

7.4.1 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.

7.4.2 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.

7.4.3 Constant Instantiation

Having an infinite number of constants prohibits the application of the constant elimination (step (4) of Algorithm 8). In that step all variables $\bar{\boldsymbol{x}}$ that are asserted to be constants are nondeterministically instantiated with constants, thereby allowing the elimination of the corresponding $\exists \bar{\boldsymbol{x}}$. 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 8 all existentially quantified variables appear in disequalities only (cf. (34)). Let $\bar{\boldsymbol{x}}_c \subseteq \bar{\boldsymbol{x}}$ be the set of existentially quantified variables that are asserted to be constants. Then for any given assignment σ of the parameters $\bar{\boldsymbol{y}}$ and the remaining variables $\bar{\boldsymbol{x}} \backslash \bar{\boldsymbol{x}}_c$, all disequalities containing variables in $\bar{\boldsymbol{x}}_c$ can be simultaneously satisfied by assigning them distinct constants occurring neither in $[\![\bar{\boldsymbol{y}}]\!]\sigma$ nor in the formula. Therefore all variables in $\bar{\boldsymbol{x}}_c$ can be removed from the formula (43), as desired.

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.

We have extended our results to queues, a type of non-recursive data structure in which an object can be constructed in more than one way [36]. We 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 [37], the theory of concatenation [38] and the theory of queues with sub-sequence relations (including subqueue, prefix or suffix relation) [26].

We also plan to extend our results to allow more than one integer function on data structures. Such extensions allow us to model a wide range of augmented data structures. For example, the theory of term algebras with two length functions, which respectively give the length of the maximum path and the length of the minimum path, can express balanced tree structures, such as AVL trees and red-black trees [**?**].

9 Acknowledgments

We thank the anonymous referees for their careful reading and suggestions. The detailed comments that pointed out several inaccuracies in an earlier version were extremely valuable.

References

[1] D. C. Oppen, Reasoning about recursively defined data structures, Journal of ACM 27 (3).

[2] W. Hodges, Model Theory, Cambridge University Press, Cambridge, UK, 1993.

[3] T. Zhang, H. B. Sipma, Z. Manna, Decision procedures for recursive data structures with integer constraints, in: the 2th International Joint Conference on Automated Reasoning (IJCAR'04), Vol. 3097 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 152–167.

[4] T. Zhang, H. B. Sipma, Z. Manna, Term algebras with length function and bounded quantifier alternation, in: the 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'04), Vol. 3223 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 321–336.

[5] H. B. Enderton, A Mathematical Introduction to Logic, Academic Press, 2001.

[6] D. C. Cooper, Theorem proving in arithmetic without multiplication, in: Machine Intelligence, Vol. 7, American Elsevier, 1972, pp. 91–99.

[7] C. R. Reddy, D. W. Loveland, Presburger arithmetic with bounded quantifier alternation, in: Proceedings of the 10th Annual Symposium on Theory of Computing, ACM Press, 1978, pp. 320–325.

[8] A. I. Mal'cev, Axiomatizable classes of locally free algebras of various types, in: The Metamathematics of Algebraic Systems, Collected Papers, North Holland, 1971, Ch. 23, pp. 262–281.

[9] M. 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, IEEE Computer Society Press, 1988, pp. 348–357.

[10] H. Comon, P. Lescanne, Equational problems and disunification, Journal of Symbolic Computation 7 (1989) 371–425.

[11] H. Comon, C. Delor, Equational formulae with membership constraints, Information and Computation 112 (2) (1994) 167–216.

[12] R. Backofen, A complete axiomatization of a theory with feature and arity constraints, Journal of Logical Programming 24 (1&2) (1995) 37–71.

[13] T. Rybina, A. Voronkov, A decision procedure for term algebras with queues, ACM Transactions on Computational Logic 2 (2) (2001) 155–181.

[14] V. Kuncak, M. Rinard, The structural subtyping of non-recursive types is decidable, in: Proceedings of the 18th Annual Symposium on Logic in Computer Science, IEEE Computer Society Press, 2003, pp. 96–107.

