Satisfiability of Compositional Separation Logic with Tree Predicates and Data Constraints

Zhaowei Xu[1,2], Taolue Chen[3,4], and Zhilin Wu[1](✉)

[1] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, Beijing, China
wuzl@ios.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Department of Computer Science, Middlesex University, London, UK
[4] State Key Laboratory of Novel Software Technology,
Nanjing University, Nanjing, China

Abstract. In this paper, we propose compositional separation logic with tree predicates (CSLTP), where properties such as sortedness and height-balancedness of complex data structures (for instance, AVL trees and red-black trees) can be fully specified. We show that the satisfiability problem of CSLTP is decidable. The main technical ingredient of the decision procedure is to compute the least fixed point of a class of inductively defined predicates that are non-linear and involve dense-order and difference-bound constraints, which are of independent interests.

1 Introduction

Program verification requires reasoning about complex, size-unbounded data structures that may carry data ranging over an infinite domain. Examples include multi-linked lists, nested lists, trees, etc. Programs manipulating these data structures may modify their shape as well as the data attached to their elements. *Separation Logic* (SL) is a well-established approach for deductive verification of programs that manipulate dynamic data structures [22,30]. Typically, SL is defined in combination with *inductive definitions* (SLID in short), which supports user-defined specifications of the data structures manipulated by a program.

Satisfiability is arguably one of the most fundamental questions for logic, and has certainly been a main focus in the study of SL. The satisfiability of SLID with data constraints is evidently undecidable in their most general forms. However, it is important—both in theory and practice—to identify subclasses which are sufficiently expressive while still being decidable. Within this context, our previous work [14] gave complete decision procedures for both the satisfiability and the entailment problem of *linearly* compositional SLID. This fragment is able

Taolue Chen is supported by UK EPSRC grant (EP/P00430X/1), European CHIST-ERA project SUCCESS, NSFC grant (61662035). He is also affiliated with Centre for Research and Innovation in Software Engineering, Southwest University. Zhilin Wu is supported by the NSFC grants (61572478, 61472474, 61100062, and 61272135).

L. de Moura (Ed.): CADE 2017, LNAI 10395, pp. 509–527, 2017.
DOI: 10.1007/978-3-319-63046-5_31

to specify typical shape properties and data/size constraints of data structures, but is restricted to linear ones such as singly and doubly linked lists.

An obvious question left over is to handle non-linear structures such as trees. Notice that most tree-shaped data structures in programming require data/size constraints of one or another. They together, however, impose great challenges. For satisfiability, the main difficulty roots at the computation of the least fixed point of the inductively defined predicates derived from SL formulae. These predicates are *non-linear*, meaning that the defined predicate may occur more than once in the body of the inductive rule. They may also involve data/size constraints to capture, for instance, sortedness and height-balancedness of trees.

Contributions. We define CSLTP, a compositional fragment of SL with *tree predicates*, where typical tree structures involving data and size constraints (e.g., binary search trees, AVL trees, and red-black trees) can be expressed. The basic rationale of CSLTP is to focus on the compositional predicates introduced in [12,13] while restricting to dense-order data constraints and difference-bound size constraints. We remark that compositionality is vital for (deductive) program verification without which the entailment checking, an indispensable procedure for checking assertions in the style of Hoare logic, would otherwise be exceedingly difficult. (The price is that, instead of trees, one has to consider trees with *one hole* to guarantee the compositionality; cf. Sect. 3.) Our main contribution is summarised as follows:

(i) We provide algorithms to compute the least fixed point of the inductively defined predicates involving data/size constraints derived from CSLTP formulae (see Theorem 2). To this end, we employ a wide range of techniques from closed-form evaluation of Datalog programs with integer gap-order constraints [28], computation of reachability sets of alternating one-counter systems [4], and the decision procedure for the reachability problem of one-counter automata [15]. In addition, we show that computation of the least fixed point of the inductively defined predicates beyond CSLTP may be difficult in general. More specifically, we prove that, for the predicate corresponding to AVL trees with one hole where *all* parameters are of the *natural number* type, its least fixed point is *inexpressible* in Presburger arithmetic (see Theorem 1).

(ii) We propose a *complete* decision procedure for the satisfiability problem of CSLTP. Namely, from each CSLTP formula φ we define $\mathsf{Abs}(\varphi)$ as an abstraction of φ such that φ and $\mathsf{Abs}(\varphi)$ are equisatisfiable. Roughly speaking, $\mathsf{Abs}(\varphi)$ introduces Boolean variables to encode the spatial part of φ and encompasses computed least fixed points from (i) to address the data and size constraints. We then can resort to the state-of-the-art SMT solvers (e.g., Z3 [34]). We remark that most decision procedures for satisfiability of SL with inductive definitions *and* data/size constraints are incomplete (see the *related work* for more details).

