EPiC Series in Computing
Volume 57, 2018, Pages 111–130
LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Decidable Inequalities over Infinite Trees

Sabine Bauer[*], Steffen Jost and Martin Hofmann

LMU Munich, Germany
`Sabine.Bauer@ifi.lmu.de, Jost@tcs.ifi.lmu.de`

Abstract

Linear tree constraints are given by pointwise linear inequalities between infinite trees labeled with nonnegative rational numbers. Satisfiablity of such constraints is at least as hard as solving the Skolem-Mahler-Lech Problem. We provide an interesting subcase, for which we prove that satisfiablity is decidable. Our decision procedure is based on intricate arguments using automata and combinatorics of words.

Our subcase allows to construct an inference mechanism for resource bounds of object oriented Java-like programs: actual resource bounds can be read off from solutions of tree constraints. So far, only the case of degenerated tree constraints (i.e. lists) was known to be decidable which, however, is insufficient to generally solve the given resource analysis problem. The present paper therefore provides a generalisation to trees of higher degree in order to cover the entire range of constraints encountered by resource analysis.

1 Introduction

1.1 Amortised Analysis

Amortised Analysis provides guaranteed worst-case bounds for sequences of operations that are tighter than simply summing the worst-case costs of individual instructions (bounds on discrete resource consumption such as memory usage, execution steps, etc.). This is achieved through a clever abstraction of all possible machine states, which reduces the state space down to a manageable size. However, finding suitable state abstractions is difficult and is highly specific to the data-structures that are used.

We give an intuition for this analysis in the object-oriented setting by an example of a doubly linked list that is constructed from a singly linked list and then transformed back again into a singly linked list. This example shows how the analysis adapts to changes of highly aliased data structures during a computation. As usual in amortised analysis, we assign a nonnegative value, called the *potential*, to each access path that leads to a certain object. Figuratively speaking, the potential can be considered as an amount of dollars from which all operations associated with this object must be payed (e.g. copying an object requires the allocation of a new object, which has a certain cost).

[*]DFG Graduiertenkolleg 1480, with Technical University of Munich

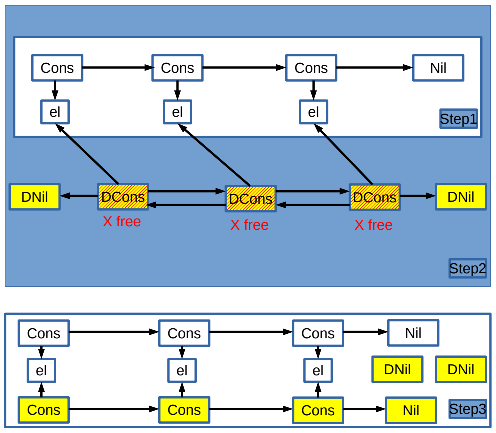

Figure 1: DList

The design of our analysis ensures that despite of aliasing and update, there is no doubling or multiplying of the potential. This is ensured through careful balancing expressed by our annotated typing rules. The potential of a list depends on its length and can be defined only via its access paths. This is illustrated in Figure 1: in step 1, we see a singly linked list consisting of `Cons` objects with a pointer to the next list and to the element, e.g. an integer number. This list is still present in step 2, and in addition to it, we have also constructed a doubly liked list with `DCons` objects and two empty lists (`DNil`) on each end. These `DCons` objects are again deallocated in step 3, when we build a new singly linked list and set the pointers such that we obtain the original list again. All yellow objects are allocated freshly.

Our automated analysis receives the program code of this example and a fixed cost model (e.g. for each class the allocation cost per object, etc.) and generates some linear tree constraints. The solution to these linear tree constraints tells us about the resource consumption of this program. Say for the sake of simplicity for this example, that each object allocation costs 1, then the solution to the generated linear tree constraints would tell us that the execution of the whole program allocates $n + 3$ objects, were n is the length of the input list.

This example can be treated in previous work by Hofmann and Rodriguez and the output constraints are already in this case hundreds in number[1]. There, the analysable programs are limited to a subset of those that consume at most a linear amount of resources. In this paper, we will overcome this restriction.

We focus now on solving the arising linear tree constraints. Further background of the amortised analysis technique is given in Section 1.3; and the interpretation of solutions to linear tree constraints with respect to analysed program code is discussed in Section 3.

1.2 Syntax, Semantics and Preliminary Results

Definition 1. *Let Σ be a finite set of branch labels and $\mathbb{D} = \mathbb{Q}_0^+ \cup \{\infty\}$. An infinite tree $T_{\mathbb{D}}^{\Sigma}$ with branches in Σ and labels in $\mathbb{D}$ is a map $\Sigma^* \to \mathbb{D}$ (with Σ^* being all words over Σ^*). A tree s is a subtree of a tree t if there is w in Σ^* such that $s(p) = t(wp)$ for all $p \in \Sigma^*$.*

[1]The code can be found here: http://raja.tcs.ifi.lmu.de/demo/demo-dlist

$$\begin{aligned} t &::= x \mid l(t),\ \text{where } l \in \Sigma \text{ with } |\Sigma| < \infty && \text{(Atomic tree)}\\ te &::= t \mid te + te && \text{(Tree term)}\\ c &::= te \geq te && \text{(Tree constraint)} \end{aligned}$$

Figure 2: Linear Tree Constraint Syntax

$$\begin{aligned} v &::= n \mid \lambda \mid \Diamond(t) && \text{(Atomic arithmetic expression)}\\ h &::= v \mid h + h && \text{(Arithmetic term)}\\ c &::= h \geq h && \text{(Arithmetic constraint)} \end{aligned}$$

Figure 3: Arithmetic Constraint Syntax

A tree constraint system is a set of pointwise linear inequalities between tree variables, for example $x \geq r(x) + l(x) + y$, where x, y are binary trees with labels l and r such that $r(x)$ is the right subtree of x (and $l(x)$ the left subtree.) In our setting, these tree variables can be instantiated with infinite trees that contain a nonnegative rational number or infinity in each node as in Definition 1. The degree of the trees is arbitrary but finite. The formal syntax for the linear tree constraints is shown in Figure 2.

A solution of the tree constraints is a set of infinite trees for which the constraints hold pointwise for each number in the nodes. More precisely, a constraint $x \geq y$ holds for concrete trees t_1, t_2, if $\Diamond(t_1) \geq \Diamond(t_2)$ and for all labels l (denoting the immediate subtrees) holds $l(t_1) \geq l(t_2)$ (cf. rule (Label) in Figure 6). The latter means the constraint holds recursively for all immediate subtrees. Similarly, a tree constraint $\sum_i x_i \geq \sum_i y_i$ holds if the sum of the roots of the x_i is greater or equal to the sum of the roots of the y_i and the same constraint holds recursively for their children. We further define that for each $a \in \mathbb{Q}_0^+ \cup \{\infty\}$ holds $a + \infty = \infty$.

In addition to the tree constraints, we have arithmetic constraints given for the numbers in selected nodes of the trees that take the form of an arbitrary linear program with integer coefficients. They are the same as tree constraints with the difference that they can include numbers and hold only for the roots, which are arithmetic variables. Let λ be a variable, n a number and let $\Diamond(x)$ denote the root of tree x. Figure 3 gives the syntax for them.

Example 1. *Consider degenerated trees (i.e. lists) with root symbol* head *(i.e.* $\text{head}(t) := \Diamond(t)$*) and the only label being* tail. *The arithmetic list constraint* $\text{head}(x) + \text{head}(y) \geq 2 + \text{head}(\text{tail}(z))$. *states that the sum of the first elements of lists x and y are greater or equal to the second element of z plus 2.*

Tree constraint systems without arithmetic constraints are always trivially satisfiable with trees consisting only of zeros or only of infinity. The arithmetic constraints play a similar role as the initial values for recurrences, starting from these values the whole tree is build up. If they are not themselves zero or infinity, they also prevent setting entire trees to zero of infinity.