[15] V. Kuncak, M. Rinard, On the theory of structural subtyping, Technical Report MIT-LCS-TR-879, Massachusetts Institute of Technology (Jan. 2003).

[16] G. Nelson, D. C. Oppen, Fast decision procedures based on congruence closure, Journal of ACM 27 (2) (1980) 356–364.

[17] J. Downey, R. Sethi, R. E. Tarjan, Variations of the common subexpression problem, Journal of ACM 27 (1980) 758–771.

[18] G. Nelson, D. C. Oppen, Simplification by cooperating decision procedures, ACM Transactions on Programming Languages and Systems 1 (2) (1979) 245–257.

[19] C. Tinelli, M. T. Harandi, A new correctness proof of the Nelson–Oppen combination procedure, in: F. Baader, K. U. Schulz (Eds.), Frontiers of Combining Systems: Proceedings of the 1st International Workshop (Munich, Germany), Applied Logic, Kluwer Academic Publishers, 1996, pp. 103–120.

[20] C. Tinelli, C. Ringeissen, Unions of non-disjoint theories and combinations of satisfiability procedures, Theoretical Computer Science 290 (1) (2003) 291–353.

[21] C. Tinelli, C. G. Zarba, Combining decision procedures for sorted theories, in: J. J. Alferes, J. A. Leite (Eds.), Proceedings of the 9th European Conference on Logic in Artificial Intelligence (JELIA'04), Vol. 3229 of Lecture Notes in Computer Science, Springer, 2004, pp. 641–653.

[22] A. Armando, S. Ranise, M. Rusinowitch, Uniform derivation of decision procedures by superposition, Lecture Notes in Computer Science 2142 (2001) 513–527.

[23] S. Ghilardi, Model-theoretic methods in combined constraint satisfiability, Journal of Automated Reasoning 33 (3-4) (2005) 221–249.

[24] C. G. Zarba, Combining sets with integers, in: A. Armando (Ed.), Frontiers of Combining Systems, Vol. 2309 of Lecture Notes in Artificial Intelligence, Springer, 2002, pp. 103–116.

[25] C. G. Zarba, Combining multisets with integers, in: A. Voronkov (Ed.), Proceedings of the 18th International Conference on Automated Deduction, Vol. 2392 of Lecture Notes in Artificial Intelligence, Springer, 2002, pp. 363–376.

[26] N. S. Bjørner, Integrating decision procedures for temporal verification, Ph.D. thesis, Computer Science Department, Stanford University (Nov. 1998).

[27] N. S. Bjørner, A. Browne, M. Colón, B. Finkbeiner, Z. Manna, H. B. Sipma, T. E. Uribe, Verifying temporal properties of reactive systems: A STeP tutorial, Formal Methods in System Design 16 (3) (2000) 227–270.

[28] T. Skolem, Untersuchungen über die axiome des klassenkalküls und über produktations- und summationsprobleme, welche gewisse klassen von aussagen betreffen, in: J. E. Fenstad (Ed.), Selected works in logic, Universitetsforlaget, 1970, pp. 67–101.

[29] S. Feferman, R. Vaught, The first order properties of products of algebraic systems, Fundamenta Mathematicae 47 (1959) 57–103.

[30] P. Revesz, Quantifier-elimination for the first-order theory of Boolean algebras with linear cardinality constraints, in: Proceedings of Advances in Databases and Information Systems (ADBIS'04), Vol. 3255 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 1–21.

[31] V. Kuncak, M. Rinard, An algorithm for deciding BAPA: Boolean algebra with Presburger arithmetic, in: R. Nieuwenhuis (Ed.), the 20th International Conference on Automated Deduction (CADE'05), Vol. 3632 of Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 260–277.

[32] K. Korovin, A. Voronkov, A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering, in: Proceedings of the 15th Annual Symposium on Logic in Computer Science, 2000, pp. 291 – 302.