Satisfiability checking serves as a cornerstone towards a complete procedure for entailment checking, which requires a separate paper to solve. It can also be widely used in, e.g., consistency checking of specifications written in SL, symbolic execution of programs manipulating dynamic data structures (see [2,20]), etc.

Related work. For SLID *without* data constraints, [6] provides a complete decision procedure, setting the satisfiability problem (almost) completely. We also mention some earlier results [2,17] which focus on the symbolic heap fragments for list segments and binary trees, providing complete proof systems. [12] proposes a compositional fragment of SLID equipped with an incomplete decision procedure. In addition, [18,19] provide complete decision procedures for the entailment problem of SLID (without data/size constraints) by reducing to the language inclusion problem of tree automata.

Towards adding data/size constraints, [29] presents a complete decision procedure for the quantifier-free fragment of SL (*without* inductive definitions) interpreted over heaplets with data elements ranging over a parametric multi-sorted (possibly infinite) domain. For SLID *with* data constraints, [8] provides an incomplete decision procedure based on invariants of inductive definitions. These invariants are essentially the fixed points of the inductively defined predicates involving data/size constraints, and are supposed to be provided by the users. [3] specifies the data/size constraints by universal quantifiers over the index variables (and thus is able to express set/multiset constraints), but restricts to the singly linked lists only. [23,27] reduces the entailment problem of SLID with data/size constraints to the satisfiability problem in the theory of uninterpreted functions, though the procedure therein is *incomplete* and *not* fully automatic since it relies on the users to provide lemmas. [24–26] encode SLID into a fragment of first-order logic with reachability predicates (whose satisfiability is decidable in NP). However, this fragment cannot accommodate the size or multiset constraints. More recently, [20] considers the data constraints expressible in Presburger arithmetic. The decision procedure therein is based on cyclic proofs [5,9] and is incomplete in general and is complete for a syntactic fragment defined with a specialized well-founded notion, which is incomparable to CSLTP.

With respect to data/size constraints, [33] is closest to our work, where the data/size constraints are expressed in Presburger arithmetic, and a complete decision procedure is given for the satisfiability problem. CSLTP differs from the fragment in [33] in both the shape properties and the data/size constraints: 1) For the shape properties, CSLTP addresses trees *with one hole* (which is crucial for the compositionality), while [33] does not. 2) For the data/size constraints, the class of data constraints in [33] is incomparable to that of CSLTP: On the one hand, CSLTP allows only one integer parameter, while [33] may have multiple ones, although there must be a dominating one. On the other hand, the order constraints (e.g. sortedness), which require comparing different data parameters and are covered by CSLTP, are inexpressible in [33]. In addition, even when restricted to size constraints, CSLTP goes beyond the fragment in [33]. For instance, the height-balancedness of red-black trees can be easily expressed in CSLTP, whereas it is inexpressible in [33]. This is because the inductive definition in [33] essentially allows only *one* inductive rule, with the aid of the max and min functions and (a form of) disjunctions in the data/size constraint.

Nevertheless, the height-balancedness of red-black trees requires multiple inductive rules to specify, even when max, min and disjunctions are present in the data/size constraint. Furthermore, we employ an automata-theoretic approach to compute the least fixed point of data predicates, which is quite different from the arguments ([33]) which are purely based on induction.

There are methods outside of the SL framework to tackle verification of tree structures and data constraints. Some of them are based on different extensions of tree automata, such as forest automata [1], tree automata with size constraints [16], ree automata with height constraints [11], and visibly tree automata with memory and constraints [10]. Interestingly, our approach to compute the least fixed point of data predicates is partially inspired by this line of work, especially [16]. Even further, [21] takes a logic-based approach to verify balanced trees. Finally, [31] proposes practical approaches for solving Horn-clause constraints, which are related to, albeit easier than, computing the least fixed point of data predicates in this paper. The method therein is based on the construction of disjunctive interpolants, which are used within an abstraction-refinement loop. The method therein is incomplete in general.