Each inequality over tree variables corresponds to infinitely many inequalities over arithmetic variables (i.e. variables for the numbers in the nodes.) Thus the problem to decide whether a set of tree constraints is simultaneously satisfiable can not directly be reduced to feasibility of a (finite) linear program.

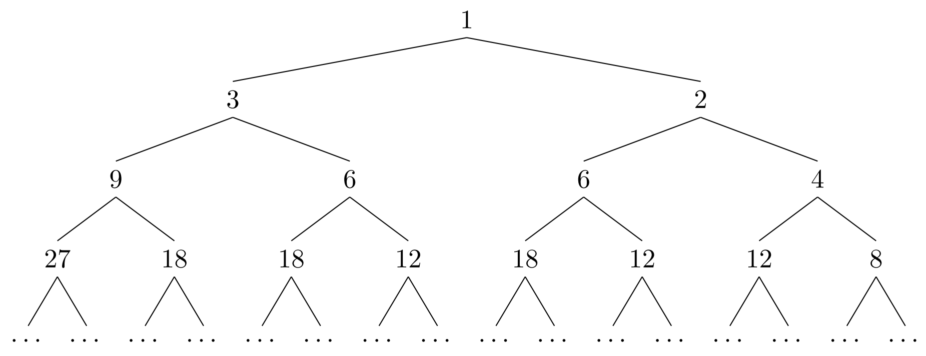

Figure 4: Tree with infinite number of different subtrees

Example 2. *The system consisting of the arithmetic constraint $\Diamond(t_1) = 1$ and the tree constraints*

$$r(t_1) \geq 2t_1 \qquad l(t_1) \geq 3t_1 \qquad t_1 \geq lr(t_1)$$

where the $\Diamond(\cdot)$ symbol denotes the variable in the root of a tree, is unsatisfiable, because it implies $1 = \Diamond(t_1) \geq \Diamond(lr(t_1)) \geq 2\Diamond(l(t_1)) \geq 6\Diamond(t_1) = 6$. The system $\Diamond(t_2) = 1, r(t_2) = 2t_2, l(t_2) = 3t_2$ has the solution in Figure 4. The subtrees can be computed by duplicating the value in the root of the subtree when going right and multiplying by three when going left:

$$\forall w \in (l|r)^* : \Diamond(w(t_2)) = 2^i 3^j,\ i = \textit{number of } r\textit{'s in } w,\ j = \textit{number of } l\textit{'s in } w.$$

The following problems are closely related [3].

- **Skolem-Mahler-Lech Problem (SML)**

  Given: A homogeneous linear recurrent sequence of degree k with initial values $b_1, \dots, b_k$ and constant rational coefficients $a_1, \dots, a_k$ of the form

  $$\begin{aligned} &x_n = a_1 x_{n-1} + \cdots + a_k x_{n-k}, n > k \\ &x_1 = b_1, \dots, x_k = b_k \\ &a_i, b_i \in \mathbb{Q}, b_i \geq 0 \text{ for all } i = 1, \dots, k, a_k \neq 0, \end{aligned}$$

  Asked: Is there an index n such that $x_n = 0$?

- **List Constraint Satisfiability Problem (LC)**

  Given: A finite system of list constraints (constraints over trees of degree 1 with label *tail*) and arithmetic constraints

  Asked: Is there a set of lists, which simultaneously satisfies all constraints in of the system in $\mathbb{D} = \mathbb{Q}_0^+ \cup \{\infty\}$?

- **Tree Constraint Satisfiability Problem (TC)**

  Given: A finite system of tree constraints and arithmetic constraints

Asked: Is there a set of trees, which simultaneously satisfies all constraints of the system in $\mathbb{D}$?

SML can be reduced to LC [3]. LC is an obvious subcase of TC. Thus LC and TC are very hard and probably undecidable problems, since the decidability status of the famous and NP-hard SML problem is still unknown. The detailed situation for recurrences and decision problems there (as SML) is described in [28, 29, 4, 12].

This led us to the consideration of *unilateral* constraints, that can be shown sufficient for our purposes (namely the program analysis). We examined the rules that generate constraints from programs more closely and found a more accurate description of the constraints that are actually produced than the original formulation by Hofmann and Rodriguez (cf. [3]). More precisely, unilateral constraints are the only sort of constraints that arise from Java-like programs.

Definition 2. *A unilateral tree constraint is a constraint with only one summand on the greater side of the inequality (i.e. of the form $c ::= t \geq te$ according to Figure 2 (Tree constraint), or equivalently $t \geq t_1 + \cdots + t_n$). We call unilateral tree constraints UTC. In contrast to the tree constraints, the arithmetic constraints remain unchanged.*

Note that the satisfiable tree constraints in Example 2 are not unilateral, because of the equality sign that is constructed by combination of constraints with $\geq$ and $\leq$, where the latter are not unilateral. We also emphasise that there are no minus signs in the sum on the right hand side. It follows from the nonnegativity of the summands on the right, that UTC is a proper fragment of TC.

Unilateral constraints are considerably easier to solve, because all coefficients on the right hand side (i.e. the smaller side of the inequality) are positive and thus complicated recurrences can not all be expressed in ULC; in particular, SML can not be reduced to the unilateral fragment. Furthermore, since all trees have only nonnegative entries, the left hand side is greater than each single summand on the right. We will make use of this fact in our decidability proof (Theorem 1) explicitely.

Indeed, it was shown that for the list case there exists a polynomial decision procedure by reduction to linear programming [3]. We show that UTC is decidable. In contrast to ULC (Unilateral List Constraints), our decision procedure is not polynomial in the size of the input.

We remark that according to the recurrence-like syntax nearly all constraints have only nonlinear solutions. For instance, in the case of lists, the linear system by Hofmann and Rodriguez can only (partially) treat periodic lists. In the tree case, we have analogous growth rates as for lists in [3]. This means, as soon as we have a tree constraint with sums like e.g. $lrlrx \geq lrx + lrx$, the tree exhibits exponential growth.

1.3 Related Work

The first automated application of amortised analysis to first-order functional programs was by Hofmann and Jost [22]. They annotated the types in the programs with the available resources of the data structure and then introduced typing rules to reason about the resource consumption of functions. There they had the restriction that the potential was required to be linear. The approach was later generalised to multivariate polynomial potential by Hoffmann [18, 19, 15, 16, 14]. This was the starting point for many other investigations in this direction. Hofmann and Moser applied amortised analysis to term rewriting [24], Hoffmann refined his work, made it fully automatic and carried over the analysis to concurrent programs and programs in C and OCaml [17, 6, 7, 20, 21]. Among other related work, mainly on resource analysis, are [10, 5, 13, 1, 11], which use different methods than our approach.

Applying the amortised analysis to a fragment of Java (RAJA, i.e. Resource Aware Java, similar to FJEU (Featherweight Java Extended with Update)), which features object oriented programming, polymorphic functions and monomorphic recursion was started by Hofmann and Jost [23]. Rodriguez further examined RAJA and introduced the tree constraint problem, which remained open in her work [27, 25, 30, 26]. Her resource-type inference algorithm outputs conditions that must hold for the potential of the objects (represented as trees) in form of linear tree constraints. One can determine the resource consumption of a RAJA program if one has a solution to its constraints. Recently, the list constraint satisfiability problem for RAJA was proven decidable [3]. In this paper, we generalise that argument to trees.

2 Decidability

We now establish our main result (Theorem 4), namely that satisfiability of unilateral linear tree constraints is decidable. The proof is structured as follows: We first observe that unsatisfiability is semi-decidable. We then show that we can reduce a set of constraint systems that contains all satisfiable ones to linear programming using the following arguments:

- We describe how to derive inequalities that follow from a set of constraints using a sound and complete proof system,
- characterise the set of trees greater than a fixed tree as a regular language,
- use these languages to find all trees bounded from above and from below,
- show that all other trees can be set to zero or infinity without changing the satisfiability properties of the system, and finally
- reduce the constraints to an equisatisfiable linear program.

