Linear Constraints over Infinite Trees

Martin Hofmann and Dulma Rodriguez

Department of Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
{martin.hofmann,dulma.rodriguez}@ifi.lmu.de

Abstract. In this paper we consider linear arithmetic constraints over infinite trees whose nodes are labelled with nonnegative real numbers. These constraints arose in the context of resource inference for object-oriented programs but should be of independent interest. It is as yet open whether satisfiability of these constraint systems is at all decidable. For a restricted fragment motivated from the application to resource inference we are however able to provide a heuristic decision procedure based on regular trees. We also observe that the related problem of optimising linear objectives over these infinite trees falls into the area of convex optimisation.

Keywords: Constraints, Infinite trees, Resource analysis.

1 Introduction

In this paper we present a new algorithmic problem related to linear arithmetic over $\mathbb{D} = \mathbb{R}^+ \cup \{\infty\}$. Indeed, it can be seen as a special case of linear arithmetic with infinitely many variables (with some schematic notation so as to make instances of the problem finite objects).

While in general linear arithmetic with infinitely many variables is easily seen to be undecidable (introduce a variable x_{it} for every position i and time t of a computation on a Turing machine) the question of decidability for our special case remains open. We do, however, provide a heuristic solution for an important subcase motivated by practical considerations.

We begin with an informal description of our constraint systems. We have arithmetic variables that take on values in $\mathbb{D} = \mathbb{R}^+ \cup \{\infty\}$ and *tree variables* whose values are infinite trees whose nodes are labelled with elements of $\mathbb{D}$. We fix a finite set $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ of labels to address the children of a node, e.g. $\mathcal{L} = \{L, R\}$ for infinite binary trees and $\mathcal{L} = \{tl\}$ for infinite lists.

Such trees can be added, scaled, and compared componentwise; furthermore, we have an operation $\Diamond(.)$ that extracts the root label of a tree, thus if t is a tree expression then $\Diamond(t)$ is an arithmetic expression. Finally, if t is a tree expression and $l \in \mathcal{L}$ then $l(t)$ is a tree expression denoting the l-labelled immediate subtree of t.

Given a system of constraints built from these constructions we can ask for satisfiability and for values of selected arithmetic variables. Asking for values of tree

N. Bjørner and A. Voronkov (Eds.): LPAR-18, LNCS 7180, pp. 343–358, 2012.

variables makes no sense in general as these are infinite objects. We can also ask for the optimum value of some linear combination of the arithmetic variables.

In Figure 1 two infinite trees t_1, t_2 over label set $\mathcal{L} = \{L, R\}$ are defined. It also contains two infinite trees over label set $\mathcal{L} = \{tl\}$ which are effectively infinite lists. Within one and the same constraint system we can only use trees over one and the same label set. These trees satisfy for example: $L(t_1) = R(t_1) = t_1$, $t_1 \sqsubseteq t_2$, $l_2 \sqsubseteq l_1$, $\Diamond(t_1) = 1$, $\Diamond(t_2) = 2$. We also have $t_1 + t_1 = t_2$ and $2t_1 = t_2$ and $tl(l_1) = l_1 + l_2$.

Now, the constraint system $tl(x) \sqsupseteq x \wedge \Diamond(x) \geq 1$ is satisfiable, for example with $x = l_1$ and its optimum value with respect to the objective $c = \Diamond(x)$ to be minimised equals 1. The constraint system $L(x) \sqsupseteq x \wedge R(x) \sqsupseteq x \wedge \Diamond(x) \geq 6$ is satisfiable, for example with $x = 6t_1$.

The constraint system $\Diamond(x) \geq 1 \wedge 2tl(x) = tl(x)$ is also satisfiable, namely by $x = 10^\omega$, but $\Diamond(x) \geq 1 \wedge 2tl(x) = tl(x) \wedge x = tl(x)$, however, is unsatisfiable.

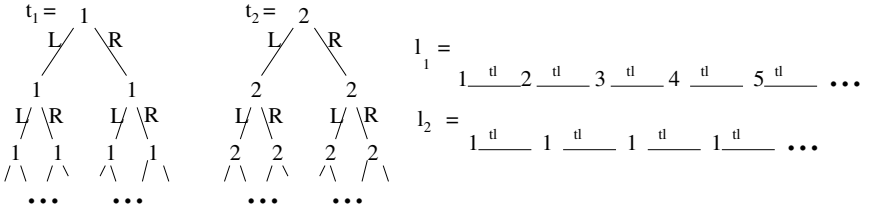

Fig. 1. Some infinite trees

As already mentioned, we currently do not know whether satisfiability of such constraint systems is in general decidable, but the heuristic method we shall present covers all the constraint systems given so far. This is because, the trees witnessing satisfiability were *regular* in the sense that their set of subtrees is finite. So, t_1, t_2, l_2 are regular, but l_1 is not. Accordingly, a constraint system like $\Diamond(x) \geq 1 \wedge tl(x) \sqsupseteq x \wedge tl(y) \sqsupseteq x + y$ is not amenable to our heuristic as it does not admit a regular solution.

In order to decide satisfiability of constraints in general it is tempting to use Büchi tree automata; however, in order to represent our "arithmetic" trees as a tree whose nodes are labelled with letters from a finite alphabet, we would have to represent the numerical annotations using extra branches and then primitive predicates such as equality cannot be recognised by a Büchi tree automaton. Indeed, we conjecture that the algebraic structure of arithmetic trees is not "automatic" in the sense of [BG00].

Nevertheless, we believe that satisfiability of our constraint systems is decidable; in support of this conjecture, we can enlist the fact that the set of solutions to a constraint system is *convex* in the sense that if $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are both solutions then so is $(1 - \lambda)\boldsymbol{t}_1 + \lambda\boldsymbol{t}_2$ for $\lambda \in [0, 1]$. Furthermore, constraint systems can be reduced by algebraic manipulations and elimination steps to canonical forms from which solutions can be read off.

We encountered these constraint systems as part of our endeavour of developing an automatic type inference for the object-oriented resource type system presented in [HJ06,HR09]. Indeed, we were able to reduce the type inference problem for that system to satisfiability of arithmetic tree constraint systems. While we do not describe this rather intricate reduction in this paper we try to give a rough indication of it so as to provide further motivation for the potential usefulness of arithmetic tree constraint systems.

The type system presented in *loc.cit.* ascribes refined types to objects in such a way that a concrete object together with its type defines a nonnegative number—the *potential* of that object. The typing rules are formulated in such a way that the potential of all reachable objects under their current typing furnishes an upper bound on the resource usage of any subsequent statement plus the potential of all reachable objects after its execution. In this way, by telescoping, the potential of the initial heap configuration furnishes an upper bound on the total resource usage of a program and this then can be used to read off an input dependent resource bound, e.g. in the form of a linear function of the resource consumption of a program. Through a very coarse lens we can represent the refined type of an object of some class C with fields L and R also of class C as an arithmetic tree over label set $\{L, R\}$. The potential of such an object is then given as the sum of all non-null access paths. If, e.g., the "type" of some object o is t_2 from Fig. 1 and its L, R fields are both *null* then this object carries a potential of 2. If, on the other hand, object o' satisfies $o'.L = o'.R = o$ then the potential of o will equal 6 (and not 4 because ascription of potential is oblivious to aliasing).

In order to infer types one can then introduce an appropriate tree variable wherever a type is required and generated constraints from side conditions of typing rules. Constraints of the form $t \sqsupseteq t'$ arise from subtyping, whereas constraints of the form $t \sqsupseteq t' + t''$ arise from *sharing*, i.e. multiple use of a variable.

We hope, though, that due to their compact and general formulation our arithmetic tree constraint systems will find other applications beyond type inference as well.