2 Preliminaries

Throughout the paper, $\mathbb{Z}$ and $\mathbb{N}$ denote the set of integers and natural numbers respectively. For each $n \in \mathbb{N}$, $[n] := \{1, \ldots, n\}$. For each vector $\boldsymbol{\alpha} = (a_1, \ldots, a_n)$, $|\boldsymbol{\alpha}|$ denotes the length of $\boldsymbol{\alpha}$ (i.e. n) and $\boldsymbol{\alpha}(i)$ denotes a_i for $i \in [n]$.

Definition 1 (A1CS and N1CS). *An* alternating one-counter system *(A1CS) is a pair $\mathcal{A} = (Q, \Theta)$, where Q is a finite set of* states*, and $\Theta \subseteq Q \times 2^{(\mathsf{Inst} \times Q)}$ is a finite set of transition rules, where $\mathsf{Inst} = \{\mathfrak{o}\ n, +n, -n, \mathsf{reset}(n)\}$ with $\mathfrak{o} \in \{=, \leqslant, \geqslant\}$ and $n \in \mathbb{N}$. A transition $(p, \{(\ell_1, q_1), \cdots, (\ell_k, q_k)\}) \in \Theta$ is usually written as $p \hookrightarrow \{(\ell_1, q_1), \cdots, (\ell_k, q_k)\}$ for readability. A* nondeterministic one-counter system *(N1CS) is an A1CS where for each $p \hookrightarrow \{(\ell_1, q_1), \cdots, (\ell_k, q_k)\}$, $k = 1$.*

A *configuration* of an A1CS $\mathcal{A}$ is (p, n) where $p \in Q$ and $n \in \mathbb{N}$ is the value of the counter. The transition rules induce a transition relation on configurations in an expected way: for $p \hookrightarrow \{(\ell_1, q_1), \cdots, (\ell_k, q_k)\} \in \Theta$, we have a *hyper-transition* $(p, n) \to \{(q_1, n_1), \cdots, (q_k, n_k)\}$ if for each $1 \leqslant i \leqslant k$, (1) $\ell_i = \mathfrak{o}\ n'$ implies that $n\ \mathfrak{o}\ n'$ and $n_i = n$, (2) $\ell_i = +n'$ implies that $n_i = n+n'$, (3) $\ell_i = -n'$ implies that $n - n' \geqslant 0$ and $n_i = n - n'$, and (4) $\ell_i = \mathsf{reset}(n')$ implies that $n_i = n'$. In this case, we say that (p, n) is the *immediate predecessor* of $\{(q_1, n_1), \cdots, (q_k, n_k)\}$.

A computation tree of $\mathcal{A}$ is a directed tree whose nodes are labelled by configurations, and where every node is either a leaf or an internal node which is labelled by a configuration c and has k children labelled by $c_1, \ldots, c_k$ respectively, satisfying that $c \to \{c_1, \ldots, c_k\}$ is a hyper-transition of $\mathcal{A}$. We define the reachability relation $\Rightarrow_{\mathcal{A}}$ as $c \Rightarrow_{\mathcal{A}} C$ if there exists a computation tree such that c labels the root and C is the set of labels of the leaves. If $c \Rightarrow_{\mathcal{A}} C$, then we say that C is reachable from c in $\mathcal{A}$. For $q \in Q$ and a set of configurations C, we use $\mathsf{Pre}^*_{\mathcal{A}}(q, C)$ to denote the set of $n \in \mathbb{N}$ such that $(q, n) \Rightarrow_{\mathcal{A}} C'$ for some $C' \subseteq C$.

The transition relation for an N1CS can be defined similarly, and is simpler in that the computation tree degenerates to a single path of configurations.

Proposition 1 **([4,7,15,32]).** *The following facts hold for A1CS and N1CS.*

1. *Let $\mathcal{A} = (Q, \Theta)$ be an A1CS, $q \in Q$ be a state, C be a* finite *set of configurations of $\mathcal{A}$. Then a quantifier-free Presburger formula $\varphi_{q,C}(x)$ in disjunctive normal form can be computed in doubly exponential time to represent* $\mathsf{Pre}^*_{\mathcal{A}}(q, C)$*. In addition, if the constants in $\mathcal{A}$ and C are encoded in unary, then the computation is in exponential time.*
2. *Let $\mathcal{A} = (Q, \Theta)$ be an N1CS, and $p, q \in Q$. Then a quantifier-free Presburger formula $\varphi_{p,q}(x, y)$ can be computed in triply exponential time to represent the relation $\{(m, n) \in \mathbb{N}^2 \mid (p, m) \Rightarrow_{\mathcal{A}} (q, n)\}$. In addition, if the constants in $\mathcal{A}$ are encoded in unary, then the computation is in doubly exponential time.*