This means satisfiability is semi-decidable. Together with the semi-decidability of unsatisfiability, this implies that satisfiability is decidable.

2.1 Unsatisfiability is Semi-Decidable

From now on we are in the realm of UTC and omit the word "unilateral". An *unfolding step* for a constraint $x \geq S_1 + \cdots + S_n$, where S_i are (sums of) tree variables, consists of adding the arithmetic constraint $\Diamond(x) \geq \Diamond(S_1) + \cdots + \Diamond(S_n)$ and application of the (LabelSum) rule in Figure 5 for all $l \in \Sigma$ to obtain the constraints for the next step.

$$\frac{S = \sum_i S_i \qquad TC \vdash x \geq S}{TC \vdash l(x) \geq \sum_i l(S_i)} \text{ (LabelSum)}$$

$$\frac{S = \sum_i S_i \qquad TC \vdash x \geq S}{TC \vdash \Diamond(x) \geq \sum_i \Diamond(S_i)} \text{ (Root)}$$

Figure 5: Label application rules

Each such step delivers a new, bigger set of arithmetic constraints that can be seen as a linear program. We have a succession of programs $(P_i)_{i\geq 0}$.

Lemma 1. *The constraint system* (AC, TC)*, where* AC *is a set of arithmetic and* TC *a set of tree constraints, is unsatisfiable if and only if one of the linear programs* P_i *is unsatisfiable.*

The proof closely follows the same ideas as the compactness proof for infinite dimensional 0-1-programming in [8]. Thus, if there is a contradiction, we find it, but if the system is satisfiable, this will not terminate.

2.2 The Set of Trees Greater than a Fixed Tree is a Regular Language

We now describe the implications of a constraint system as given in Figure 6. Intuitively, if a constraint $x \geq y$ holds for trees x and y, then also each subtree of x is greater than or equal to the subtree of y with the same label, and similarly the root of x must be greater or equal to the root of y. Further, the greater-or-equal relation must be transitive.

$$\frac{}{TC \vdash ux \geq ux}\ \text{(Reflexivity)}$$

$$\frac{TC \vdash x \geq y}{TC \vdash l(x) \geq l(y)}\ \text{(Label)}$$

$$\frac{x \geq y_1 + ... + y_n \in TC \qquad TC \vdash uy_i \geq z}{TC \vdash ux \geq z}\ \text{(Transitivity)}$$

Figure 6: Proof system for unilateral tree constraints

Tree expressions are of the form ux where $u : \Sigma^*$ and Σ is the set of tree labels like *left*, *right*, etc., and x is a variable. We use letters t, x, y, z, possibly decorated with indices, for both variables and tree expressions.

The judgement $TC \models x \geq y$, where x and y are expressions, has the meaning that $x \geq y$ follows semantically from the tree constraints in TC. That is, every valuation that satisfies TC also satisfies $x \geq y$. The judgement $TC \vdash x \geq y$ means that the inequality $x \geq y$ is derivable by the rules in Figure 6.

Theorem 1. *The proof system in Figure 6 is sound and complete (i.e.* $TC \models x \geq y \Leftrightarrow TC \vdash x \geq y$*).*

Proof. Soundness is trivial. For completeness we argue as follows. Let T be the set of all tree expressions over the variables in TC and define a graph $G = (V, E)$ where $V = T$ and

$$\begin{aligned} E =& \{(x, y) | \text{ there is a constraint } x' \geq y_1 + \cdots + y_n \in TC \text{ and} \\ & u : \Sigma^*.x = ux' \text{ and } y = uy_i \text{ for some } i\}. \end{aligned}$$

Now fix a tree expression x_0 and define a valuation η in such a way that $\eta(\Diamond(z)) = 0$ if z is reachable from x_0 in G and $\eta(\Diamond(z)) = \infty$ otherwise. We claim that η satisfies TC. Indeed, suppose that $x \geq y_1 + \cdots + y_n$ is a constraint in TC. We must show that $\eta(\Diamond(ux)) \geq \eta(\Diamond(uy_1)) + \cdots + \eta(\Diamond(uy_n))$ holds for all $u : \Sigma^*$. If ux is unreachable from x_0 then $\eta(\Diamond(ux)) = \infty$ and the inequality holds. Otherwise, if ux is reachable then $uy_1, \ldots, uy_n$ are also reachable and the inequality holds as well.

Now suppose that $x_0 \geq y$ is an inequality that is not derivable from TC. In this case, y is not reachable from x_0 in G. The valuation η constructed above then satisfies TC yet $\eta(x_0) = 0$ and $\eta(y) = \infty$ so $x_0 \geq y$ is not a semantic consequence of TC. □

For the next step, we are interested in the set $L_z^{\geq}$ of tree expressions greater or equal to a fixed tree expression z (resp. $L_z^{\leq}$); in short all x such that $TC \vdash x \geq z$. Let us define the language $L_{x,t} \coloneqq \{u \mid TC \vdash ux \geq t\}$ with x a fixed tree variable and t a (possibly prefixed) tree[2], as an auxiliary step to compute $L_z^{\geq}$.

Theorem 2. *The language $L_{x,t}$ is regular.*

Proof. We construct a finite automaton that accepts a word w, if and only if $TC \vdash wx \geq t$.

With the proof system in Figure 6, we can first build a pushdown automaton $\mathcal{A}$ from TC, that reads no input and such that $u : L_{x,t}$ if and only if $\mathcal{A}$ accepts beginning from stack ux.

We give the idea for the construction of a slightly more general pushdown automaton, namely an automaton that accepts a word $vx(wt)^r$ (w^r denotes the word w reversed) if and only if the constraints imply $vx \geq wt$. Acceptance is by empty stack, and we start by writing v on the stack while we are in a so-called "write-state", then go into a state named "x", there modify vx nondeterministically and without reading from the input, as the constraints describe (possibly going to state "t" for another variable t). After that, we leave state "t" (or "x") and go into a "compare-state" where we compare the obtained stack with w and empty it if they both are equal.

Example 3. *Consider the constraints*

$$lr(x) \geq rr(x) \qquad\qquad lr(x) \geq l(y)$$

The pushdown automaton $\mathcal{A}_0 = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ such that

$$Z = \{z_0, z_\infty, z_x, z_y, z', z''\}, \Sigma = \{l, r, x, y\}, \Gamma = \{L, R, \#\}$$

and the transition relation δ is defined as depicted in Figure 7. Note that the lower case input symbols in Σ correspond to the according upper case letters in the stack alphabet Γ. Here z_0 is the write-state, z_∞ the compare-state and z', z'' are auxiliary states. For instance, in Figure 7, the auxiliary states are used as intermediate steps to rewrite $lr(x)$ to $rr(x)$ or to $l(y)$. We use the usual notation with triples for the current state, the read input symbol and the stack content, that are then mapped to the next state and the new stack content by δ. In the picture, the triples on the arrows mean the input symbol, the stack before and the stack after the transition.