We were surprised to find practically no directly related work. One notable exception is [DV07] where constraint satisfaction problems with infinitely many variables are introduced and studied. The difference to our work is twofold: first, the range of individual variables in *loc.cit.* is finite, e.g. Boolean in contrast to $\mathbb{D}$ in our case; secondly, the access policy is much more general and leads to undecidability in general. Interestingly, the near absence of related work has also been noted in *loc.cit.*

2 Infinite Trees

In this section we present infinite trees labelled with nonnegative real numbers. Fix a finite set of labels $\mathcal{L} = \{l_1, \ldots, l_n\}$. The set $\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ of infinite trees is given by $\mathsf{T}^{\mathcal{L}}_{\mathbb{D}} = \{t \mid t : \mathcal{L}^* \to \mathbb{D}\}$ where $\mathbb{D} = \mathbb{R}^+ \cup \{\infty\}$ with $0 \in \mathbb{R}^+$. We will refer to elements $w \in \mathcal{L}^*$ as *paths*. We write $|w|$ for the length of w, where $|\epsilon| = 0$ and $|lw| = |w| + 1$. A tree t' is a *sub-tree* of a tree t if there exists $w \in L^*$ so that $t'(p) = t(w\,p)$ for all $p \in L^*$. Further, we say that an infinite tree is *regular*

if it contains a finite number of different sub-trees. The set $\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ carries a final coalgebra structure consisting of the function

$$\begin{array}{ll} \langle \Diamond, \mathsf{step} \rangle : & \mathsf{T}^{\mathcal{L}}_{\mathbb{D}} \to \mathbb{D} \times (\mathcal{L} \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}) \\ t & \mapsto \langle t(\epsilon), \lambda l\, w\, .\, t(l\, w) \rangle \end{array}$$

where $\mathsf{step}\, l_i$ returns the ith subtree, and $\Diamond$ gives the label of the root node tree [SR10]. We write l_i as a short notation for $\mathsf{step}\, l_i$. For any domain U, every family of functions $lt_i : U \to U$ and $o : U \to \mathbb{D}$ defines a unique function $h : U \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$, such that $\Diamond(h(x)) = o(x)$ and $l_i(h(x)) = h(lt_i(x))$. We define a preorder $\sqsubseteq$ between trees as follows:

Definition 1. *Let $t, t' \in \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$. We define $t \sqsubseteq t'$ coinductively by $t \sqsubseteq t' \iff \Diamond(t) \leq \Diamond(t')$ and $l_i(t) \sqsubseteq l_i(t')$ for all $l_i \in \mathcal{L}$.*

Alternatively, we can define the same preorder pointwise by:

Definition 2. *Let $t, t' \in \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$. Then $t \sqsubseteq_{\mathsf{ind}} t' \iff$ for all $w \in L^*\ .\ t(w) \leq t'(w)$.*

Lemma 1. *$t \sqsubseteq_{\mathsf{ind}} t' \iff t \sqsubseteq t'$.*

We define addition of trees ($+ : \mathsf{T}^{\mathcal{L}}_{\mathbb{D}} \times \mathsf{T}^{\mathcal{L}}_{\mathbb{D}} \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$) by: $\Diamond(t + t') = \Diamond(t) + \Diamond(t')$ and $l_i(t + t') = l_i(t) + l_i(t')$ and multiplication of trees with a nonnegative scalar ($\cdot : \mathbb{R}^+ \times \mathsf{T}^{\mathcal{L}}_{\mathbb{D}} \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$) by: $\Diamond(c \cdot t') = c \cdot \Diamond(t')$ and $l_i(c \cdot t') = c \cdot l_i(t')$ for each $l_i \in L$.

Defining a Complete Lattice For the following we recall that the domain $\mathbb{D} = \mathbb{R}^+ \cup \{\infty\}$ is a complete lattice under its usual order by the completeness axiom for $\mathbb{R}$ and because it has top and bottom elements: ∞ and 0. For each $d \in \mathbb{D}$ we define $\widehat{d} \in \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ by $\Diamond\left(\widehat{d}\right) = d$ and $l_i(\widehat{d}) = \widehat{d}$ for each $l_i \in \mathcal{L}$. Then, $\widehat{\infty}$ is the top element in $\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ and $\widehat{0}$ the bottom. We will show that $(\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}, \sqsubseteq)$ is a complete lattice. For each subset of $\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$, we define its least upper bound and its greatest lower bound as follows.

- $\bigwedge : \mathcal{P}(\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}) \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ is totally determined by $\Diamond(\bigwedge T) = \min_{t \in T}(\Diamond(t))$ and $l_i(\bigwedge T) = \bigwedge l_i(T)$.
- $\bigvee : \mathcal{P}(\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}) \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ is totally determined by: $\Diamond(\bigvee T) = \max_{t \in T}(\Diamond(t))$ and $l_i(\bigvee T) = \bigvee l_i(T)$.

Lemma 2 (Complete Lattice). *Let $t \in \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ and $T \subseteq \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$. Then:*

1. *$\widehat{\infty} \sqsubseteq t$ and $t \sqsubseteq \widehat{0}$.*
2. *$\bigvee T$ is the least upper bound of T and $\bigwedge T$ is the greatest lower bound of T.*
3. *$(\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}, \sqsubseteq)$ is a complete lattice.*

3 Constraints

Next, we consider a system of inequalities among tree expressions and a system of linear arithmetic constraints. Let X be a fixed, countably infinite set of variables

and Λ be a fixed countably infinite set of arithmetic variables where $X \cap \Lambda = \emptyset$. We write TAExp to denote the set of tree expressions that represent a path. We call these expressions *atomic*. The set TExp denotes expressions that represent either a path or a sum of paths. We call expressions in TExp, that are not atomic, *compound*. Moreover, we write AExp to denote linear arithmetic expressions. An arithmetic expression is either a number n, an arithmetic variable λ, an expression representing a potential found at some path $\Diamond(\mathsf{tae})$ or a sum of two expressions $\mathsf{ae}_1 + \mathsf{ae}_2$. We build the sets of *valid* expressions TExp and AExp by the following grammar, where $x \in X$, $n \in \mathbb{D}$, $\lambda \in \Lambda$ and $l \in \mathcal{L}$.

$$\begin{array}{lll} \mathsf{tae} ::= x \mid l(\mathsf{tae}) & & \in \mathsf{TAExp} \\ \mathsf{te} \;\; ::= \mathsf{tae} \mid \mathsf{te} + \mathsf{te} & & \in \mathsf{TExp} \\ \mathsf{ae} \;\; ::= n \mid \lambda \mid \Diamond(\mathsf{tae}) \mid \mathsf{ae} + \mathsf{ae} & & \in \mathsf{AExp} \\ \mathsf{tc} \;\; ::= \mathsf{te} \sqsubseteq \mathsf{te} & & \in \mathsf{TConstr} \\ \mathsf{ac} \;\; ::= \mathsf{ae} \leq \mathsf{ae} & & \in \mathsf{AConstr} \end{array}$$

A system of constraints is a set of valid tree constraints and arithmetic constraints, i.e. a pair $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$ where $\mathcal{TC}$ and $\mathcal{AC}$ are finite subsets of $\mathsf{TConstr}$ and $\mathsf{AConstr}$ respectively. We write $\mathsf{Vars}(\mathsf{te}) \subseteq X$ for the set of tree variables that occur in the tree expression te and $\mathsf{Vars}(\mathsf{ae}) \subseteq X \cup \Lambda$ for the set of tree and arithmetic variables that appear in the arithmetic expression ae. Moreover, we write $\mathsf{Vars}(\mathcal{C})$ for the set of tree and arithmetic variables that appear in $\mathcal{C}$. Sometimes we write $\mathcal{C}(\boldsymbol{x}, \boldsymbol{\lambda})$ as a short notation for $\mathsf{Vars}(\mathcal{C}) = \boldsymbol{x}, \boldsymbol{\lambda}$.

Meaning of Constraints. Let $\pi = (\pi_t, \pi_a)$ where $\pi_t : X \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ and $\pi_a : \Lambda \to \mathbb{D}$. The meaning of arithmetic expressions $\pi(\mathsf{ae}) : \mathbb{D}$ is defined in the obvious way, e.g. $\pi(\lambda) = \pi_a(\lambda)$ and $\pi(\Diamond(\mathsf{tae})) = \Diamond(\pi(\mathsf{tae}))$. The meaning of tree expressions $\pi(\mathsf{te}) : \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ is defined as one might expect, e.g. $\pi(x) = \pi_t(x)$ and $\pi(l(\mathsf{tae})) = l(\pi(\mathsf{tae}))$. Then, π satisfies a tree constraint $\mathsf{te} \sqsubseteq \mathsf{te}'$ (written $\pi \models \mathsf{te} \sqsubseteq \mathsf{te}'$) if $\pi(\mathsf{te}) \sqsubseteq \pi(\mathsf{te}')$. Similarly, π satisfies an arithmetic expression $\mathsf{ae}_1 \leq \mathsf{ae}_2$ ($\pi \models \mathsf{ae}_1 \leq \mathsf{ae}_2$) if $\pi(\mathsf{ae}_1) \leq \pi(\mathsf{ae}_2)$. Finally, we say π satisfies a system of constraints $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$ if $\pi \models \mathsf{tc}$ for each $\mathsf{tc} \in \mathcal{TC}$ and $\pi \models \mathsf{ac}$ for each $\mathsf{ac} \in \mathcal{AC}$.