3 Compositional Separation Logic with Tree Predicates

In this section, we introduce the *compositional separation logic with tree predicates*, denoted by $\mathsf{CSLTP}[P]$, where P is an *inductive predicate*. We consider three data types, i.e., *location* type $\mathbb{L}$, *value* type $\mathbb{D}$, and *size* type $\mathbb{N}$. Intuitively, $\mathbb{D}$ represents the data values stored in the nodes of tree structures, and $\mathbb{N}$ represents the size of tree structures (e.g. height of trees), which we assume to be natural numbers. As a convention, we use $l, l', \cdots \in \mathbb{L}$ to denote locations, $d, d', \cdots \in \mathbb{D}$ to denote values, and $n, n', \cdots \in \mathbb{N}$ to denote sizes. Accordingly, variables in $\mathsf{CSLTP}[P]$ comprise *location variables* LVars ranged over by uppercase letters $E, F, X, Y, \cdots$, *value variables* DVars ranged over by $x, y, \cdots$, and *size variables* IVars ranged over by $h, i, j, \cdots$.

We consider two kinds of *fields*, i.e., location fields from $\mathcal{F}$ and data fields from $\mathcal{D}$. Each field $\mathfrak{f} \in \mathcal{F}$ (resp. $\mathfrak{d} \in \mathcal{D}$) is associated with $\mathbb{L}$ (resp. $\mathbb{D}$). We assume $\mathbb{D}$ is an *ordered, countably infinite, dense* set. That is, $\mathbb{D}$ is equipped with $<$ such that for each $d < d' \in \mathbb{D}$, $d'' \in \mathbb{D}$ exists with $d < d'' < d'$. Examples of $\mathbb{D}$ include the set of rationals with the natural order relation, and the set of strings with the lexicographical order relation. Note that any arithmetic over $\mathbb{D}$ is disregarded.

$\mathsf{CSLTP}[P]$ formulae may contain tree predicates, each of which is of the form $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$ and has an associated inductive definition. The parameters of a tree predicate are classified into two groups: *source parameters* $E, \boldsymbol{\alpha}$ and *destination parameters* $F, \boldsymbol{\beta}$. We require that the source parameters $E, \boldsymbol{\alpha}$ and the destination parameters $F, \boldsymbol{\beta}$ are *matched* in types, namely, E and F are of the location type, and two tuples $\boldsymbol{\alpha}, \boldsymbol{\beta}$ have the same length $\ell > 0$ and for each $i : 1 \leqslant i \leqslant \ell$, both α_i and β_i have the natural number type or the value type. The parameters E, F are called the *location parameters* of P and $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are called the *data parameters* of P. Intuitively, a tree predicate $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$ defines binary trees with one hole and data constraints.

The $\mathsf{CSLTP}[P]$ formulae comprise three types of formulae: *pure formulae* $\varPi$, *data formulae* $\varDelta$, and *spatial formulae* $\varSigma$, which are defined as follows,

$$\begin{array}{rcll}
\Pi & ::= & E = F \mid E \neq F \mid \Pi \wedge \Pi & \text{(pure formulae)} \\
\Delta & ::= & \Delta_{\mathbb{D}} \wedge \Delta_{\mathbb{N}} & \text{(data formulae)} \\
\Delta_{\mathbb{D}} & ::= & \mathtt{true} \mid x\ \mathfrak{o}\ d \mid x\ \mathfrak{o}\ x' \mid \Delta_{\mathbb{D}} \wedge \Delta_{\mathbb{D}} & \text{(value formulae)} \\
\Delta_{\mathbb{N}} & ::= & \mathtt{true} \mid h\ \mathfrak{o}\ n \mid h\ \mathfrak{o}\ h' + n \mid \Delta_{\mathbb{N}} \wedge \Delta_{\mathbb{N}} & \text{(size formulae)} \\
\Sigma & ::= & \mathtt{emp} \mid E \mapsto \rho \mid P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}) \mid \Sigma * \Sigma & \text{(spatial formulae)} \\
\rho & ::= & \rho_{\mathsf{f}}, \rho_{\mathsf{d}} & \text{(field-variable sequences)} \\
\rho_{\mathsf{f}} & ::= & (\mathfrak{f}, X) \mid \rho_{\mathsf{f}}, \rho_{\mathsf{f}} & \text{(location field-variable sequences)} \\
\rho_{\mathsf{d}} & ::= & (\mathfrak{d}, x) \mid \rho_{\mathsf{d}}, \rho_{\mathsf{d}} & \text{(data field-variable sequences)}
\end{array}$$

where $\mathfrak{o} \in \{=, <, >, \leqslant, \geqslant\}$, $\mathfrak{f} \in \mathcal{F}$, and $\mathfrak{d} \in \mathcal{D}$. For spatial formulae Σ, formulae of the form $\mathtt{emp}$, $E \mapsto \rho$, or $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$ are called *spatial atoms*. In particular, formulae of the form $E \mapsto \rho$ and $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$ are called *points-to atoms* and *predicate atoms* respectively.