[33] K. Korovin, A. Voronkov, Knuth-Bendix constraint solving is NP-complete, in: Proceedings of 28th International Colloquium on Automata, Languages and

Programming (ICALP'01), Vol. 2076 of Lecture Notes in Computer Science, Springer-Verlag, 2001, pp. 979–992.

[34] T. Zhang, H. B. Sipma, Z. Manna, The decidability of the first-order theory of term algebras with Knuth-Bendix order, in: R. Nieuwenhuis (Ed.), the 20th International Conference on Automated Deduction (CADE'05), Vol. 3632 of Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 131–148.

[35] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, 1979.

[36] T. Zhang, H. B. Sipma, Z. Manna, Decision procedures for queues with integer constraints, in: R. Ramanujam, S. Sen (Eds.), Proceedings of the 25th International Conference on the Foundatations of Software Technology and Theoretical Computer Science (FSTTCS'05), Vol. 3821 of Lecture Notes in Computer Science, Springer Verlag, Hyderabad, India, 2005, pp. 225–237.

[37] K. N. Venkataraman, Decidability of the purely existential fragment of the theory of term algebras, Journal of ACM 34,2 (1987) 492–510.

[38] G. S. Makanin, The problem of solvability of equations in a free semigroup, Math. Sbornik 103 (1977) 147–236.

[39] D. C. Oppen, Elementary bounds for Presburger arithmetic, in: Proceedings of 5th ACM Symposium on Theory of Computing, 1973, pp. 34–37.

[40] L. Lovász, Combinatorial Problems and Exercises, Elsevier, Horth-Holland, 1993.

[41] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, Cambridge, Massachusetts, 2001.

A Appendix

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

Lemma 54 *Let $d_1, \ldots, d_n$ be positive integers and gcd their* greatest common divisor. *Then*

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

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

$$\textit{if} \quad gcd \mid x \quad \textit{then} \quad \exists x_1, \ldots, x_n \geq 0 \left(x = \sum_{i=1}^{n} x_i d_i\right) \ . \tag{A.2}$$

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. For ease of reference we restate the propositions and theorems.

Proposition 24 Φ_Δ obtained by Algorithm 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 $\text{Tree}(|t|)$ and $\text{Tree}^\alpha(|t|)$ 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 $\text{Tree}(x)$ and $\text{Tree}^\alpha(x)$ 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 $\text{Tree}(x)$ (or $\text{Tree}^\alpha(x)$) [39]. Of course, these equivalent quantifier-free formulas are still of constant length.

In fact Tree(x) can be expressed directly by a quantifier-free formula. Indeed, Tree(x) states that $x-1$ is a non-negative linear combination of $d_1,\ldots,d_n$. By Lemma 54 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 lemma. Therefore Tree(x) is equivalent to

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

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

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

Theorem 25 Φ_Δ obtained by Algorithm 4 is an LCC for $\Phi_\mathbb{T}$.

PROOF. To show that Φ_Δ computed by Algorithm 4 is an LCC for $\Phi_\mathbb{T}$, we need to show, by Definition 17, the validity of

$$(\forall \bar{\boldsymbol{x}} : \mathbb{T})\Big[\Phi_\mathbb{T}(\bar{\boldsymbol{x}}) \;\rightarrow\; (\exists \bar{\boldsymbol{z}} : \mathbb{Z})\Big(\Phi_\Delta(\bar{\boldsymbol{z}}) \;\wedge\; |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}}\Big)\Big] \;, \tag{8}$$

$$(\forall \bar{\boldsymbol{z}} : \mathbb{Z})\Big[\Phi_\Delta(\bar{\boldsymbol{z}}) \;\rightarrow\; (\exists \bar{\boldsymbol{x}} : \mathbb{T})\Big(\Phi_\mathbb{T}(\bar{\boldsymbol{x}}) \;\wedge\; |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}}\Big)\Big] \;. \tag{9}$$

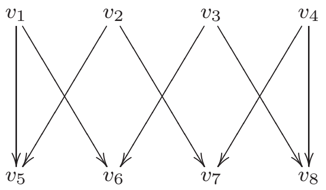
Clearly, from the description of Algorithm 4, for any variable assignment σ satisfying $\Phi_\mathbb{T}$, $|\sigma|$ satisfies $\Phi_\mathbb{T}$ and thus (8) holds.