The language L corresponding to this generalisation is *not always* a regular language: this can be seen with the Pumping Lemma. Assume L would be regular and let p be the Pumping Lemma number and let the constraints be $lrx \geq lrlrx$ and consider words $(lr)^i xx(rl)^{i+1}$, which are implied by the constraints and thus in L. Then the word $\alpha = (lr)^p$ is obviously longer than p. For each division of $\alpha xx(rl)^{p+1} = uvw$ in three words u, v, w such that $|uv|$ is not longer than p, for instance with $v = lr$, holds $\forall k.(lr)^{p+k} xx(rl)^{p+1} \in L$, which is not implied by the

[2]Note that t may just be a tree variable.

$$
\begin{aligned}
\delta(z_0, a, B) &= (z_0, AB),\\
\delta(z_0, x, B) &= (z_x, B),\\
\delta(z_0, y, B) &= (z_y, B),\\
\delta(z_x, \epsilon, RB) &= (z', B),\\
\delta(z', \epsilon, LB) &= (z'', B),\\
\delta(z'', \epsilon, B) &= \{(z_x, RRB), (z_y, LB)\},\\
\delta(z_x, x, B) &= (z_\infty, B),\\
\delta(z_y, y, B) &= (z_\infty, B),\\
\delta(z_\infty, r, RB) &= (z_\infty, B),\\
\delta(z_\infty, l, LB) &= (z_\infty, B),\\
\delta(z_\infty, \epsilon, \#) &= (z_\infty, \epsilon),\\
B \in \Gamma^*, a &\in \{l, r\}.
\end{aligned}
$$

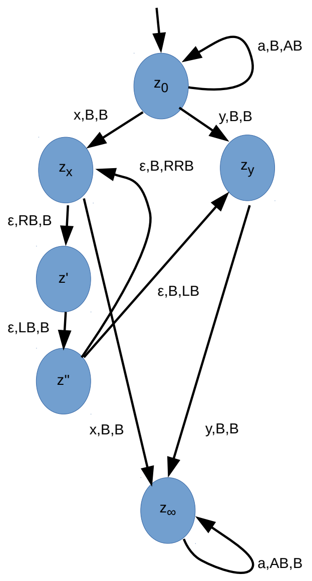

Figure 7: Example pushdown automaton

constraints. In any case we have that the label word before the x's is longer than the label word after them, which is not a consequence of the constraint. This is a contradiction.

However, we can use L to show the regularity of the language $L_{x,t} = \{u \mid TC \vdash ux \geq t\}$ where the second variable t including its label word is fixed and x is fixed, but not its label word.[3] For that, we build the above mentioned pushdown automaton $\mathcal{A}$ similar to the construction for $\mathcal{A}_0$ above. We can assume w.l.o.g. (possibly by introducing new states), that $\mathcal{A}$ has only transitions of the form $x \to y, x \overset{pop(a)}{\longrightarrow} y$, or $x \overset{push(a)}{\longrightarrow} y$, with x, y states that belong to variables. We then define the set

$$Q = \{(x, y) | x \to^* y\}$$

and enumerate it using dynamic programming and the rules in Figure 8. Now we obtain for $L_{x,t}$ the representation in Figure 9. From this we can read off a finite automaton $\mathcal{B}$ for $L_{x,t}$ directly: the states are the states of $\mathcal{A}$, for each pair $(x, y) \in Q$ we introduce an ϵ-move from state x to state y, and the $\mathcal{A}$-transitions $x' \overset{pop(a)}{\longrightarrow} x''$ are the nontrivial moves that consume the letter a. □

We then also have $L_{ux,y} = \{w \mid wu \in L_{x,y}\}, L_{x,vy}$ and $L_{ux,vy}$ regular. The disjoint union over the sets $L_{x,vy}$ for all x equals $L^{\geq}_{z}$, for the expression $z = vy$.

From now on we omit the brackets for trees with prefixed labels and write lx instead of $l(x)$.

Example 4. *Let x, y, z be tree variables ranging over trees with labels in $\{l, r\}$, as before. Let*

[3] In the list case, where we have only one label, this is a direct consequence of Parikh's theorem.

$$\frac{x \to y}{(x,y) : Q}$$

$$\frac{x \longrightarrow y \qquad (y,z) : Q \qquad z \longrightarrow w}{(x,w) : Q}$$

$$\frac{x \overset{push(a)}{\longrightarrow} y \qquad (y,z) : Q \qquad z \overset{pop(a)}{\longrightarrow} w}{(x,w) : Q}$$

Figure 8: Rules for Q

$$\frac{(x,y) : Q}{\epsilon : L_{x,y}}$$

$$\frac{u : L_{x''',t} \qquad (x'',x''') : Q \qquad x' \overset{pop(a)}{\longrightarrow} x'' \qquad (x,x') : Q}{ua : L_{x,t}}$$

Figure 9: Rules for $L_{x,t}$

the constraints be

$$lx \geq x \qquad x \geq rz \qquad lrz \geq lly \qquad ly \geq y$$

then $L_{x,y} = L_{x,ly} = l^+$ and $L_{lx,y} = l^*$. The language $L^{\leq}_{lrz} = \{lr_z, ll_y, l_y, \epsilon_y\}$, where the subscript x at label word w means that $w \in L_{x,lrz}$.

Recall that tree constraints systems without arithmetic constraints are always trivially satisfiable by setting all tree entries to zero. Analogously, all nodes that have bounds only in one direction (i.e. are only implied to be greater than a set of arithmetic variables a_i or only less) can be set to zero or infinity. The only interesting case appears when we have subtrees x whose root $\Diamond(x)$ lies between two arithmetic variables a and b. The set $L^{\geq}_{a} \cap L^{\leq}_{b} := \{x \mid a \leq \Diamond(x) \leq b\}$ can be computed using the languages $L^{\geq}_{x}$ and $L^{\leq}_{y}$ for certain subtrees x, y. These trees are defined as the subtree starting at the point where the arithmetic variables a, b are located. For instance, if $a = \Diamond(lrrz)$, then the subtree x is $lrrz$. Thus we can write $L^{\geq}_{a} \cap L^{\leq}_{b} = \{x \mid TC \vdash y_a \leq x \leq y_b\}$, where y_a (resp. y_b) is the tree with root $\Diamond(y_a) = a$ (resp. b).

Example 5. *Consider the constraints*

$$\Diamond(x) = 1 \qquad lrx \geq x \qquad lx \geq x \qquad mx \geq x \qquad x \geq rlx \qquad x \geq mlx$$

The language of trees with root being either greater-than-or-equal, equal, or less-than-or-equal to *the root of x are obtained by iteratively applying the constraints and transitivity.*

$$L^{\geq}_{\Diamond(x)} = (lr \mid l \mid m)^*_x, \quad L^{=}_{\Diamond(x)} = L^{\leq}_{\Diamond(x)} \cap L^{\geq}_{\Diamond(x)} = m(lr)^* l_x = ml(rl)^*_x, \quad L^{\leq}_{\Diamond(x)} = (ml \mid rl)^*_x$$

2.3 Normal Form for Tree Constraints

We now bring the constraints into a normal form to start our procedure. Constraints in this normal form all have a variable with label word of length n on the left hand side, and all label words on the right are at most of length n. The variables with label word of length exactly n can be represented as a directed acyclic graph with an edge between x and y if and only if $TC \vdash x \leq y$. Further, there are no arithmetic constraints below level $n-1$ (i.e. for trees with label word of length more than $n-1$).

To obtain constraints in this normal form, we examine the form of each constraint. If there is only one label word of maximal length, we take this word and isolate it on the left side of the inequality. If there is more than one such word, we write k-times repeated addition of the same summand s as $k \cdot s$. If there is then only one summand s of maximal length, we bring it on the left hand side and divide both sides by k. Otherwise, we build l constraints by bringing the l'th of the longest summands on the left (possibly again by dividing by a positive integer). Then, we apply the rules in Figure 5 to all the obtained constraints until all label words on the left have the same length. The result is an equivalent constraint system (i.e. a system with exactly the same solutions) consisting of unfolded tree constraints and a new, bigger set of arithmetic constraints.

Depending on the position of this longest label word and according to the unilateral syntax, we have — after bringing the longest label word on the left side — three kinds of constraints:

- *lower bounds* are of the form $wx \geq w_1 y_1 + \cdots + w_i y_i$, with all w_i shorter than w.
- *upper bounds* have the shape $wx \leq w'y - w_1 y_1 - \cdots - w_i y_i$, with all w_i, w' shorter than w.
- *undirected constraints* have two label words of the same length on both sides, as for instance $x \geq y + z, x \leq y - z$.