We say that the variable x occurs *only positively* in the system of constraints $\mathcal{C}$ (and write $\mathcal{C}(x^+)$) when it appears only on the right hand side of constraints. Conversely, we say that it appears *only negatively* (and write $\mathcal{C}(x^-)$) when it appears only on the left hand side. Finally, if the variable appears sometimes on the left, sometimes on the right, we write $\mathcal{C}(x^+, x^-)$.

Lemma 3. *Let $\mathcal{C}(x^+)$ and $\mathcal{D}(x^-)$ be systems of constraints and $t, \hat{t} \in \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ with $t \sqsubseteq \hat{t}$.*

1. *If $\pi[x \mapsto t] \models \mathcal{C}(x^+)$ then $\pi[x \mapsto \hat{t}] \models \mathcal{C}$.*
2. *If $\pi[x \mapsto \hat{t}] \models \mathcal{D}(x^-)$ then $\pi[x \mapsto t] \models \mathcal{D}$.*

Given a tree expression te and a path w we define $\mathsf{te}_w : \mathsf{AExp}$ inductively by $\mathsf{te}_\epsilon = \Diamond(\mathsf{te})$ and $\mathsf{te}_{lw} = l(\mathsf{te})_w$. The resulting expression may not be valid, but it can easily be transformed into an equivalent valid one with the following transformations

$$l(\mathsf{tae}_1 + \mathsf{tae}_2) = l(\mathsf{tae}_1) + l(\mathsf{tae}_2) \qquad \Diamond(\mathsf{tae}_1 + \mathsf{tae}_2) = \Diamond(\mathsf{tae}_1) + \Diamond(\mathsf{tae}_2) \quad (3.1)$$

For example, $(x + y)_l = \Diamond(l(x + y))$ is not valid but it is equivalent to $\Diamond(l(x)) + \Diamond(l(y))$. Moreover, we define substitution of tree variables with tree expressions in constraints $\mathcal{C}\,[\mathsf{te}/x]$ as usual and ensure that the resulting constraints are valid, again by the transformations (3.1).

3.1 Algorithmic Problems

In this section we discuss algorithmic problems regarding a system of constraints $\mathcal{C}$ whose study would be of interest.

Satisfiability. One important problem, with a direct application to type inference for the RAJA typing system[HJ06], is satisfiability. That is, if we have given a system of constraints, we would like to know whether it is satisfiable. Moreover, we would like to obtain a valuation π that satisfies the constraints. Here we give a slightly weaker definition of the satisfiability problem. We are interested in a *finite* set of arithmetic constraints that is satisfiable iff the system of constraints $\mathcal{C}$ is satisfiable. Since the trees we are studying are infinite, it is not possible to obtain a valuation $\pi_t : X \to \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ in general. However, we will see in Section 4 that we can effectively deliver a valuation π_t when all the values in $\mathrm{ran}(\pi_t)$ are regular trees.

Reducing the satisfiability problem to the problem of satisfying a finite set of arithmetic constraints is advantageous because there are effective ways of solving linear arithmetic constraints. Moreover, we remark that the problem of obtaining an *infinite* set of arithmetic constraints equivalent to $\mathcal{C}$ is trivial. If we follow the definition of inequality ($\sqsubseteq_{\mathsf{ind}}$) we notice that a set of inequalities over trees $\mathcal{TC} = \bigcup_i \mathsf{te}_i \sqsubseteq \mathsf{te}'_i$ is satisfiable iff the following set of arithmetic constraints is satisfiable: $\Gamma(\mathcal{TC}) = \{\mathsf{te}_{iw} \leq \mathsf{te}'_{iw} \mid w \in \mathcal{L}^*\}$. In Section 4 we provide an algorithm for solving satisfiability that is sound in all cases and complete for constraints systems of a restricted form.

Example 1. Let $\mathcal{L} = \{l\}$ and $\mathcal{TC} = \{x \sqsubseteq l(x), l(x) + l(x) \sqsubseteq z\}$ and $\mathcal{AC} = \{1 \leq \Diamond(x)\}$. The set $\mathcal{AC}' = \{1 \leq \lambda, \lambda + \lambda \leq \delta\}$ is equivalent to $(\mathcal{TC}, \mathcal{AC})$. This example can be analysed by our algorithm.

Elimination of a Tree Variable. The problem of eliminating a variable x from a system of constraints $\mathcal{C}$ while keeping the satisfiability of the constraints (Fig. 2) is interesting for various reasons. The first one is efficiency. Eliminating variables can reduce significantly the size of a system of constraints. Thus, it is a good idea to eliminate variables first, and then try to solve the resulting system. On the other hand, eliminating variables can help in bringing constraints in a form that is particularly suitable for applying a given algorithm (see Section 4.3). In Section 5 we give an algorithm for variable elimination that, however, does not succeed in eliminating all variables. If we had an algorithm that solved

Satisfiability
Given: A finite system of constraints $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$.
Wanted: A finite set of linear arithmetic constraints $\mathcal{AC}'$ such that: there is π_a with $\pi_a \models \mathcal{AC}'$ iff there is π_t such that $(\pi_t, \pi_a) \models \mathcal{C}$.

Optimisation
Given: A finite system of constraints $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$ and a linear objective function f defined on the arithmetic variables.
Wanted: A valuation π_a of the arithmetic variables such that $(\pi_t, \pi_a) \models \mathcal{C}$ for some valuation of the tree variables and whenever $(\pi'_t, \pi'_a) \models \mathcal{C}$ then $f(\pi'_a) \leq f(\pi_a)$.

Elimination of a tree variable
Given: A finite system of constraints $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$ and a variable $x \in X$.
Wanted: A finite system of constraints $\mathcal{C}'$ with $x \notin \mathsf{Vars}(\mathcal{C}') \subseteq \mathsf{Vars}(\mathcal{C})$ and $\pi \models \mathcal{C}'$ iff $\exists t.\pi[x \mapsto t] \models \mathcal{C}$.

Fig. 2. Algorithmic problems

the elimination problem, the algorithm would solve satisfiability as well, since a finite system of constraints without tree variables is automatically a finite set of arithmetic constraints.

Example 2. Assume we wish to eliminate y from $\mathcal{C} = \{x \sqsubseteq y, y \sqsubseteq l(x)\}, \{1 \leq \Diamond(y)\}$. Then our algorithm would return $\mathcal{C}' = \{x \sqsubseteq l(x)\}, \{1 \leq \Diamond(l(x))\}$ which is equivalent to $\mathcal{C}$. However, our algorithm is not able to eliminate x from $\mathcal{C}'$.

4 Solving a System of Constraints

In this section we present an algorithm for solving a system of constraints $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$. The linear arithmetic constraints $\mathcal{AC}$ can be solved easily by an LP-Solver. Thus, the challenge is to deal with constraints over trees. Our goal is to reduce the problem of solving these constraints to the problem of solving a finite set of linear arithmetic constraints. We noticed in last section that the canonical set $\Gamma(\mathcal{TC})$ is infinite. But in some particular cases when the constraints admit regular solutions, we can obtain a finite set of arithmetic constraints. Our algorithm seeks solutions to the constraints in the case that the trees must also satisfy some (given) regular structure. When the algorithm is given a regular structure for the tree variables that occur in $\mathcal{TC}$, that we call a *tree schema* Ts, it calculates a finite set of arithmetic constraints. We prove that the algorithm is sound. Clearly, the algorithm is not complete in the general case since not all constraints admit a regular solution. Further, we give in Lemma 4 an upper bound on the size of the resulting set of arithmetic constraints in terms of the sizes of $\mathcal{TC}$ and $\mathsf{Vars}(\mathcal{TC})$.