To demonstrate the validity of (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 $R\Updownarrow$ 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 55 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 $\sigma_\mathbb{T}(v_k) = \ldots = \sigma_\mathbb{T}(v_2) = \sigma(v_1)$.

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,



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

Fig. A.1. The DAG of $\Phi_{\mathbb{T}}$ in Example 55.

$\sigma_{\mathbb{T}}$ respects $R\Downarrow$, that is, for any terms t and s, $(t, s) \in R\Downarrow$ 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 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 55 *To illustrate the variable abstraction and the construction of a satisfying variable assignment consider the following constraint*

$$\Phi_{\mathbb{T}} : \ \mathrm{Is}_{\mathrm{cons}}(x) \ \wedge \ \mathrm{cons}(y, z) = \mathrm{cons}(\mathrm{cdr}(x), z) \ \wedge \ \mathrm{car}(x) \neq z \ . \qquad \text{(A.3)}$$

The DAG of $\Phi_{\mathbb{T}}$, shown in Figure A.1, is the same as that of Example 18 and as in that example, $R\Downarrow$ is

$$\{ (v_1, v_2), \ (v_3, v_4), \ (v_6, v_7) \} \ .$$

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

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

which satisfies $\Phi_{\mathbb{T}}$.

Proposition 30 $\mathrm{CNT}^{\alpha}_{k,n}(x)$ 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 $O(n)$.

For $x > 1$, $\mathscr{N}(x)$ can be expressed as a recurrence relation:

$$\mathscr{N}(x) \;=\; \sum_{i=1}^{m} \quad \sum_{\left\{\substack{\langle x_1,\ldots,x_{d_i}\rangle \;\mid \\ x_1+\ldots+x_{d_i}=x-1\;\wedge \\ x_1,\cdots,x_{d_i}>0}\right\}} \quad \prod_{j=1}^{d_i} \mathscr{N}(x_j)\ . \tag{A.4}$$

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 $\mathscr{N}(1), \mathscr{N}(2), \ldots$, with $\mathscr{N}(1) = k$, until we reach the first x_{min} such that $\mathscr{N}(x_{min}) > n$.

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

$$\mathrm{Tree}(x_1)\ \wedge\ \mathrm{Tree}(x_2)\ \wedge\ x_1 > x_2 \quad\rightarrow\quad \mathscr{N}(x_1) > \mathscr{N}(x_2)\ . \tag{A.5}$$

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, (A.5) 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 Lemma 54 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 $\text{Tree}(x)$ holds, that is, $x - 1$ can be expressed as a non-negative linear combination of $d_1, \ldots, d_m$. By Lemma 54, (A.1), $gcd \ \mid \ x - 1$. 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 Lemma 54, (A.2), $x - x_{min}$ 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 33 Φ_Δ obtained by Algorithm 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 24, the call to Algorithm 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 6 adds $\text{CNT}^{\alpha}_{k,m}(x)$ to Φ_Δ. By Proposition 30, $\text{CNT}^{\alpha}_{k,m}(x)$ 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 34 Φ_Δ obtained by Algorithm 6 is an RLCC for $\Phi_\mathbb{T}/\theta_\mathbb{Z}$.

PROOF. Let $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}})$ be a type- and equality-complete term constraint and $\theta_{\mathbb{Z}}(|\bar{\boldsymbol{x}}|)$ be a Presburger formula. To show that Φ_{Δ} computed by Algorithm 6 is an RLCC for $\Phi_{\mathbb{T}}/\theta_{\mathbb{Z}}$, we need to show, by Definition 20, the validity of

$$(\forall \bar{\boldsymbol{x}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \ \wedge\ \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}) \ \rightarrow\ (\exists \bar{\boldsymbol{z}} : \mathbb{Z}) \Big(\Phi_{\Delta}(\bar{\boldsymbol{z}}) \ \wedge\ |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \Big) \Big] \ , \tag{11}$$

$$(\forall \bar{\boldsymbol{z}} : \mathbb{Z}) \Big[\Phi_{\Delta}(\bar{\boldsymbol{z}}) \ \rightarrow\ (\exists \bar{\boldsymbol{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}) \ \wedge\ \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}) \ \wedge\ |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \Big) \Big] \ . \tag{12}$$