Example 6. *The constraint $x \geq lx + y$ is an upper bound and is rewritten to $lx \leq x - y$.*

The last set (the undirected constraints) can be transformed into a directed acyclic graph by removing cycles as follows: If we can derive $x \geq y_1 \geq y_2 \geq \cdots \geq x + R$ by just using transitivity (not through label application, which would require again a regular language), then we conclude that $x = y_i$ forall i and that R is identical to the tree consisting only of zeros. In this graph, we have now encoded upper and lower bounds simultaneously.

Example 7. *Let the constraints be $x \geq y + z$, $z \geq t$, $y \geq t$. They correspond to the graph with an edge from x to y and to z and from y and z to t. We then add the four constraints $t \leq z$, $t \leq y$, $y \leq x - z$, $z \leq x - y$ to our system. We must traverse this graph in two directions (i.e. we need both kinds of bounds that are implied by it) to obtain an order in which we treat the nodes. The need for this will be explained in Example 8.*

2.4 Idea and Examples

Let us briefly explain the intuition behind our procedure before covering the technical details. The idea is that we label all nodes in the trees which are in a set $L^{\geq}_{a} \cap L^{\leq}_{b}$ for arithmetic variables a and b with sets of intervals to which the number in the node has to belong to. These intervals are derived from the constraints. Nodes with the same set of intervals are defined to be in the same *class*. It can then be shown that there are only finitely many different classes for all constraint systems that contain the satisfiable ones.

Last, we translate the statement that all these intervals are nonempty into a finite set of linear inequalities between the arithmetic variables. This linear program is equisatisfiable to the constraints (i.e. if the intervals are nonempty, then there exists a solution with the valuation of each $\Diamond(x)$ in the interval assigned to $\Diamond(x)$). If they are satisfiable, we thus get the answer in terms of a satisfiable linear program, and, in addition to that, an assignment of a class to each of the nodes that can be seen as a certificate for satisfiability.

Combining this with Lemma 1, we have a decision procedure that either returns an unsatisfiable linear program implied by the constraints or a schematic notation for the intervals that contain their solutions from which it is directly possible to compute a solution.

Example 8. *Consider the (list-) constraints*

$$\Diamond(x) = \Diamond(y) = 1 \qquad x \geq y \qquad lx \leq x \qquad ly \geq y$$

They are equivalent to the system $\Diamond(x) = \Diamond(y) = 1$ in conjunction with $C = \{c_1, c_2, c_3, c_4\}$, where

$$c_1 \coloneqq ly \leq lx \qquad c_2 \coloneqq lx \geq ly \qquad c_3 \coloneqq lx \leq x \qquad c_4 \coloneqq ly \geq y$$

which is in normal form (with constraint c_1 being redundant in this case). The constraint c_3 delivers the interval $[0, 1]$ for $\Diamond(lx)$ (resp. c_4 delivers $[1, \infty]$ for $\Diamond(ly)$). Thus c_1 gives us the interval $[1, 1]$ for $\Diamond(ly)$ and c_2 gives the same interval for $\Diamond(lx)$. By duplicating the constraint, we ensure that we treat the nodes in subsequent levels of the DAG (as constructed above) correctly. Imagine we had only c_1 without c_2, then we would miss the bound on $\Diamond(ly)$. In the next steps, we derive no new bounds any more.

The system in Example 8 is satisfiable by the trees consisting of only 1s. If we modify it slightly, it becomes unsatisfiable:

Example 9. *The constraints $x \geq y, \Diamond(y) \geq 1, lx + lx \leq x, ly \geq y + y$ are equivalent to $\Diamond(x) \geq \Diamond(y) \geq 1$ and $D = \{d_1, d_2, d_3, d_4\}$, where*

$$d_1 \coloneqq ly \leq lx \qquad d_2 \coloneqq lx \geq ly \qquad d_3 \coloneqq lx \leq 0.5x \qquad d_4 \coloneqq ly \geq 2y$$

We have the intervals $[0, 0.5\Diamond(x)]$ and $[2\Diamond(y), \infty]$ for $\Diamond(lx)$ by d_3 and d_2, and $[2\Diamond(y), \infty]$ and $[0, 0.5\Diamond(x)]$ for $\Diamond(ly)$ by d_4 and d_1. In the next steps, the factors will be 0.25 and 4, etc. The list x is exponentially decreasing, whereas y grows exponentially. So if the roots of x and y are neither zero nor infinity, then no matter which number they are, x will at some point be less than y. Here we see that there is a contradiction, but we can not say after how many iterations we will find it. There the other part of our algorithm, namely Lemma 1, applies.

2.5 Satisfiability is Decidable

Before we prove the central facts (Theorem 3 and Theorem 4) of this paper, we need some word-combinatorial preliminaries. We say that two label words are *dependent* if one is a suffix of the other. The next two lemmas are well known and can be found for instance in [9].

Lemma 2. *Let $u, s, t \in \Sigma^+$ such that $tu = st$. There then exist $q, r \in \Sigma^*$ and $i \in \mathbb{N}$ such that $s = qr$, $u = rq$, $t = q(rq)^i$.*

	$a_1a_2 \dots a_n$	$a_3 \dots a_n$
a_1a_2	$a_3 \dots a_n a_1 a_2$	$a_3 \dots a_n$

Figure 10: Commuting words a_1a_2 and $a_3 \dots a_n$

Lemma 3. *If for word x, y, z holds $x^n y^m = z^k$, with $n, m, k \geq 2$, then exists t such that $x, y, z \in t^*$.*

Lemma 4. *Let c be a unilateral tree constraint. If it is of the form*

$$a_1a_2 \dots a_n x \geq c_1 \cdot a_2 \dots a_n x + \dots + c_n \cdot a_n x + c_{n+1} \cdot x \tag{1}$$

with all $a_i \neq \epsilon, c_j \in \mathbb{N}_0$ and label words and x a tree, then it can be transformed (by application of labels from the left) into a constraint with all $a_k \in p^+$ for a suitable word p and all other summands independent.

Proof. We assume that there is a label word $t \in \Sigma^*$ which we can apply from the left such that all summands stay dependent of $a_1a_2 \dots a_n x$. (If such a t does not exist, then all summands are already independent.) This is,

$$ta_1a_2 \dots a_n \text{ has the suffixes } ta_2 \dots a_n, ta_3 \dots a_n, \dots ta_n, t. \tag{2}$$

We apply Lemma 2 to $ta_1a_2 \dots a_n = st$ with $u = a_1 \dots a_n$ and obtain $r, q \in \Sigma^*$ such that $a_1 \dots a_n = rq, t = q(rq)^i = q(a_1 \dots a_n)^i$. Thus $ta_2 \dots a_n = q(a_1 \dots a_n)^i a_2 \dots a_n$. According to (2), $ta_2 \dots a_n$ is a suffix of $ta_1a_2 \dots a_n = q(a_1 \dots a_n)^{i+1}$. This means that a_1 and $a_2 \dots a_n$ commute. Hence there is p_1 such that both are in p_1^+.