Tree Constraints in Normal Form. We say that tree expressions are in *normal form* when they are either atomic or a compound expression of the restricted form: $\mathsf{tae} + \mathsf{tae}'$. Moreover, we say that a tree constraint $\mathsf{tc} = \mathsf{te}_1 \sqsubseteq \mathsf{te}_2$

is in normal form if te_1 and te_2 are in normal form and only one of them is compound. Arbitrary tree constraints $\mathsf{tc} \in \mathsf{TConstr}$ can be brought into this form by introducing new variables, for example the tree constraint $x \sqsubseteq y + z + w$ is equivalent to $\{x \sqsubseteq y + v, v = z + w\}$. In the following section we assume that the tree constraints are in normal form. This will simplify our computation of $|\Gamma_{\mathsf{Ts}}(\mathcal{TC})|$ because we will be able to use the fact that $|\mathsf{Vars}(\mathsf{tc})| \leq 3$ for each constraint tc.

4.1 Tree Schema Substitution and $\Delta_{\mathsf{Ts}}(\mathcal{C})$

In the following we define tree schemas: a finite set of tree variables, a finite set of regular trees and a pair of maps, which represent a regular structure for a set of infinite trees.

Definition 3 (Tree Schema). *A* tree schema Ts *consists of*

- *a finite subset* $\mathsf{Ts}.X \subseteq X$.
- *a finite subset* $\mathsf{Ts}.\mathsf{T}^{\mathcal{L}}_{\mathbb{D}} \subseteq \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ *closed under* $l(.)$ *for every* $l \in \mathcal{L}$.
- *a total map* $\mathsf{Ts.next} : \mathcal{L} \times \mathsf{Ts}.X \to \mathsf{Ts}.X \cup \mathsf{Ts}.\mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$.
- *a total injective map* $\mathsf{Ts}.\Diamond : \mathsf{Ts}.X \to \Lambda$.

A valuation $\pi = (\pi_t, \pi_a)$ matches *tree schema* Ts *if the following conditions hold for every* $x \in \mathsf{Ts}.X$*:*

- *if* $\mathsf{Ts}.\Diamond(x) = \lambda \in \Lambda$ *then* $\Diamond(\pi_t(x)) = \pi_a(\lambda)$*;*
- *if* $\mathsf{Ts.next}(l, x) = y \in \mathsf{Ts}.X$ *then* $l(\pi_t(x)) = \pi_t(y)$.
- *if* $\mathsf{Ts.next}(l, x) = t \in \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$ *then* $l(\pi_t(x)) = t$.

Example 3 (Tree schema). Assume $x_1, x_2 \in X$ and $\lambda_1, \lambda_2 \in \Lambda$ and $\mathcal{L} = \{l\}$. Let Ts be a tree schema defined by $\mathsf{Ts}.X = \{x_1, x_2\}$ and $\mathsf{Ts}.\Diamond(x_i) = \lambda_i$ for $i \in \{1, 2\}$ and $\mathsf{Ts.next}(l, x_1) = x_2$ and $\mathsf{Ts.next}(l, x_2) = x_1$. Now define the trees t_1 and t_2 by $\Diamond(t_1) = 1$, $l(t_1) = t_2$ and $\Diamond(t_2) = 2$, $l(t_2) = t_1$. The valuation π given by $\pi_t(x_i) = t_i$ and $\pi_a(\lambda_i) = i$ matches Ts.

The reason why the set $\Gamma(\mathcal{TC})$ is infinite is that it contains expressions te_w for each $w \in L^*$. The main advantage of having a tree schema is that we can eliminate expressions containing labels (like $x_{1ll} = l(l(x_1))$) from a set of constraints. The substitution of such expressions with tree schemas delivers a variable. In this case $l(l(x_1))[\mathsf{Ts}]$ delivers x_1 because $\mathsf{Ts.\,next}(l, x_1) = x_2$ and $\mathsf{Ts.\,next}(l, x_2) = x_1$. We define the functions $\mathsf{tae}[\mathsf{Ts}] : X \cup \mathsf{T}^{\mathcal{L}}_{\mathbb{D}}$, $\mathsf{te}[\mathsf{Ts}] : \mathsf{TExp}$ and $\mathsf{ae}[\mathsf{Ts}] : \mathsf{AExp}$ formally in Fig. 3. These functions simplify the given expressions with respect to a particular tree schema so that $\mathcal{TC}[\mathsf{Ts}]$ returns a set of constraints over trees with no (sub)expressions of the form $l(\mathsf{tae})$, while $\mathcal{AC}[\mathsf{Ts}]$ returns a set of arithmetic constraints that contains no tree variables.

In Fig. 3 we also define the set $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$, a set of arithmetic constraints whose satisfiability implies satisfiability of $\mathcal{TC}$. We build the set $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$ as follows: for each constraint $\mathsf{te} \sqsubseteq \mathsf{te}' \in \mathcal{TC}$ and each path $w \in \mathcal{L}^*$, we add the arithmetic constraints $\mathsf{te}_w[\mathsf{Ts}] \leq \mathsf{te}'_w[\mathsf{Ts}]$ to the set. The use of tree schema substitution

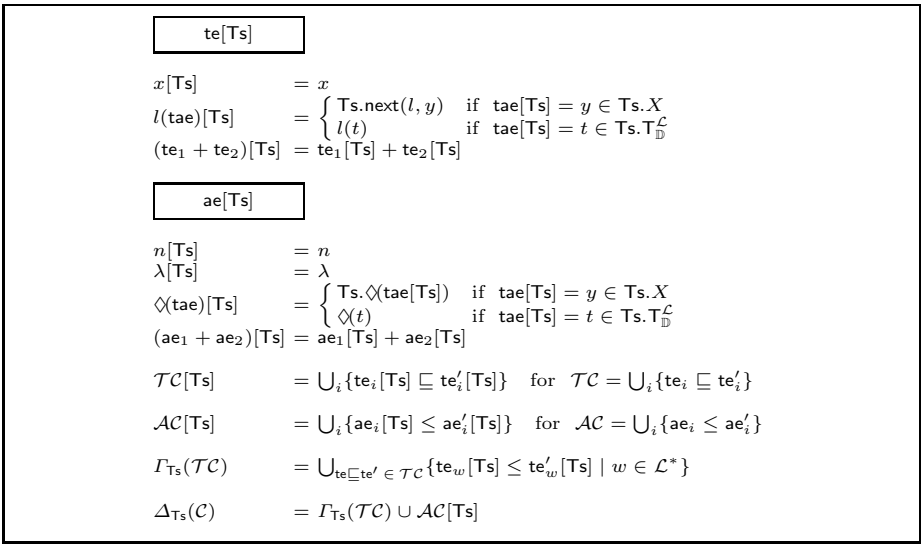

Fig. 3. Tree schema substitution and $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$ and $\Delta_{\mathsf{Ts}}(\mathcal{C})$

ensures that $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$ is finite, in contrast to $\Gamma(\mathcal{TC})$. In the following Lemma we compute an upper bound on the size of $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$ as a function of the sizes of $\mathcal{TC}$ and $\mathsf{Vars}(\mathcal{TC})$.

Lemma 4 (Cardinality of the Set $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$). *Let $\mathcal{TC}$ be a set of constraints and Ts a tree schema with $\mathsf{Ts}.X = \mathsf{Vars}(\mathcal{TC})$. Then $|\Gamma_{\mathsf{Ts}}(\mathcal{TC})| \leq |\mathcal{TC}| \cdot |\mathsf{Ts}.X|^3$.*

The set of arithmetic constraints $\Delta_{\mathsf{Ts}}(\mathcal{C})$, also defined in Fig. 3, is obtained by adding the constraints in $\mathcal{AC}$, after their substitution with the tree schema Ts, to the set $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$. Thus, $\Delta_{\mathsf{Ts}}(\mathcal{C})$ is a finite set of arithmetic constraints without tree variables. We will show below that satisfiability of $\Delta_{\mathsf{Ts}}(\mathcal{C})$ implies satisfiability of $\mathcal{C}$.