For (11) consider an arbitrary variable assignment σ such that $\Phi_{\mathbb{T}} \wedge \theta_{\mathbb{Z}}$ is true. By Algorithm 6, Φ_{Δ} consists of $\Phi_{\Delta,4} \wedge \theta_{\mathbb{Z}} \wedge (\mathrm{CNT}^{\alpha}_{k,n})_i$, where $\Phi_{\Delta,4}$ is the constraint computed by Algorithm 4 and $(\mathrm{CNT}^{\alpha}_{k,n})_i$ are the counting constraints added in step (3). By Theorem 25, 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 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 (12) consider an arbitrary integer assignment σ_{Δ}, assigning $\bar{\boldsymbol{z}} := \bar{\boldsymbol{d}}$, such that $\Phi_{\Delta}(\bar{\boldsymbol{d}})$ holds. We have to show that there exists a term assignment $\sigma_{\mathbb{T}}$, assigning $\bar{\boldsymbol{x}} := \bar{\boldsymbol{t}}$ such that $|\bar{\boldsymbol{t}}| = \bar{\boldsymbol{d}}$ and $\Phi_{\mathbb{T}}(\bar{\boldsymbol{t}})$ and $\theta_{\mathbb{Z}}(\bar{\boldsymbol{t}})$ hold. In contrast with the proof of Theorem 25, 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 $R\rrbracket$ the bidirectional closure for $\Phi_{\mathbb{T}}$ as constructed in Algorithm 1. We assume, as in the proof of Theorem 25, 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 $R\rrbracket$ 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}:

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

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 $t^{(1)}_i$ $(0 \leq i \leq n_1)$ is either a constant or a term variable as its length is the smallest, and we only need to consider $t_i^{(1)}$ which are term variables. By Algorithm 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 $t_i^{(1)}$ (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 $\theta_{\mathbb{Z}}$, $\theta_{\mathbb{Z}}(\bar{\boldsymbol{d}})$ holds, and therefore $\sigma_{\mathbb{T}}$ also satisfies $\theta_{\mathbb{Z}}$, as required. $\square$

Theorem 39 All transformations in Algorithm 10 preserve equivalence.

PROOF. Recall that Algorithm 10 first transforms a formula into one in solved form, that is in the form

$$(\exists \bar{\boldsymbol{x}}) \left[\bigwedge_i x_{p(i)} \neq t_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \right] \\ \wedge \ \bigwedge_k F_k y_{f(k)} \neq u_k(\bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_l G_l y_{g(l)} = H_l y_{h(l)} \ . \quad (42)$$

using Algorithm 8. It then eliminates the quantifiers by removing all literals containing $\bar{\boldsymbol{x}}$. It is straightforward to show that all transformations in Algorithm 8 preserve equivalence, and we omit the proof. It remains to show that (42) is equivalent to

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

that is, that

$$(\forall \bar{\boldsymbol{y}} : \mathbb{T})(\exists \bar{\boldsymbol{x}} : \mathbb{T}) \left[\bigwedge_i x_{p(i)} \neq t_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \bigwedge_j L_j y_{q(j)} \neq s_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \right] . \quad (43)$$

is valid, or that

$$\bigwedge_i x_{p(i)} \neq t_i(\bar{\boldsymbol{x}}, \bar{\boldsymbol{b}}) \ \wedge \ \bigwedge_j L_j b_{q(j)} \neq s_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{b}}) \ , \quad \text{(A.6)}$$

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

Let $\bar{\boldsymbol{x}}$ be $x_1, \ldots, x_n$. Let ξ be the maximal length of terms not containing $\bar{\boldsymbol{x}}$ in (A.6) and let $\rho > \xi$ be such that there exist at least n distinct nonconstant terms $\bar{\boldsymbol{d}} = d_1, \ldots, d_n$ of length ρ. We claim that $\bar{\boldsymbol{d}}$ satisfies (A.6). Observe that, due to step (4) of Algorithm 8, none of the x_i are constrained to be constants, and thus all variables can be assigned nonconstant terms.