Similarly, $ta_1a_2 \dots a_n$ has the suffix $ta_3 \dots a_n = q(a_1 \dots a_n)^i a_3 \dots a_n$, and thus a_1a_2 and $a_3 \dots a_n$ commute (see Figure 10, where words of the same length are written in boxes). We can thus conclude that there is p_2 with a_1a_2 and $a_3 \dots a_n$ are in p_2^+. We proceed the same way until we obtain in the last step that $a_1 \dots a_{n-1}$ and a_n commute. We now write $a_1 = p_1^{i_1}, a_2 \dots a_n = p_1^{j_1}$ and $a_1a_2 = p_2^{i_1}, a_3 \dots a_n = p_2^{j_2}$, etc. Application of Lemma 3 allows us to conclude from $(a_1 \dots a_n)^2 = p_1^{i_1+j_1} p_2^{i_2+j_2} = p_3^{2(i_3+j_3)}$ that $p_1, p_2, p_3 \in p^*$ for a certain p. Thus all p_i are in p^+ and for all i, we have $a_i \in p^+$. □

Theorem 3. *Satisfiability of linear tree constraints is semi-decidable.*

Proof. Assume that all constraints are in the normal form described above. Introduce an arithmetic variable for each node above level n. Recall that there are no arithmetic constraints below level $n-1$ and all left hand sides of the constraints have label word of length exactly n. Now calculate the sets $L_a^{\geq} \cap L_b^{\leq}$ for all pairs of arithmetic variables a, b. One may assume w.l.o.g. that all a, b are neither zero nor infinity: Otherwise, let A, B be disjoint subsets of the set of arithmetic variables and test all combinations of additional constraints $A \ni a_i = 0$ and $B \ni a_i = \infty$. If one of them is satisfiable, we return this as the result.

Our procedure starts with step 1 at level n and assigns a set of intervals to each node. For the lower bounds, which have the form $wx \geq w_1y_1 + \dots + w_my_m$, with all w_i shorter than w, we add the interval $[\sum_i \Diamond(w_iy_i), \infty]$. For the upper bounds, that have the shape $wx \leq w'y - w_1y_1 - \dots - w_my_m$, with all w_i and w' shorter than w, we add the interval $[0, \Diamond(w'y) - \sum_i \Diamond(w_iy_i)]$.

For the undirected constraints, we observe that the membership of all nodes in $L_a^{\geq} \cap L_b^{\leq}$ ensures that we have already an interval for the starting nodes of the DAG constructed above.

We traverse it in both directions and add for constraints $x \geq y_1 + \cdots + y_m$ the new set of intervals $\big\{[\sum_i a_i, \infty] \bigm| [a_i, b_i]$ is an interval for $\Diamond(y_i)\big\}$; respectively for $x \leq y - z_1 - \cdots - z_m$ the new intervals $\big\{[0, b - \sum_i c_i] \bigm| [a, b]$ is an interval for $\Diamond(y)$ and $[c_i, d_i]$ is an interval for $\Diamond(z_i)\big\}$.

Further, we set all nodes that have bounds in only one direction to $[0, 0]$ or $[\infty, \infty]$. We denote the set of intervals for node $\Diamond(x)$ with $I(x)$, and I_n is the set of all $I(x)$ obtained until step n.

The unilateral constraint syntax allows us to define a meaningful addition and subtraction on interval sets that formalises how we compute new interval sets.

$$I(x) + I(y) = \big\{[a + c, \infty] \mid [a, b] \in I(x), [c, d] \in I(y)\big\}$$
$$I(x) - I(y) = \big\{\,[0,\ b - c] \mid [a, b] \in I(x), [c, d] \in I(y)\big\}$$

Observation 1. *The order of evaluation does not play any role for sums of interval sets, i.e.* $I(x) - I(y) - I(z) = I(x) - \big(I(y) + I(z)\big)$.

To prove this, let w.l.o.g. be $I(x) = [a, b]$, $I(y) = [c, d]$ and $I(z) = [e, f]$. Then

$$\begin{aligned}[a, b] - [c, d] - [e, f] &= [0, b - c - e] = [0, b - (c + e)] \\ &= [a, b] - [c + e, \infty] = [a, b] - ([c, d] + [e, f]).\end{aligned}$$

In step $n + 1$, we apply the rule (LabelSum) from Figure 5 to the constraints to increase the length of the label word of the left sides by 1. Then, for the lower bounds we no longer necessarily have arithmetic variables as roots of the trees on the right, but also nodes equipped with intervals. Thus, we proceed in a similar way as for the undirected constraints on level n, namely add the intervals that can be derived from the variables on the right. We do the same for the upper bounds and the undirected constraints on level $n + 1$. More precisely, for the lower bounds, we set $I(x) = I(y_1) + \cdots + I(y_m)$, and for the upper bounds $I(x) = I(y) - I(z_1) \cdots - I(z_m)$.

We claim that after a finite amount of steps, no new intervals are derived any more: If we see the set of intervals that belong to a node as its *class*, then there are only finitely many different classes. The reason is that if the intersection of one of the interval sets would be constantly shrinking, we would infinitely often add a nonzero number to the lower bound or subtract a nonzero number from the upper bound or divide the upper bound by a positive integer (by the assumption that all arithmetic variables are neither zero nor infinity). Since all considered nodes are bounded from above and below, we would at some point obtain a contradiction (see Example 9).

It is thus necessary to give a criterion that ensures that we have found all classes already and must not search for new classes anymore. Having found all intervals, the condition that the intersection of all intervals that belong to the same node is nonempty delivers an equisatisfiable linear program.

We now define S as the least common multiple of all differences of label word lengths that appear in the constraint system. For instance, S for the single constraint $lrrlx \geq x + lx$ is $12 = \mathrm{lcm}(4, 3)$. Note that in the list case, S is a bound on the period length of the solution lists (cf. [3]). The criterion looks as follows: Let L be the set defined by

$$L = \cup_{a,b}(L_a^{\geq} \cap L_b^{\leq}),$$

which describes the nodes with bounds in two directions. If in S iterations no new interval sets for the nodes in L are derived any more (i.e. for each node x on a certain level and word p with $|p| = S$, the intersection of all intervals that belong to px is equal to the intersection of the intervals for x), then we have found all of them.

There are two things to show, namely that the premise of this criterion implies $I = I_n$ for a $n \in \mathbb{N}$ and that this premise will finally hold.

Claim 1 (Part 1). *If there is a $n \in \mathbb{N}$ such that for all $x \in L$ on level $n, \dots, n+S$ and for all label words p with $|p| = S$, the set $I(px)$ is equal to $I(x)$, then $I = \cup_{j \in \mathbb{N}} I_j = I_{n+S}$.*

Claim 2 (Part 2). *There is a $n \in \mathbb{N}$ such that for all $x \in L$ on level $n, \dots, n+S$ and for all label words p with $|p| = S$, the set $I(px)$ is equal to $I(x)$.*

To prove the first, we show that for all $k = 0, \dots, S$ and for all $l \in \mathbb{N}$, we have $I_{n+k+l \cdot S} = I_{n+k}$. We consider three cases:

1. If we have a lower bound constraint $px \geq \sum_i y_i$, we know that for all label words p, q with $px, qpx \in L$ and $|p|, |q| = S$, this implies $qpx \geq \sum_i qy_i$. The lower bounds of the intervals for qy_i are not stronger than those for y_i. This follows from the assumption if $qy_i \in L$. It is also true if $qy_i \notin L$, because then the interval for qy_i must be $[0, 0]$ (since it is less or equal to qpx, which is at most b and so it can not be $[\infty, \infty]$) and thus it delivers no new bounds at all. We mark this property by $(\star)$.

2. Similarly, in case of upper bounds $px \leq y - \sum_i z_i$, the upper bounds for qy and the lower bounds of the intervals for qz_i are not stronger than those for y and z_i. Again, if $qy, qz_i \in L$, this is a consequence of the assumption, and if $qy \in L$ and $qz_i \notin L$, then qz_i has interval $[0, 0]$. Last, if $qy \notin L$ then qy has interval $[\infty, \infty]$ and delivers no new bounds. This property is called $(\star\star)$.