Example 4. Let $X, \Lambda, \mathcal{L}$ and Ts be defined as in Example 3. Moreover, let $\mathcal{C} = \{l(x_1) \sqsubseteq x_2, l(x_2) \sqsubseteq x_1\}, \{1 \leq \Diamond(x_1), 2 \leq \Diamond(x_2)\}$. Then $\Gamma_{\mathsf{Ts}}(\mathcal{TC}) = \{\lambda_2 \leq \lambda_2, \lambda_1 \leq \lambda_1\}$ and $\Delta_{\mathsf{Ts}}(\mathcal{C}) = \{\lambda_2 \leq \lambda_2, \lambda_1 \leq \lambda_1, 1 \leq \lambda_1, 2 \leq \lambda_2\}$.

In the following we would like to show the soundness of the algorithm for computing $\Delta_{\mathsf{Ts}}(\mathcal{C})$: if we have a solution for $\Delta_{\mathsf{Ts}}(\mathcal{C})$, we can also find a solution for $\mathcal{C}$. This result is based on the following Lemma, which states that, given a tree schema Ts and a valuation $\pi = (\pi_t, \pi_a)$ that matches Ts, π satisfies $\mathcal{C}$ iff π_a satisfies $\Delta_{\mathsf{Ts}}(\mathcal{TC})$. Moreover we show that all the trees in $\mathrm{ran}(\pi_t)$ are regular.

Lemma 5. *Let Ts be a tree schema with $\mathsf{Ts}.X = \mathsf{Vars}(\mathcal{TC})$ and $\pi = (\pi_t, \pi_a)$ be a valuation that matches Ts. Then:*

1. $\pi_a \models \Gamma_{\mathsf{Ts}}(\mathcal{TC}) \iff \pi \models \mathcal{TC}$.
2. $\pi_a \models \Delta_{\mathsf{Ts}}(\mathcal{C}) \iff \pi \models \mathcal{C}$.
3. *if $t \in \mathrm{ran}(\pi_t)$ then t is regular.*

$$
\begin{array}{lll}
\mathsf{subst}(x, \mathsf{Ts}, \pi_a) = t & & \\
\quad \text{where } \langle\!\langle t \rangle\!\rangle = \pi_a(\mathsf{Ts}.\langle\!\langle x \rangle\!\rangle) & & \\
\quad \text{and } l(t) & = \begin{cases} \mathsf{subst}(y, \mathsf{Ts}, \pi_a) & \text{if } \mathsf{Ts.next}(l, x) = y \in \mathsf{Ts}.X \\ t' & \text{if } \mathsf{Ts.next}(l, x) = t' \in \mathsf{Ts}.\mathsf{T}_{\mathbb{D}}^{\mathcal{L}} \end{cases} & \\
\qquad \text{for each } l \in \mathcal{L} & & \\
\\
\mathsf{Ts}[\pi_a] & = \{x \mapsto \mathsf{subst}(x, \mathsf{Ts}, \pi_a) \mid x \in \mathsf{Ts}.X\} &
\end{array}
$$

Fig. 4. Extending a tree schema Ts to a valuation $\mathsf{Ts}[\pi_a] : X \to \mathsf{T}_{\mathbb{D}}^{\mathcal{L}}$

Given a tree schema Ts and a valuation π_a that satisfies $\Delta_{\mathsf{Ts}}(\mathcal{C})$, we can build a valuation $\mathsf{Ts}[\pi_a] : \mathsf{Ts}.X \to \mathsf{T}_{\mathbb{D}}^{\mathcal{L}}$, as shown in Fig. 4, such that the valuation $(\mathsf{Ts}[\pi_a], \pi_a)$ matches Ts. Thus, by Lemma 5, $(\mathsf{Ts}[\pi_a], \pi_a)$ satisfies $\mathcal{C}$.

Theorem 1 (Soundness of $\Delta_{\mathsf{Ts}}(\mathcal{C})$). *Let Ts be a tree schema with $\mathsf{Ts}.X = \mathsf{Vars}(\mathcal{TC})$ and $\pi_a : \Lambda \to \mathbb{D}$ be a valuation with $\pi_a \models \Delta_{\mathsf{Ts}}(\mathcal{C})$. Then there exists a valuation $\pi_t : \mathsf{Ts}.X \to \mathsf{T}_{\mathbb{D}}^{\mathcal{L}}$ such that $(\pi_t, \pi_a) \models \mathcal{C}$ and if $t \in \mathrm{ran}(\pi_t)$ then t is regular.*

Lemma 5 also provides a sufficient condition on $\mathcal{C}$ which guarantees that its satisfiability implies satisfiability of $\Delta_{\mathsf{Ts}}(\mathcal{C})$. If it is possible to construct a tree schema such that there is a satisfying valuation for $\mathcal{C}$ that matches it, then $\Delta_{\mathsf{Ts}}(\mathcal{C})$ is satisfiable. Moreover, it follows that $\mathcal{C}$ must admit regular solutions.

Lemma 6 (Condition for Completeness of $\Delta_{\mathsf{Ts}}(\mathcal{C})$). *Let Ts be a tree schema with $\mathsf{Ts}.X = \mathsf{Vars}(\mathcal{TC})$ and let $\pi \models \mathcal{C}$ with π matches Ts. Then $\pi_a \models \Delta_{\mathsf{Ts}}(\mathcal{C})$.*

4.2 Computation of $\Delta_{\mathsf{Ts}}(\mathcal{C})$

In last section we described the set $\Delta_{\mathsf{Ts}}(\mathcal{C})$ and proved that its satisfiability implies the satisfiability of $\mathcal{C}$. The natural question that arises is how to compute $\Delta_{\mathsf{Ts}}(\mathcal{C})$. Computing $\mathcal{AC}[\mathsf{Ts}]$ is simple, the challenge is the computation of $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$. Adding constraints to the set for each path $w \in \mathcal{L}^*$ according to the definition is clearly infeasible since there are infinitely many paths. However, we can calculate the desired set by iteration: we build a set $\Gamma^i_{\mathsf{Ts}}(\mathcal{TC})$ iteratively. In the i-th step of the iteration the set contains exactly the constraints corresponding to the paths w with $|w| \leq i$. We prove that the iteration terminates, i.e. that there is an index j with $\Gamma^j_{\mathsf{Ts}}(\mathcal{TC}) = \Gamma^{j+1}_{\mathsf{Ts}}(\mathcal{TC})$ and that this set contains all the constraints in $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$.

The sets $\Gamma^i_{\mathsf{Ts}}(\mathcal{TC})$ are useful for proving the soundness of the iteration and for understanding how it works. However, actually building the sets in each iteration would be inefficient. Instead, we build a set of tree constraints $\mathcal{TC}^i_{\mathsf{Ts}}$ iteratively (Fig. 5), by adding new constraints in each step, so that the following invariant holds: for all i, $\Gamma^0_{\mathsf{Ts}}(\mathcal{TC}^i_{\mathsf{Ts}}) = \Gamma^i_{\mathsf{Ts}}(\mathcal{TC})$. In the following, we prove the soundness of the iteration: $\Gamma_{\mathsf{Ts}}(\mathcal{TC}) = \Gamma^0_{\mathsf{Ts}}(\mathcal{TC}^\infty_{\mathsf{Ts}})$ that follows directly from the invariant.

$$
\begin{array}{ll}
\Gamma^i_{\mathsf{Ts}}(\mathcal{TC}) & = \bigcup_{\mathsf{te} \sqsubseteq \mathsf{te}' \in \mathcal{TC}} \{\mathsf{te}_w[\mathsf{Ts}] \sqsubseteq \mathsf{te}'_w[\mathsf{Ts}] \mid w \in \mathcal{L}^*, |w| \leq i\} \\
\\
\mathsf{treeConstrs}(\mathcal{TC}) & = \{l(\mathsf{te})[\mathsf{Ts}] \sqsubseteq l(\mathsf{te}')[\mathsf{Ts}] \mid \mathsf{te} \sqsubseteq \mathsf{te}' \in \mathcal{TC}, l \in \mathcal{L}\} \\
\\
\mathcal{TC}^0_{\mathsf{Ts}} & = \mathcal{TC}[\mathsf{Ts}] \\
\mathcal{TC}^{i+1}_{\mathsf{Ts}} & = \mathcal{TC}^i_{\mathsf{Ts}} \cup \mathsf{treeConstrs}(\mathcal{TC}^i_{\mathsf{Ts}}) \\
\mathcal{TC}^{\infty}_{\mathsf{Ts}} & = \bigcup_{i \geq 0} \mathcal{TC}^i_{\mathsf{Ts}}
\end{array}
$$