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

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

(1) t_i does not contain any variable in $\bar{\boldsymbol{x}}$. Then $|t_i(\bar{\boldsymbol{b}})| \leq \xi$ while $|d_{p(i)}| = \rho > \xi$, and hence $d_{p(i)} \neq t_i(\bar{\boldsymbol{b}})$.
(2) t_i properly contains a variable in $\bar{\boldsymbol{x}}$. Then $|d_{p(i)}| = \rho$ but $|t_i(\bar{\boldsymbol{d}}, \bar{\boldsymbol{b}})| > \rho$, and hence $d_{p(i)} \neq t_i(\bar{\boldsymbol{b}})$.
(3) t_i is a constant c_i. Then $d_{p(i)} \neq c_i$ holds since none of terms in $\bar{\boldsymbol{d}}$ is a constant.
(4) t_i is a variable in $\bar{\boldsymbol{x}}$. 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 $\bar{\boldsymbol{d}}$. □

Theorem 40 Algorithm 10 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 8), 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 [1].

It suffices to analyze each step of Algorithm 8. 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 9) generates at most $2^{O(n)}$ disjuncts. In addition, it can increase the matrix size to $O(n^2)$, due to solving $\bar{\boldsymbol{x}}$ in terms of $\bar{\boldsymbol{y}}$ (step (3) of Algorithm 9); 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 48 If $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is strongly solved in $\bar{\boldsymbol{x}}$ and cluster complete, then $\Phi_{\Delta}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ computed by Algorithm 6 is an RLCC for $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$.

PROOF. To show that Φ_{Δ} computed by Algorithm 6 is an RLCC for $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, we need to show, by Definition 45, the validity of

$$(\forall \bar{\boldsymbol{x}}, \bar{\boldsymbol{y}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \rightarrow (\exists \bar{\boldsymbol{z}} : \mathbb{Z}) \Big(\Phi_{\Delta}(\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}}) \wedge |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \Big) \Big] , \quad (52)$$

$$\begin{aligned} (\forall \bar{\boldsymbol{y}} : \mathbb{T})(\forall \bar{\boldsymbol{z}} : \mathbb{Z}) \Big[& \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \ \wedge \ \Phi_{\Delta}(\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}}) \\ & \rightarrow (\exists \bar{\boldsymbol{x}} : \mathbb{T}) \Big(\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ |\bar{\boldsymbol{x}}| = \bar{\boldsymbol{z}} \Big) \Big] . \end{aligned} \quad (53)$$

The validity of (52) can be shown by a similar argument as was given for (11) in the proof of Theorem 34.

To prove (53) consider arbitrary assignments σ_{Δ} and $\sigma_{\mathbb{T}}^{(y)}$, assigning $\bar{\boldsymbol{z}} := \bar{\boldsymbol{p}}$ and $\bar{\boldsymbol{y}} := \bar{\boldsymbol{s}}$ respectively, such that $\Phi_{\Delta}(\bar{\boldsymbol{p}}, \bar{\boldsymbol{s}})$ and $\Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{s}})$ hold. We have to show that there exists a term assignment $\sigma_{\mathbb{T}}^{(x)}$, assigning $\bar{\boldsymbol{x}} := \bar{\boldsymbol{t}}$ such that $|\bar{\boldsymbol{t}}| = |\bar{\boldsymbol{p}}|$ and $\Phi_{\mathbb{T}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ and $\theta_{\mathbb{Z}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ hold. As in the proof of Theorem 34, we incrementally construct $\sigma_{\mathbb{T}}^{(x)}$, starting with terms of smallest length.