In these two cases, the right hand side in $(\star)$ and $(\star\star)$ is on level less or equal to the level on the left. We may assume that all $py_i, qy, qz_i \in L$. For all label words q, p with $qpx \in L$ and $|q|, |p| = S$, we have $I(qy_i) = I(y_i)$, $I(qy) = I(y)$, $I(qz_i) = I(z_i)$ and the set of intervals for qpx, which is the sum of the intervals for the qy (resp. the difference between the intervals of qy and the qz_i) is (after intersection) not smaller than the set of intervals for qpx (according to $(\star)$ and $(\star\star)$), and also not smaller than the interval for x. More precisely, we have

$$I(qpx) = I(qy_i) + \dots + I(qy_m) = I(y_i) + \dots + I(y_m) = I(px) = I(x), \text{ or resp.}$$
$$I(qpx) = I(qy) - I(qz_i) - \dots - I(qz_m) = I(y) - I(z_1) - \dots - I(z_m) = I(px) = I(x).$$

This means that if we jump S levels down in the tree and nothing changes regarding the intervals, then the next iteration will never again obtain a new interval.

3. The last case occurs if we have an undirected constraint $x \geq y+z$. This implies $px \geq py+pz$, with $I(px) = I(x)$, $I(py) = I(y)$, $I(pz) = I(z)$, and for the new interval obtained from the undirected constraint $I_{new}(px) = I(py) + I(pz) = I(y) + I(z) = I(x)$, etc. If there are no changes in the intervals for p, q of length S, then there are no changes at all and $I = I_{n+S}$.

To prove the second part of the claim, we assume that for all levels there is a p of length S and x on that level (optionally plus a number between 1 and S) such that $I(px) \neq I(x)$. In other words, we derive at least one new interval on each Sth level.

All constraints on the variables in L (except a subset of the undirected constraints where all label words have the same length) imply constraints of the form $qy \geq z + R$ (resp. $qy \leq z - R$) with $|q| = S$. For the lower bounds, we have to consider all possibilities for the choice of z in the following, whereas for the upper bounds there is only one positive summand and thus z is unique.

We can assume that for all x holds $I(px) \subset I(x)$ because of the choice of S: if this was not the case and pqy had strictly weaker lower bounds (resp. strictly weaker upper bounds) than pz, the interval $I(pqy)$ would contain more than $I(pz)$ and thus qy could not have z as a bound.

Thus we have $I(pqy) = I(pz) + I(R)$ (resp. $I(pqy) = I(pz) - I(R)$). We now assume w.l.o.g. that y plays the role of the x above and that $z = y$ holds[4]. So we have a constraint $px \geq x + R$ or $px \leq x - R$ which is derivable just by unfolding using the (LabelSum) rule in Figure 5. We just treat the first since both are similar.

According to Lemma 4, either some label words of the summands in $R = r_1 + \cdots + r_m$ and p are powers of the same path t, or in the next step all summands in $qR \coloneqq qr_1 + \cdots + qr_m$ are independent of qpx for all q of length S. If the second happens, if no tree $t^i x$ is reachable[5] from R, then either this constraint delivers no new bounds below level $n + S$, or R must contain a tree z that is at least constant when seen as a list $(p'^i z)_i$ along a path $p' \in t^*$ of length S. This implies $p = p'$, which again implies $ppx = t^j px = t^{2j} x \geq t^j x + t^{kj} z + R'$, and that means $p^l x \geq p^{l-1} x + p^{l+k-2} z$ holds — just like in the first case. Overall, we have that either the constraint does not deliver an infinite amount of new bounds or has the form of a strictly increasing list along the path $(\Diamond(p^i))_i$. In both cases, this is a contradiction, since only finitely many $p^i x$ can be in L, thus $I(p^i x)$ is at some point equal to $[\infty, \infty]$ and then stops changing.

Claim 1 and Claim 2 ensure that in a set of cases including the satisfiable ones, we will only derive finitely many different intervals. Thus the problem to decide whether the values of the arithmetic variables can be chosen such that these intervals are all nonempty can be solved by linear programming. If and only if they can be chosen this way, the constraints are satisfiable. $\square$

Example 10. *Consider the constraint system*

$$llx \geq rx + mx \qquad\qquad \Diamond(rx) = 1$$

There are no arithmetic constraints below level 1*, and the system is already in normal form. All summands on the right hand side of the tree constraint are independent of the variable on the left hand side. We derive the interval* $[1, \infty]$ *for* llx *and continue with unfolding the constraint to obtain*

$$rllx \geq rrx + rmx \qquad lllx \geq lrx + lmx \qquad mllx \geq mrx + mmx$$

Because of the independency, we obtain no new intervals at all, since none of the trees on the right sides are in L*.*

Example 11. *Let the constraints be*

$$\Diamond(y) = 1 \qquad y \geq ly \qquad y \geq ry \qquad lx \geq x + y \qquad rx \geq x + y \qquad x \geq lrx$$

Then all nodes in $(l|r)^+ y$ *are assigned the intervals* $[0, 0]$*. Similarly,* $l^+ x$ *and* $r^+ x$ *and all other nodes are set to* $[\infty, \infty]$*, except those nodes that have bounds in two directions (i.e. are in* L*). The only nodes in* L *are the roots of* $lr^* x$ *and* $\Diamond(y)$*. Thus we only need to compute intervals for* $lr^* x$*. The root* $\Diamond(x)$ *gets the interval* $[1, 1]$*. On level two, there is no node in* L*. Then, on level three,* $\Diamond(lrx)$ *is labeled with the intervals* $[1, \infty]$ *and* $[0, 1]$*. Thus their intersection is equal to*

[4] If this is not the case for the initial y, the next candidate for x is z.

[5] No lower bound constraint on any summand $t'x$ in R with $t' \in t^*$ on the right exists — which is decidable according to Theorem 2.

$[1,1]$. *The same happens on level* $4, 6, 8$, *etc. Indeed, we can easily check that another solution than one with* $\forall w \in (lr)^*.\Diamond(wx) = 1$ *is not possible.*

Combined with the semi-decidable unsatisfiability, we can decide UTC.

Theorem 4. *The unilateral tree constraint problem is decidable.*

Proof. For a given constraint system, we run the two semi-decision procedures for satisfiability and unsatisfiability in parallel. As we proved, one of them will terminate and this yields our result. □

2.6 Complexity

If the constraints are unsatisfiable, we cannot determine the complexity at all, because this contradiction may be located arbitrarily deep in the tree, depending on the initial values. For instance, if we have tree constraints

$$y \geq x \qquad lrx \geq x \qquad y \geq 2lry \qquad \Diamond(x) = 1 \qquad \Diamond(y) \geq 1024 = 2^{10}$$

then we will need 10 steps to find a contradiction, if $\Diamond(y) = 1024$. However, for $\Diamond(y) \geq 2^{100}$ we will need 100 steps. The complexity for the algorithm in the satisfiable case is at least exponential according to the number of linear programs used when assuming that all initial values are nonzero.

3 Relation to RAJA programs

We briefly sketch how we can read off resource bounds from constraint solutions. The amortised analysis approach associates a potential (i.e. resources) to *each reference* to an object. For each individual access path to a runtime object, we find the associated resources as a number within a tree. Since we may have cyclic object references, these trees must be assumed to be infinite; as cyclic references allow for access paths of arbitrary lengths. More precisely, the number in the node of tree t encodes the required resources for an object x itself and the subtrees of t belong to the attributes of x, which are again objects. The amortized resource analysis for object-oriented programs produces UTC as a result; the solutions then compute the actual numbers. Note that a solution gives us more information than just some constant linear factors, since the information whether or not a factor is non-zero determines asymptotic complexity.

Consider the RAJA program shown in Figure 11, which creates a degenerated binary tree out of a tree by replacing the left subtree by the sum of all elements of the right subtree. The class `Tree` has subclasses `NonemptyTree` and `Empty`.