Fig. 5. $\Gamma^i_{\mathsf{Ts}}(\mathcal{TC})$ and $\mathcal{TC}^i_{\mathsf{Ts}}$

Lemma 7 (Soundness of Iteration). *Let $\mathcal{TC}$ be a set of constraints and* Ts *be a tree schema. Then:*

1. *For all i, $\Gamma^0_{\mathsf{Ts}}(\mathcal{TC}^i) = \Gamma^i_{\mathsf{Ts}}(\mathcal{TC})$.*
2. *$\Gamma_{\mathsf{Ts}}(\mathcal{TC}) = \Gamma^0_{\mathsf{Ts}}(\mathcal{TC}^\infty)$.*

Next, we prove termination of the iteration. The proof consists of two parts. First we notice that, since $\mathcal{TC}^i_{\mathsf{Ts}} \subseteq \mathcal{TC}^{i+1}_{\mathsf{Ts}}$ for all i, if there exists an index n_0 with $\mathsf{treeConstrs}(\mathcal{TC}^{n_0}_{\mathsf{Ts}}) \subseteq \mathcal{TC}^{n_0}_{\mathsf{Ts}}$, then $\mathcal{TC}^{n_0}_{\mathsf{Ts}} = \mathcal{TC}^{n_0+1}_{\mathsf{Ts}}$ and for all $i \geq n_0$ $\mathcal{TC}^i_{\mathsf{Ts}} = \mathcal{TC}^{n_0}_{\mathsf{Ts}}$. The second part of the proof consists in showing that such an index exists for this sequence. It follows from the soundness of the iteration and from the fact that the set $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$ is finite.

Lemma 8 (Termination of Iteration). *Let $\mathcal{TC}$ be a set of constraints and* Ts *be a tree schema. Then:*

1. *If there is n_0 with $\mathsf{treeConstrs}(\mathcal{TC}^{n_0}_{\mathsf{Ts}}) \subseteq \mathcal{TC}^{n_0}_{\mathsf{Ts}}$ then $\forall i \geq n_0 \,.\, \mathcal{TC}^i_{\mathsf{Ts}} = \mathcal{TC}^{n_0}_{\mathsf{Ts}}$.*
2. *There is n_0 with $\mathcal{TC}^{n_0}_{\mathsf{Ts}} = \mathcal{TC}^{n_0+1}_{\mathsf{Ts}}$ and $\mathcal{TC}^{\infty}_{\mathsf{Ts}} = \mathcal{TC}^{n_0}_{\mathsf{Ts}}$.*

Example 5 (Computation of $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$). Let Ts and $\mathcal{L}$ be defined as in Example 3 and $\mathcal{TC}$ be defined as in Example 4. Then, we can build $\Gamma_{\mathsf{Ts}}(\mathcal{TC})$ as follows: $\mathcal{TC}^0_{\mathsf{Ts}} = \{x_2 \sqsubseteq x_2, x_1 \sqsubseteq x_1\}$ and $\mathcal{TC}^1_{\mathsf{Ts}} = \{x_1 \sqsubseteq x_1, x_2 \sqsubseteq x_2\}$. Since $\mathcal{TC}^0_{\mathsf{Ts}} = \mathcal{TC}^1_{\mathsf{Ts}}$, it follows $\mathcal{TC}^\infty_{\mathsf{Ts}} = \mathcal{TC}^1_{\mathsf{Ts}}$. Moreover $\Gamma_{\mathsf{Ts}}(\mathcal{TC}) = \Gamma^0_{\mathsf{Ts}}(\{x_1 \sqsubseteq x_1, x_2 \sqsubseteq x_2\}) = \{\lambda_1 \leq \lambda_1, \lambda_2 \leq \lambda_2\}$.

4.3 Linear Constraint System (LCS)

We proved in a previous section that our algorithm for solving satisfiability for a system of constraints is sound for *any* given tree schema. We also noticed that the algorithm can not be complete, because it imposes a regularity condition on the solutions. In this section we study the following questions: Is there a subset of TConstr, for which the algorithm is complete? Then, how do we find the right tree schemas?

A set of tree constraints $\mathcal{TC}$ induces a graph $G = (V, E)$ whose vertices V are the tree variables occurring in $\mathcal{TC}$. The set of edges E is defined as follows: for each $\mathsf{te} \sqsubseteq \mathsf{te}' \in \mathcal{TC}$, for each $x \in \mathsf{Vars}(\mathsf{te})$ and $y \in \mathsf{Vars}(\mathsf{te}')$, we add (x, y) to E.

Then, we say that a set of tree constraints $\mathcal{TC}$ *contains a loop*, when its corresponding graph G contains a closed path. Moreover, we say that a subset $\mathcal{TC}' \subseteq \mathcal{TC}$ *is a loop*, if the graph G contains a closed path P and for all $\mathsf{tc} \in \mathcal{TC}'$ there exists a variable $x_i \in P$ with $x_i \in \mathsf{Vars}(\mathsf{tc})$ and for all $x_i \in P$ there exists $\mathsf{tc} \in \mathcal{TC}'$ with $x_i \in \mathsf{Vars}(\mathsf{tc})$.

We wish to describe a subset LCS of $\mathsf{TConstr}$ such that for each $\mathcal{C} \in \text{LCS}$ we can effectively construct a tree schema $\mathsf{Ts}_{\mathcal{C}}$ with the following property: there exists a valuation π matching $\mathsf{Ts}_{\mathcal{C}}$ and satisfying $\mathcal{C}$. We will describe that set as a collection of loops of a certain restricted form together with constraints defining relations between the variables that appear in the loops. In particular, the loops should not contain compound expressions. Moreover, every variable x that appear in a loop may appear in arithmetic constraints only in subexpressions $\Diamond(x)$. The following grammar describes the restricted sets of tree constraints $\mathsf{LTConstr}$, $\mathsf{RTConstr}$ and the restricted set of arithmetic constraints $\mathsf{LAConstr}$.

$$\begin{array}{lll} \mathsf{ltc} & ::= l(x) \sqsubseteq x & \in \mathsf{LTConstr} \\ \mathsf{rtc} & ::= x \sqsubseteq l(x) & \in \mathsf{RTConstr} \\ \mathsf{pae} & ::= n \mid \lambda \mid \Diamond(x) \mid \mathsf{pae} + \mathsf{pae} & \in \mathsf{PAExp} \\ \mathsf{lac} & ::= \mathsf{pae} \leq \mathsf{pae} & \in \mathsf{LAConstr} \end{array}$$

Definition 4 (Linear Loop). *Let $\mathcal{TC}' \subseteq \mathcal{TC}$ be a loop.*

1. *We say that $\mathcal{TC}'$ is a* left linear loop *if $\mathcal{TC}' \subseteq \mathsf{LTConstr}$ and for all $x \in \mathcal{TC}'$ holds x occurs only positively in $\mathcal{TC} \setminus \mathcal{TC}'$.*
2. *Further, we say that $\mathcal{TC}'$ is a* right linear loop *if $\mathcal{TC}' \subseteq \mathsf{RTConstr}$ and for all $x \in \mathcal{TC}'$ holds x occurs only negatively in $\mathcal{TC} \setminus \mathcal{TC}'$.*
3. *We say that $\mathcal{TC}'$ is a* linear loop *if it is a left linear or a right linear loop and for all $x \in \mathsf{Vars}(\mathcal{TC}')$ holds if $x \in \mathsf{Vars}(\mathsf{ac})$ for some arithmetic constraint ac then $\mathsf{ac} \in \mathsf{LAConstr}$.*

Definition 5 (Linear Constraint System (LCS)). *We say that $\mathcal{TC}$ is* linear *if $\mathcal{TC} = \mathcal{TC}' \cup (\bigcup_{i=1,\ldots,n} \mathcal{TC}_i)$ where each $\mathcal{TC}_i$ is a linear loop with $\mathsf{Vars}(\mathcal{TC}_i) \cap \mathsf{Vars}(\mathcal{TC}_j) = \emptyset$ for $i \neq j$ and $\mathsf{Vars}(\mathcal{TC}') \subseteq \mathsf{Vars}(\bigcup_{i=1,\ldots,n} \mathcal{TC}_i)$.*

$\mathcal{TC}'$ does not contain loops. This follows by the definition since the variables in $\mathcal{TC}$ must appear either only positively or only negatively in $\mathcal{TC}'$.