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

$$\underbrace{|u_0^{(1)}| = \cdots = |u_{n_1}^{(1)}|}_{block\ 1} \ < \ \underbrace{|u_0^{(2)}| = \cdots = |u_{n_2}^{(2)}|}_{block\ 2} \ < \ \cdots\cdots < \underbrace{|u_0^{(j)}| = \cdots = |u_{n_j}^{(j)}|}_{block\ j}$$

Let l_i be the length of terms in the i^{th} block. For any TA-term t occurring in $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, $|\bar{\boldsymbol{t}}|$ appears in $\Phi_{\Delta}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ and hence in the above sequence. In general a block can contain more than one cluster for each type. However, by Proposition 50, maximal clusters are mutually independent, 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 $u_1^{(1)}, \ldots, u_{n_1}^{(1)}$, we incrementally construct a satisfying assignment for $\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{s}})$. We only need to consider $u_{n_j}^{(j)}$ which contains $\bar{\boldsymbol{x}}$. Obviously $u_i^{(1)}$ $(0 \leq i \leq n_1)$ is either a constant or a variable in $\bar{\boldsymbol{x}}$ as its length is the smallest. By Algorithm 6 we know that $\mathrm{CNT}_{k,n_1}^{\alpha_1}(l_1)$ is in $\Phi_\Delta(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. As $\bar{\boldsymbol{p}}$ satisfies $\Phi_\Delta(\bar{\boldsymbol{x}}, \bar{\boldsymbol{s}})$, there are at least $n_1 + 1$ different terms of length l_1. Therefore we can simply assign each $u_i^{(1)}$ (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 $\bar{\boldsymbol{x}}$ 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 $\mathrm{CNT}_{k,n_{i+1}}^{\alpha_{i+1}}(l_{i+1})$ in $\Phi_\Delta(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, 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 34. The only problematic case is that a cluster may have selector terms containing $\bar{\boldsymbol{y}}$ as well as constructor terms containing $\bar{\boldsymbol{x}}$. For example, suppose that a cluster C in the i^{th} block contains both Ly and $\alpha_i(t_1(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}), \ldots, t_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}))$ for $\mathrm{ar}(\alpha_i) = k$. $[\![Ly]\!]$ is fixed by $\sigma_{\mathbb{T}}^{(y)}$ and $[\![t_1(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})]\!], \ldots, [\![t_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})]\!]$ have been determined before the i^{th} round and hence $[\![\alpha_i(t_1(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}), \ldots, t_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}))]\!]$ is determined too. By Definition 46, however, the disequality $Ly \neq t(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is redundant, which can only be the case if for some j $(1 \leq j \leq k)$, $\mathrm{s}_j^{\alpha_i} Ly \neq t_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is implied by $\Phi_{\mathbb{T}}$. Since $|t_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})| < |\alpha_i(t_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}), \ldots, t_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}))|$, $t_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ has been assigned before the i^{th} round such that $[\![\mathrm{s}_j^{\alpha_i} Ly]\!] \neq [\![t_j(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})]\!]$. As a consequence we have $[\![Ly]\!] \neq [\![\alpha_i(t_1(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}), \ldots, t_k(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}))]\!]$.

Since at each step we can build a satisfying partial assignment for $\bar{\boldsymbol{x}}$, by induction, we can eventually construct a satisfying assignment $\bar{\boldsymbol{t}}$ such that $\Phi_{\mathbb{T}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ and $|\bar{\boldsymbol{t}}| = \bar{\boldsymbol{p}}$. It is clear that $\theta_{\mathbb{Z}}(\bar{\boldsymbol{t}}, \bar{\boldsymbol{s}})$ also holds as Φ_Δ implies $\theta_{\mathbb{Z}}$ (thanks to step (2) of Algorithm 6). □

Proposition 50 Algorithm 11 produces a formula which is in strongly solved form and cluster complete.