A *tree predicate* P (with one hole) is defined by one base rule, and at least one inductive rule of the form R_1 or R_2:

- base rule R_0: $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}) ::= E = F \wedge \boldsymbol{\alpha} = \boldsymbol{\beta} \wedge \mathtt{emp}$,

- left-hole inductive rule R_1:
$$P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}) ::= \exists X \exists Y \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \Delta \wedge E \mapsto ((\mathtt{left}, X), (\mathtt{right}, Y), \rho_{\mathsf{d}}) * P(X, \boldsymbol{\delta}; F, \boldsymbol{\beta}) * P(Y, \boldsymbol{\gamma}; \mathsf{nil}, \boldsymbol{\epsilon}),$$
where Δ is a data formula and ρ_{d} is a data field-variable sequence.

- right-hole inductive rule R_2:
$$P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}) ::= \exists X \exists Y \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \Delta \wedge E \mapsto ((\mathtt{left}, X), (\mathtt{right}, Y), \rho_{\mathsf{d}}) * P(X, \boldsymbol{\gamma}; \mathsf{nil}, \boldsymbol{\epsilon}) * P(Y, \boldsymbol{\delta}; F, \boldsymbol{\beta}),$$
where Δ is a data formula and ρ_{d} is a data field-variable sequence.

The left-hand (resp. right-hand) side of a rule is called the *head* (resp. *body*) of the rule. We note that the bodies of R_1 and R_2 do not contain pure formulae.

In the sequel, we specify some constraints on the inductive rules.

The first constraint **C1** guarantees that $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$ enjoys the composition lemma $P(E_1, \boldsymbol{\alpha}_1; E_2, \boldsymbol{\alpha}_2) * P(E_2, \boldsymbol{\alpha}_2; E_3, \boldsymbol{\alpha}_3) \Rightarrow P(E_1, \boldsymbol{\alpha}_1; E_3, \boldsymbol{\alpha}_3)$, which is vital for compositionality (cf. [13]). Note that the destination parameter F does not occur elsewhere in the body of the inductive rules by definition, since X, Y are two existentially quantified location variables.

C1 Variables from $\boldsymbol{\beta}$ do *not* occur elsewhere in the body of the inductive rules.

The second constraint **C2** forbids the repeated occurrences of the variables in $\boldsymbol{\gamma}, \boldsymbol{\delta}$ and requires that no existentially quantified variables occur in the static parameters $\boldsymbol{\epsilon}$.

C2 $\boldsymbol{\gamma}, \boldsymbol{\delta} \subseteq \boldsymbol{\alpha} \cup \boldsymbol{x} \cup \boldsymbol{h} \cup \mathbb{D} \cup \mathbb{N}$, each variable occurs at most once in $\boldsymbol{\gamma}$ (resp. $\boldsymbol{\delta}$), and $\boldsymbol{\epsilon} \subseteq \boldsymbol{\alpha} \cup \mathbb{D} \cup \mathbb{N}$.

The third constraint **C3** forbids the situation that an existentially quantified variable occurs only in Δ, but not in spatial atoms.

C3 All existentially quantified variables $\boldsymbol{x}, \boldsymbol{h}$ occur in some spatial atom.

The fourth constraint **C4** is to avoid the difficulty of dealing with inductive predicates with more than one size source parameter (cf. Theorem 1).

C4 $\boldsymbol{\alpha}$ contains at most *one* parameter of the *size* type, in addition, if $\boldsymbol{\alpha}(i)$ is of size type, then it must hold that, (i) $\boldsymbol{\delta}(i), \boldsymbol{\gamma}(i) \in \boldsymbol{h}$ and $\boldsymbol{\epsilon}(i) \in \mathbb{N}$, and (ii) the size-formula part of Δ is of the form $\boldsymbol{\alpha}(i) = \boldsymbol{\delta}(i) + n \wedge \Delta_{\mathbb{N}}$ or $\boldsymbol{\alpha}(i) = \boldsymbol{\gamma}(i) + n \wedge \Delta_{\mathbb{N}}$ such that $\boldsymbol{\alpha}(i)$ does not occur in $\Delta_{\mathbb{N}}$.