Example 6. Let $\mathcal{L} = \{l\}$. Let $\mathcal{C} = \mathcal{TC} = \{l(x_1) \sqsubseteq x_2, l(x_2) \sqsubseteq x_1\}, \{\Diamond(x_1) \leq \Diamond(x_2) + 1, 1 \leq \Diamond(x_2)\}$. Then, $\mathcal{TC}$ is a left linear loop and $\mathcal{C} \in \text{LCS}$.

In Fig. 6 we define a tree schema for a LCS $\mathcal{C}$. We show in the following Lemma that if $\mathcal{C}$ is satisfiable there is a valuation that both satisfies $\mathcal{C}$ and matches the tree schema $\mathsf{Ts}_{\mathcal{C}}$. For the construction of such valuation we use a valuation $\pi_a^{(\pi_t, \mathsf{Ts})} : \Lambda \to \mathbb{D}$ (Fig. 6) that we build on the basis of another valuation π_t and a tree schema Ts.

Lemma 9. *Let $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$ be a satisfiable LCS. Then there is a valuation π' with $\pi' \models \mathcal{C}$ and π' matches $\mathsf{Ts}_{\mathcal{C}}$.*

$$
\begin{array}{ll}
\mathsf{Ts}_{\mathcal{C}}.X & = \mathsf{Vars}(\mathcal{TC}) \\
\mathsf{Ts}_{\mathcal{C}}.\mathsf{T}_{\mathbb{D}}^{\mathcal{L}} & = \{\widehat{0}, \widehat{\infty}\} \\
\forall\, x_i \in \mathsf{Ts}_{\mathcal{C}}.X : & \\
\quad \mathsf{Ts}_{\mathcal{C}}.\langle\!\langle(x_i) & = \lambda_i \quad \text{where } \lambda_i \notin \mathsf{Vars}(\mathcal{AC}) \\
\quad \forall l_j \in \mathcal{L}\,.\,\mathsf{Ts}_{\mathcal{C}}.\mathsf{next}(l_j, x_i) & = \begin{cases} x_k & \text{if } (l_j(x_i) \sqsubseteq x_k) \in \mathcal{TC} \text{ or } (x_k \sqsubseteq l_j(x_i)) \in \mathcal{TC} \\ \widehat{\infty} & \text{otherwise, if } x_i \text{ occurs in a left linear loop.} \\ \widehat{0} & \text{otherwise, if } x_i \text{ occurs in a right linear loop.} \end{cases}
\end{array}
$$

$$
\pi_a^{(\pi_t, \mathsf{Ts})} = \{\delta \mapsto \langle\!\langle(\pi_t(x)) \mid x \in \mathsf{Ts}.X, \mathsf{Ts}.\langle\!\langle(x) = \delta\}
$$

Fig. 6. Tree schema for a LCS $\mathcal{C} = (\mathcal{TC}, \mathcal{AC})$

Proof. We have given a valuation $\pi = (\pi_t, \pi_a)$ with $\pi \models \mathcal{C}$. Let $\mathcal{TC} = \mathcal{TC}' \cup (\bigcup_{j=1,\ldots,m} \mathcal{TC}_j(\boldsymbol{x_j}))$. For ease of notation, let us assume that $|\boldsymbol{x_j}| = 1$. Thus, $\boldsymbol{x_j} = x_j$ and let $\pi(x_j) = t_j$. We define $\hat{t}_j$ by:

Case $\mathcal{TC}_j = l(x_j) \sqsubseteq x_j)$. We set $l_k(\hat{t}_j) = \widehat{\infty}$ for $l_k \neq l \in \mathcal{L}$.
Case $\mathcal{TC}_j = x_j \sqsubseteq l(x_j)$. We set $l_k(\hat{t}_j) = \widehat{0}$ for $l_k \neq l \in \mathcal{L}$.

Moreover we set $\langle\!\langle(\hat{t}_j) = \langle\!\langle(t_j)$ and $l(\hat{t}_j) = \hat{t}_j$. Now we set $\hat{\pi}_t = \pi_t[x_j \mapsto \hat{t}_j]$ and $\hat{\pi}_a = \pi_a \cup \pi_a^{(\hat{\pi}_t, \mathsf{Ts}_{\mathcal{C}})}$ and $\hat{\pi} = (\hat{\pi}_t, \hat{\pi}_a)$. We show $\hat{\pi} \models \mathcal{C}$ and $\hat{\pi}$ matches $\mathsf{Ts}_{\mathcal{C}}$: $\hat{\pi}$ matches $\mathsf{Ts}_{\mathcal{C}}$ by construction, $\hat{\pi} \models \mathcal{TC}_j$ follows by construction and $\hat{\pi} \models \mathcal{AC}$ follows by $\pi \models \mathcal{AC}$ and $\langle\!\langle(\hat{t}_j) = \langle\!\langle(t_j)$. Moreover, $\hat{\pi} \models \mathcal{TC}'$ follows by Lemma 3 because if $\mathcal{TC}_j$ is a left linear loop then $\mathcal{TC}'(x_j^+)$ and $t_i \sqsubseteq \hat{t}_i$ and if $\mathcal{TC}_j$ is a right linear loop then $\mathcal{TC}'(x_j^-)$ and $\hat{t}_i \sqsubseteq t_i$.

Theorem 2 (Completeness of $\Delta_{\mathsf{Ts}}(\mathcal{C})$). *Let $\mathcal{C}$ be a satisfiable LCS. Then there is a tree schema* Ts *and a valuation π_a with $\pi_a \models \Delta_{\mathsf{Ts}}(\mathcal{C})$.*

Proof. By Lemma 9 we obtain a valuation $\pi \models \mathcal{C}$ with $\pi = (\pi_t, \pi_a)$ matches $\mathsf{Ts}_{\mathcal{C}}$. Moreover, by Lemma 6, we obtain $\pi_a \models \Delta_{\mathsf{Ts}_{\mathcal{C}}}(\mathcal{C})$.

The restriction to linear constraint systems could seem very strong. However, we will show an algorithm for eliminating variables from constraints while maintaining their satisfiability in the next section. In most cases we are able to eliminate the variables that are not part of a loop with that procedure. Further, we can often bring the loops in the required form by eliminating intermediate variables. For example, the loop $\{l(x) \sqsubseteq y, y \sqsubseteq x\}$ can be transformed into $l(x) \sqsubseteq x$ if we eliminate y. On the other hand, there are systems such as $\{x + x \sqsubseteq l(x)\}, \{1 \leq \langle\!\langle(x)\}$ that can not be transformed into an equivalent linear one. In fact, there is no regular solution for that system.

5 Elimination of Tree Variables

In this section we define an algorithm for eliminating tree variables from a set of tree constraints while keeping their satisfiability.

$$\frac{\mathcal{C}(y^+) \text{ or } \mathcal{C}(y^-)}{\text{erase } y \text{ from } \mathcal{C}} \ (\triangleright \mathsf{Prune}) \qquad \frac{(\bigcup_{i=1..n}\{y \sqsubseteq \mathsf{te}_i\}) \cup \mathcal{D}(y^+), \mathcal{AC}(y^+)}{\bigcup_{i=1..n}(\mathcal{D}, \mathcal{AC})\,[\mathsf{te}_i/y]} \ (\triangleright \mathsf{Elim}^+)$$

$$\frac{(\bigcup_{i=1..n}\{\mathsf{te}_i \sqsubseteq y\}) \cup \mathcal{D}(y^-), \mathcal{AC}(y^-)}{\bigcup_{i=1..n}(\mathcal{D}, \mathcal{AC})\,[\mathsf{te}_i/y]} \ (\triangleright \mathsf{Elim}^-)$$

$$\frac{\mathcal{C}(y^+, y^-) \qquad \mathcal{C}(y^{\mathsf{proj}}) \cap \mathcal{C}(y^{\mathsf{whole}}) = \emptyset \qquad l_i \in \mathcal{L} \text{ and } \boldsymbol{z}, \lambda \text{ new}}{\mathcal{C}(y^{\mathsf{proj}}) \cup \mathsf{unfold}(\mathcal{C}(y^{\mathsf{whole}}))\,[z_i/l_i(y)][\lambda/\Diamond\!\!(y)]} \ (\triangleright \mathsf{Elim}^{+/-})$$

Fig. 7. Elimination of tree variables from a set of tree constraints

We say that a variable x occurs *projected* in a set of tree constraints when x appears exclusively in (sub)expressions $l(\mathsf{tae})$, $\Diamond\!\!(\mathsf{tae})$. If x appears exclusively as a variable (sub)expression "x" we say that x occurs *as a whole*. We write $\mathcal{C}(x^{\mathsf{proj}})$ for the subset of $\mathcal{C}$ where x occurs projected and we write $\mathcal{C}(x^{\mathsf{whole}})$ for the subset of $\mathcal{C}$ where x appears as a whole. The following function $\mathsf{unfold}(\mathcal{TC})$ unrolls the definition of inequality ($\sqsubseteq$) in the constraints once. The validity of the resulting constraints is ensured by applying the transformations (3.1).