PROOF. The production of a strongly solved form is guaranteed by Algorithm 13. It suffices to show that Algorithm 13 preserves cluster completeness. Step (1a) of Algorithm 13 may generate new equalities as well as new disequality literals of the form $Ly \neq t'(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$, which in general destroys the mutual independence of clusters. Step (2) (Algorithm 12), however, is called after each run of step (1a) to restore cluster completeness. The termination argument is given in the description of the algorithm. □

Theorem 51 All transformations in Algorithm 14 preserve equivalence.

PROOF. Clearly, Algorithm 11 preserves equivalence. So, it suffices to show the equivalence between

$$(\exists \bar{\boldsymbol{x}} : \mathbb{T}) \Big[\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \ \wedge \ \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \Big] \tag{59}$$

and

$$\Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \ \wedge \ (\exists \bar{\boldsymbol{u}} : \mathbb{Z}) \Big[\Phi_{\Delta}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}) \ \wedge \ \Phi_{\mathbb{Z}}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}) \Big] \ . \tag{61}$$

By Proposition 50 $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is in strongly solved form and cluster complete. It then follows from Theorem 48 that $\Phi_{\Delta}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is an RLCC for $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. By Proposition 22, $\Phi_{\Delta}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ is an RLCC for $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})/\bar{\boldsymbol{x}}/(\theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}}))$.

To show that (59) implies (61), assume that for all $\bar{\boldsymbol{y}}$ and for all $\bar{\boldsymbol{z}}$ there exists $\bar{\boldsymbol{x}}$ such that $\Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ holds. Then by Definition 45, (52), there exists $\bar{\boldsymbol{u}} = |\bar{\boldsymbol{x}}|$ such that $\Phi_{\Delta}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ holds, and hence (61) holds.

To show that (61) implies (59), assume that for all $\bar{\boldsymbol{y}}$ and for all $\bar{\boldsymbol{z}}$ there exists $\bar{\boldsymbol{u}}$ such that $\Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}}) \wedge \Phi_{\Delta}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ holds. Then by Definition 45, (53), there exists $\bar{\boldsymbol{x}}$ such that $|\bar{\boldsymbol{x}}| = \bar{\boldsymbol{u}}$ and $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \theta_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{Z}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{z}})$ holds, and hence, noting that $\Phi_{\mathbb{T}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \equiv \Phi_{\mathbb{T}}^{(1)}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) \wedge \Phi_{\mathbb{T}}^{(2)}(\bar{\boldsymbol{y}})$, (59) holds, as required. □

Theorem 52 Algorithm 14 eliminates a block of quantifiers in time $2^{O(n^2 \lg n)}$.

PROOF. We only need to analyze each step of Algorithm 11. As shown in the proof of Theorem 40, step (1) (basic normalization, Algorithm 8) generates $2^{O(n)}$ disjuncts each of which contains $O(n)$ distinct terms.

Step (2) (cluster completion, Algorithm 12) produces $2^{O(n \lg n)}$ cluster completions. An equality completion is a valid product of a partition of terms on syntactic equality and a partition on terms on length equality. For a set of size n the number of distinct partitions is equal to the Bell number, $B(n)$. 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 $\rho(n) \lg \rho(n) = n$ [40]. Obviously $B(n)$ is bounded by $2^{O(n \lg n)}$, and so is the number of cluster completions.

Step (3) (Algorithm 13) is a bit more costly. Let the input to Algorithm 13 be a formula of the form (43) that induces n clusters containing variables in $\bar{\boldsymbol{x}}$. Assign to the rank r of each of these clusters a number $p(r)$ in $[1..n]$ such that if $r_1 < r_2$, then $p(r_1) < p(r_2)$. The sum of all rank numbers is bounded by n^2. Each run of step (1-2) reduces the sum by at least 1, and generates at most one new term appearing in disequalities. The total number of distinct terms after Algorithm 13 is bounded by $O(n^2)$. The number of cluster completions is therefore bounded by $2^{O(n^2 \lg n^2)} = 2^{O(n^2 \lg n)}$, including the cost of Algorithm 12 and Algorithm 9, both of which may be called $O(n^2)$ times in the run of Algorithm 13. $\square$