For a tree predicate P, let $\mathrm{Flds}(P)$ (resp. $\mathrm{LFlds}(P)$) denote the set of fields (resp. location fields) occurring in the inductive rules of P. Evidently, $\mathrm{LFlds}(P) = \{\mathtt{left}, \mathtt{right}\}$. For a spatial atom a, let $\mathrm{Flds}(a)$ denote the set of fields that a refers to: if $a = E \mapsto \rho$, then $\mathrm{Flds}(a)$ is the set of fields occurring in ρ; if $a = P(-)$, then $\mathrm{Flds}(a) = \mathrm{Flds}(P)$.

We write $\mathsf{CSLTP}[P]$ for the collection of separation logic formulae $\varphi = \Pi \wedge \Delta \wedge \Sigma$ such that P is the only tree predicate allowed to appear in Σ, and for each points-to atom occurring in Σ, the set of fields of this atom is $\mathrm{Flds}(P)$. For a $\mathsf{CSLTP}[P]$ formula φ, let $\mathsf{Vars}(\varphi)$ (resp. $\mathsf{LVars}(\varphi)$, $\mathsf{DVars}(\varphi)$, $\mathsf{IVars}(\varphi)$) denote the set of (resp. location, value, size) variables occurring in φ. Moreover, we use $\varphi[\boldsymbol{\mu}/\boldsymbol{\alpha}]$ to denote the simultaneous replacement of the variables α_j by μ_j in φ.

For the semantics of $\mathsf{CSLTP}[P]$, each formula is interpreted on states. Formally, a *state* is a pair $(\mathfrak{s}, \mathfrak{h})$, where

- $\mathfrak{s}$ is an assignment function which is a partial function from $\mathsf{LVars} \cup \mathsf{DVars} \cup \mathsf{IVars}$ to $\mathbb{L} \cup \mathbb{D} \cup \mathbb{N}$ such that $dom(\mathfrak{s})$ is finite and $\mathfrak{s}$ respects the data type,
- $\mathfrak{h}$ is a *heap* which is a partial function from $\mathbb{L} \times (\mathcal{F} \cup \mathcal{D})$ to $\mathbb{L} \cup \mathbb{D}$ such that
  - $\mathfrak{h}$ respects the data type of fields, that is, for each $l \in \mathbb{L}$ and $\mathfrak{f} \in \mathcal{F}$ (resp. $l \in \mathbb{L}$ and $\mathfrak{d} \in \mathcal{D}$), if $\mathfrak{h}(l, \mathfrak{f})$ (resp. $\mathfrak{h}(l, \mathfrak{d})$) is defined, then $\mathfrak{h}(l, \mathfrak{f}) \in \mathbb{L}$ (resp. $\mathfrak{h}(l, \mathfrak{d}) \in \mathbb{D}$); and
  - $\mathfrak{h}$ is field-consistent, i.e. every location in $\mathfrak{h}$ possess the same set of fields.

For a heap $\mathfrak{h}$, we use $\mathsf{ldom}(\mathfrak{h})$ to denote the set of locations $l \in \mathbb{L}$ such that $\mathfrak{h}(l, \mathfrak{f})$ or $h(l, \mathfrak{d})$ is defined for some $\mathfrak{f} \in \mathcal{F}$ and $\mathfrak{d} \in \mathcal{D}$. Moreover, we use $\mathrm{Flds}(\mathfrak{h})$ to denote the set of fields $\mathfrak{f} \in \mathcal{F}$ or $\mathfrak{d} \in \mathcal{D}$ such that $\mathfrak{h}(l, \mathfrak{f})$ or $\mathfrak{h}(l, \mathfrak{d})$ is defined for some $l \in \mathbb{L}$.

Two heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are said to be *field-compatible* if $\mathrm{Flds}(\mathfrak{h}_1) = \mathrm{Flds}(\mathfrak{h}_2)$. We write $\mathfrak{h}_1 \# \mathfrak{h}_2$ if $\mathsf{ldom}(\mathfrak{h}_1) \cap \mathsf{ldom}(\mathfrak{h}_2) = \varnothing$. Moreover, we write $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ for the disjoint union of two field-compatible fields $\mathfrak{h}_1$ and $\mathfrak{h}_2$ (this implies that $\mathfrak{h}_1 \# \mathfrak{h}_2$).