The analysis produces 1398 constraints for this program (including all class definitions that are not shown in Figure 11). One of these constraints is

$$rx \geq x + y$$

where the inhomogeneity belongs to a constant tree and thus causes linear growth for the tree x, when going along the rightmost path. Intuitively, the reason for the nonlinear bound is that we create a new `NonemptyTree` object `l` in each recursive step of the function `sums`, which each time requires a constant amount of resources. So each node must have these resources at hand to spend them for the new allocation. In addition, the nodes of the right subtrees also have to have enough resources for calling the function `toDegTree` recursively. The recursion here

```
class NonemptyTree extends Tree{
  int elem; Tree left; Tree right;

  Tree sums(){
    let l = new NonemptyTree in
    let _ = l.elem <- this.elem + this.left.sums().elem
      + this.right.sums().elem in return l;}

  Tree toDegTree(){
    let _ = this.left  <- this.right.sums() in
    let _ = this.right <- this.right.toDegTree() in return this;}
}

class Main{
  Tree main(NonemptyTree t'){return t'.toDegTree();}
}
```

Figure 11: RAJA code with nonlinear constraints

leads to the condition that the right subtree of x must contain the same or more resources than x itself. This program thus needs super-linear (more precisely at least quadratic) resources, because we obtain the overall resource consumption by adding the required resources for each single node. Thus this program that cannot be analyzed with the results in previous work, but with the decision procedure in this paper, programs of this kind become analyzable.

4 Conclusion

We have proven that linear constraints over infinite trees, as generated by an automatic resource type inference for the language RAJA, are decidable. Our approach uses pushdown automata, nondeterministic finite automata and combinatorics of words to generalise the list constraint theory to trees. For the latter, satisfiability was previously proven decidable in polynomial time. In contrast to that, our algorithm for trees needs exponential time, because the number of the linear programs that we reduce the problem to is exponential in the size of the input.

This decidability result enables us to analyse arbitrary RAJA programs with respect to their resource consumption. We can read off upper bounds on the memory usage from the solutions of the constraints. Until now this was only possible for a subset of programs that need linear resources.

For example, sorting a list using merge-sort is one of the examples provided with the prototype implementation of RAJA[6]. The analysis of this example previously only succeeded thanks to using static garbage collection, namely *free* expressions. While there is research on static garbage collection [31, 2], there are still many open questions about the realization of a static garbage collector in Java. The results of this paper now enables a successful resource analysis of the merge-sort example even in the absence of static garbage collection (i.e. no *free* expressions in the code). Thus nonlinear bounds make our analysis independent of this construction and thus closer to real Java. Note that especially for bigger programs or programs with auxiliary

[6]raja.tcs.ifi.lmu.de

functions or nested data structures, the constraint generation is very involved and often leads to superlinear resource annotations.

Future work includes a procedure for finding closed formulas for the optimal, namely minimal solutions, similar to the list case in [3]. In order to develope an implementation based on the existing RAJA tool, it would be useful to optimise our algorithms and then determine the exact complexity of our decision procedure.

Acknowledgements

The technical details for the proof of the main theorem were concluded by Sabine Bauer and Martin Hofmann in 2017. Therefore we naturally include the late Martin Hofmann as an author for this paper, due to his insightful and essential contribution. We also thank Helmut Seidl for his helpful advice to conclude this paper.

References

[1] Elvira Albert, Puri Arenas, Jesús Correas, Samir Genaim, Miguel Gómez-Zamalloa, Germán Puebla, and Guillermo Román-Díez. Object-sensitive cost analysis for concurrent objects. *Software Testing, Verification and Reliability*, 25(3):218–271, 2015.

[2] Elvira Albert, Samir Genaim, and Miguel Gómez-Zamalloa. Parametric Inference of Memory Requirements for Garbage Collected Languages. In Jan Vitek and Doug Lea, editors, *Proceedings of the 2010 International Symposium on Memory Management (ISMM'10)*, pages 121–130. ACM, 2010.

[3] Sabine Bauer and Martin Hofmann. Decidable linear list constraints. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 181–199. EasyChair, 2017.

[4] Jason P. Bell and Stefan Gerhold. On the positivity set of a linear recurrence sequence. *Israel Journal of Mathematics*, 157:333–345, 2007.

[5] Achim Blumensath and Erich Grädel. Automatic Structures. In *Proceedings of the Fifteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2000)*, pages 51–62. IEEE Computer Society Press, June 2000.

[6] Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. Compositional Certified Resource Bounds. pages 467–478. Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2015.

[7] Quentin Carbonneaux, Jan Hoffmann, Zhong Shao, and Tahina Ramananandro. End-to-end verification of stack-space bounds for C programs. pages 270–281. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2014.

[8] Vijay Chandru and John Hooker. *Optimization Methods for Logical Inference.* Wiley, 1999.

[9] Christian Choffrut and Juhani Karhumäki. *Combinatorics of Words.* Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[10] Stefan Dantchev and Frank D Valencia. On Infinite CSPs. *Modelling and Reformulating Constraint Satisfaction Problems*, 2009.

[11] Florian Frohn and Juergen Giesl. Analyzing Runtime Complexity via Innermost Runtime Complexity. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 249–268. EasyChair, 2017.

[12] Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumäki. Skolem's problem - on the border between decidability and undecidability. *TUCS Technical Reports*, (683), 2005.

[13] Reinhold Heckmann and Christian Ferdinand. Worst-Case Execution Time Prediction by Static Program Analysis. AbsInt Angewandte Informatik GmbH. http://www.absint.com.

[14] Jan Hoffmann. *Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis*. PhD thesis, University of Munich, 2011.

[15] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate Amortized Resource Analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14:1–14:62, November 2012.

[16] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. *Resource Aware ML*, pages 781–786. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[17] Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards automatic resource bound analysis for OCaml. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 359–373, 2017.

[18] Jan Hoffmann and Martin Hofmann. Amortized Resource Analysis with Polymorphic Recursion and Partial Big-Step Operational Semantics. In *APLAS*, pages 172–187, 2010.

[19] Jan Hoffmann and Martin Hofmann. *Amortized Resource Analysis with Polynomial Potential*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[20] Jan Hoffmann and Zhong Shao. *Type-Based Amortized Resource Analysis with Integers and Arrays*, pages 152–168. Springer International Publishing, Cham, 2014.

[21] Jan Hoffmann and Zhong Shao. *Automatic Static Cost Analysis for Parallel Programs*, pages 132–157. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[22] Martin Hofmann and Steffen Jost. Static Prediction of Heap Space Usage for First-order Functional Programs. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '03, pages 185–197, New York, NY, USA, 2003. ACM.

[23] Martin Hofmann and Steffen Jost. Type-Based Amortised Heap-Space Analysis. In Peter Sestoft, editor, *Programming Languages and Systems*, pages 22–37. Springer Berlin Heidelberg, 2006.

[24] Martin Hofmann and Georg Moser. Multivariate Amortised Resource Analysis for Term Rewrite Systems. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 241–256, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[25] Martin Hofmann and Dulma Rodriguez. Efficient Type-Checking for Amortised Heap-Space Analysis. In *CSL: 18th EACSL Annual Conference on Computer Science Logic*. LNCS, Springer-Verlag, 2009.

[26] Martin Hofmann and Dulma Rodriguez. Linear Constraints over Infinite Trees. In *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'12, pages 343–358, Berlin, Heidelberg, 2012. Springer-Verlag.

[27] Martin Hofmann and Dulma Rodriguez. Automatic Type Inference for Amortised Heap-Space Analysis. In *ESOP: 22nd European Symposium on Programming*, 2013.

[28] Joël Ouaknine and James Worrell. Decision problems for linear recurrence sequences. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *Reachability Problems: 6th International Workshop, RP 2012, Bordeaux, France, September 17-19, 2012. Proceedings*. Springer Berlin Heidelberg, 2012.

[29] Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. *CoRR*, abs/1307.2779, 2013.

[30] Dulma Rodriguez. *Amortized Analysis for Object Oriented Programs*. PhD thesis, University of Munich, 2012.

[31] Leena Unnikrishnan and Scott D. Stoller. Parametric heap usage analysis for functional programs. In *Proceedings of the 8th International Symposium on Memory Management, ISMM*, pages 139–148, 2009.