Definition 6 (Unfold Constraints). *Let $\mathcal{TC}$ be a set of tree constraints. We define a function* $\mathsf{unfold}(\mathcal{TC})$ *by unfolding the definition of inequality:*
$\mathsf{unfold}(\mathcal{TC}) = \bigcup_{\mathsf{te} \sqsubseteq \mathsf{te}' \in \mathcal{TC}} \bigcup_{l \in \mathcal{L}} \{l(\mathsf{te}) \sqsubseteq l(\mathsf{te}')\}, \bigcup_{\mathsf{te} \sqsubseteq \mathsf{te}' \in \mathcal{TC}} \{\Diamond\!\!(\mathsf{te}) \leq \Diamond\!\!(\mathsf{te}')\}$.

In the following we define the algorithm $\mathsf{elim}_{\cdot}(.)$ as a set of inference rules (Fig. 7). If the tree variable y appears only positively or negatively in the constraints then it can be safely removed altogether from the system of constraints ($\triangleright \mathsf{Prune}$). If the variable appears in a constraint such as $\mathsf{tae}_1(y) + \mathsf{tae}_2 \sqsubseteq \mathsf{tae}$, then we return $\mathsf{tae}_2 \sqsubseteq \mathsf{tae}$. Otherwise, when it appears in a constraint $\mathsf{tae}_1 \sqsubseteq \mathsf{tae}_2$ then we remove the whole constraint. Further, if the variable appears in an arithmetic constraint $\Diamond\!\!(\mathsf{tae}(y)) + \mathsf{ae}_2 \leq \mathsf{ae}$, we return $\mathsf{ae}_2 \leq \mathsf{ae}$.

Next, we consider the case when the variable has at least one upper or lower bound and appears otherwise only positively ($\triangleright \mathsf{Elim}^+$) or only negatively ($\triangleright \mathsf{Elim}^-$) in the constraints. Then, the elimination takes place by substituting the variable in the constraints with its upper bounds, if the variable occurs only positively, or with its lower bounds, if the variable appears only negatively.

The last and more complicated case is when the variable appears both positively and negatively. Then we calculate $\mathcal{C}(y^{\mathsf{proj}})$ and $\mathcal{C}(y^{\mathsf{whole}})$. If they are disjoint sets, we unfold $\mathcal{C}(y^{\mathsf{whole}})$ and substitute $l_i(y)$ and $\Diamond\!\!(y)$ with fresh variables z_i and λ, respectively. If $\mathcal{C}(y^{\mathsf{proj}})$ and $\mathcal{C}(y^{\mathsf{whole}})$ are not disjoint sets, then the variable cannot be eliminated. The reason for this restriction is that, with this rule, we create new variables and we want to eliminate them as well. However, if $\mathcal{C}(y^{\mathsf{proj}})$ and $\mathcal{C}(y^{\mathsf{whole}})$ are not disjoint sets, we would keep eliminating variables and creating new ones without ever coming to an end. Suppose we have the constraint

Table 1. Experimental results. $|\mathsf{Vars}(\mathcal{C})|$ represents the number of tree variables in the constraints before the elimination. $|\mathsf{Ts}.X|$ represents the number of variables remaining after the elimination which is equal to the number of variables in the created tree schema Ts.

Program	**LoC**	$\lvert\mathsf{Vars}(\mathcal{C})\rvert$	$\lvert\mathsf{Ts}.X\rvert$
List Duplication	37	362	4
Doubly-linked Lists	47	568	6
Constant-time List Append	60	674	12
Insertion Sort	66	872	32
List Append	80	1116	8
Merge Sort	127	2818	10
Bank Account	200	3566	10

$l(x) \sqsubseteq x$ and $\mathcal{L} = \{l\}$ and we want to eliminate the variable x. If we applied the rule $(\triangleright\mathsf{Elim}^{+/-})$ we would obtain $l(l(x)) \sqsubseteq l(x)$ after unfolding and $l(z) \sqsubseteq z$ after substituting $l(x)$ with a fresh variable z. If we now tried to eliminate z, we would go through the same procedure again and would never be able to eliminate the variable. The correctness of the elimination procedure follows from the fact that the trees form a complete lattice.

Theorem 3 (Correctness of $\mathsf{elim}()$). *Let $\mathcal{C}(\boldsymbol{x}, y, \boldsymbol{\lambda})$ be a system of constraints. If $\mathsf{elim}_y(\mathcal{C}) = \mathcal{C}'(\boldsymbol{x}, \boldsymbol{\lambda})$ then for all $\pi : \pi \models \mathcal{C}' \iff$ there exists t with $\pi \cup \{y \mapsto t\} \models \mathcal{C}$.*

6 Applications to Resource Analysis

We have implemented the algorithms described in this paper in Ocaml and used them for solving the constraints that arose during our static heap-space analysis of object-oriented programs. Our implementation consists of the following steps:

1. Eliminate variables from the constraints until the only remaining variables are those that appear in a loop.
2. Check if the resulting constraint system is a LCS. In the positive case, construct the tree schema as described in Section 4.3, otherwise construct a tree schema using a heuristic procedure.
3. Compute $\Delta_{\mathsf{Ts}}(\mathcal{C})$ and solve it with an LP-Solver.

Table 1 shows the programs that we could analyse with our tool. For each example, we could solve the constraints and resultantly provide a (linear) upper bound for its heap-space requirements. Notice that the number of tree variables that were generated is proportional to the size of the programs, while the number of variables that remain after the elimination reflects the amount of loops in the constraints and the amount of variables in the loops. There is a demo website where all the examples can be analysed and downloaded [raj].

7 Conclusions

We have presented a system of constraints over infinite trees and we have studied their satisfiability and elimination problems. We have given an algorithm that solves satisfiability for a subcase. Moreover, we have presented a correct algorithm that eliminates a tree variable in most cases. We hope to settle the question of decidability of our tree constraints in general and plan to identify larger tractable subproblems relevant for resource analysis.

Acknowledgements. We acknowledge support by the DFG Graduiertenkolleg 1480 Programm- und Modell-Analyse (PUMA). We also thank Luke Ong for valuable comments.

References

BG00. Blumensath, A., Grädel, E.: Automatic structures. In: LICS, pp. 51–62 (2000)
DV07. Dantchev, S., Valencia, F.D.: On infinite csp's (2007)
HJ06. Hofmann, M.O., Jost, S.: Type-Based Amortised Heap-Space Analysis. In: Sestoft, P. (ed.) ESOP 2006. LNCS, vol. 3924, pp. 22–37. Springer, Heidelberg (2006)
HR09. Hofmann, M., Rodriguez, D.: Efficient Type-Checking for Amortised Heap-Space Analysis. In: Grädel, E., Kahle, R. (eds.) CSL 2009. LNCS, vol. 5771, pp. 317–331. Springer, Heidelberg (2009)
raj. `http://raja.tcs.ifi.lmu.de`
SR10. Silva, A., Rutten, J.J.M.M.: A coinductive calculus of binary trees. Inf. Comput. 208(5), 578–593 (2010)