Let $(\mathfrak{s}, \mathfrak{h})$ be a state and φ be an $\mathsf{CSLTP}[P]$ formula. The semantics of $\mathsf{CSLTP}[P]$ formulae is defined as follows,

- $(\mathfrak{s}, \mathfrak{h}) \models E = F$ (resp. $(\mathfrak{s}, \mathfrak{h}) \models E \neq F$) if $\mathfrak{s}(E) = \mathfrak{s}(F)$ (resp. $\mathfrak{s}(E) \neq \mathfrak{s}(F)$),
- $(\mathfrak{s}, \mathfrak{h}) \models \Pi_1 \wedge \Pi_2$ if $(\mathfrak{s}, \mathfrak{h}) \models \Pi_1$ and $(\mathfrak{s}, \mathfrak{h}) \models \Pi_2$,
- $(\mathfrak{s}, \mathfrak{h}) \models x \ \mathfrak{o} \ c$ (resp. $(\mathfrak{s}, \mathfrak{h}) \models x \ \mathfrak{o} \ x'$) if $\mathfrak{s}(x) \ \mathfrak{o} \ c$ (resp. $\mathfrak{s}(x) \ \mathfrak{o} \ \mathfrak{s}(x')$),
- $(\mathfrak{s}, \mathfrak{h}) \models h \ \mathfrak{o} \ c$ (resp. $(\mathfrak{s}, \mathfrak{h}) \models h \ \mathfrak{o} \ h' + c$) if $\mathfrak{s}(h) \ \mathfrak{o} \ c$ (resp. $\mathfrak{s}(h) \ \mathfrak{o} \ \mathfrak{s}(h') + c$),
- $(\mathfrak{s}, \mathfrak{h}) \models \Delta_1 \wedge \Delta_2$ if $(\mathfrak{s}, \mathfrak{h}) \models \Delta_1$ and $(\mathfrak{s}, \mathfrak{h}) \models \Delta_2$,
- $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{emp}$ if $\mathsf{ldom}(\mathfrak{h}) = \varnothing$,
- $(\mathfrak{s}, \mathfrak{h}) \models E \mapsto \rho$ if $\mathsf{ldom}(\mathfrak{h}) = \mathfrak{s}(E)$, and for each $(\mathfrak{f}, X) \in \rho$ (resp. $(\mathfrak{d}, x) \in \rho$), $\mathfrak{h}(\mathfrak{s}(E), \mathfrak{f}) = \mathfrak{s}(X)$ (resp. $\mathfrak{h}(\mathfrak{s}(E), \mathfrak{d}) = \mathfrak{s}(x)$),
- $(\mathfrak{s}, \mathfrak{h}) \models P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$ if $(\mathfrak{s}, \mathfrak{h}) \in [\![P(E, \boldsymbol{\alpha}; \boldsymbol{F}, \boldsymbol{\beta})]\!]$,
- $(\mathfrak{s}, \mathfrak{h}) \models \Sigma_1 * \Sigma_2$ if there are $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models \Sigma_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models \Sigma_2$.

where the semantics of predicates $[\![P(E,\boldsymbol{\alpha};F,\boldsymbol{\beta})]\!]$ is given by the least fixed point of a monotone operator constructed from the body of rules for P in a standard way as in [6].

For a formula φ, let $[\![\varphi]\!]$ denote the set of states $(\mathfrak{s},\mathfrak{h})$ such that $(\mathfrak{s},\mathfrak{h}) \models \varphi$. We focus on the satisfiability problem, i.e., given a $\mathsf{CSLTP}[P]$ formula φ, decide whether $[\![\varphi]\!]$ is empty.

Example 1. The first example *bsth* specifies *binary search trees with one hole*, which exemplifies the usage of value variables for the sortedness constraints. Here x, y represent the lower and upper bounds of the data values from $\mathbb{D}$.

$$\begin{aligned}
bsth(E,x,y;F,x',y') &::= E = F \wedge x = x' \wedge y = y' \wedge \mathtt{emp},\\
bsth(E,x,y;F,x',y') &::= \exists X,Y,z,x'',y''.\ y'' < z < x'' \wedge\\
&\qquad E \mapsto ((\mathtt{left},X),(\mathtt{right},Y),(\mathtt{data},z)) *\\
&\qquad bsth(X,x,y'';F,x',y') * bsth(Y,x'',y;\mathsf{nil},y,y),\\
bsth(E,x,y;F,x',y') &::= \exists X,Y,z,x'',y''.\ y'' < z < x'' \wedge\\
&\qquad E \mapsto ((\mathtt{left},X),(\mathtt{right},Y),(\mathtt{data},z)) *\\
&\qquad bsth(X,x,y'';\mathsf{nil},x,x) * bsth(Y,x'',y;F,x',y').
\end{aligned}$$

Note that a binary search tree can be simply defined as $bsth(E,x,y;\mathsf{nil},x,x)$ or $bsth(E,x,y;\mathsf{nil},y,y)$, where E is the root, and x, y are the lower respective upper bounds for the data values occurring in the tree nodes.

The second example *balthole* specifies *height-balancedness of AVL-trees with one hole*, which exemplifies the usage of size parameters. Here $h \in \mathbb{N}$ represents the height of the tree.

$$\begin{aligned}
balthole(E,h;F,h') &::= E = F \wedge h = h' \wedge \mathtt{emp},\\
balthole(E,h;F,h') &::= \exists X,Y,h_1,h_2.\ h_1 \leqslant h_2 \leqslant h_1 + 1 \wedge h = h_2 + 1 \wedge\\
&\qquad E \mapsto ((\mathtt{left},X),(\mathtt{right},Y)) * balthole(X,h_1;F,h') * balthole(Y,h_2;\mathsf{nil},0),\\
balthole(E,h;F,h') &::= \exists X,Y,h_1,h_2.\ h = h_1 + 1 \wedge h_1 = h_2 + 1 \wedge\\
&\qquad E \mapsto ((\mathtt{left},X),(\mathtt{right},Y)) * balthole(X,h_1;F,h') * balthole(Y,h_2;\mathsf{nil},0),\\
balthole(E,h;F,h') &::= \exists X,Y,h_1,h_2.\ h_1 \leqslant h_2 \leqslant h_1 + 1 \wedge h = h_2 + 1 \wedge\\
&\qquad E \mapsto ((\mathtt{left},X),(\mathtt{right},Y)) * balthole(X,h_1;\mathsf{nil},0) * balthole(Y,h_2;F,h'),\\
balthole(E,h;F,h') &::= \exists X,Y,h_1,h_2.\ h = h_1 + 1 \wedge h_1 = h_2 + 1 \wedge\\
&\qquad E \mapsto ((\mathtt{left},X),(\mathtt{right},Y)) * balthole(X,h_1;\mathsf{nil},0) * balthole(Y,h_2;F,h').
\end{aligned}$$

The definitions of *bsth* and *balthole* can be combined to form a tree predicate $avlth(E,x,y,h;F,x',y',h')$, which specifies both the *sortedness* and the *height-balancedness* property of AVL-trees with one hole.

4 The Least Fixed Point of Data Predicates

Let $P(E,\boldsymbol{\alpha};F,\boldsymbol{\beta})$ be a tree predicate. The *data predicate* induced by P, denoted by $P_D(\boldsymbol{\alpha};\boldsymbol{\beta})$, is the predicate whose definition is obtained from the rules of P by ignoring the spatial variables and spatial atoms. Formally, $P_D(\boldsymbol{\alpha};\boldsymbol{\beta})$ is defined by the rules of the following form,

- base rule: $P_D(\boldsymbol{\alpha}; \boldsymbol{\beta}) ::= \boldsymbol{\alpha} = \boldsymbol{\beta}$,
- for each left-hole inductive rule
  $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}) ::= \exists X, Y \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \Delta \wedge E \mapsto ((\texttt{left}, X), (\texttt{right}, Y), \rho_{\mathsf{d}}) * P(X, \boldsymbol{\delta}; F, \boldsymbol{\beta}) * P(Y, \boldsymbol{\gamma}; \mathsf{nil}, \boldsymbol{\epsilon})$,
  there is an inductive rule for P_D of the form:
  $$P_D(\boldsymbol{\alpha}; \boldsymbol{\beta}) ::= \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \Delta \wedge P_D(\boldsymbol{\delta}; \boldsymbol{\beta}) \wedge P_D(\boldsymbol{\gamma}; \boldsymbol{\epsilon}),$$
- similarly for the right-hole inductive rules.

Naturally, $P_D(\boldsymbol{\alpha}; \boldsymbol{\beta})$ induces a monotonic function and we use $\mathsf{lfp}(P_D)$ to denote its least fixed point.

We start with a "negative" result stating that, if multiple size source parameters were allowed in the tree predicates then $\mathsf{lfp}(P_D)$ would be inexpressible in Presburger arithmetic in general. This result underpins the constraint **C4** which dictates that only one source parameter of type $\mathbb{N}$ is allowed.

Theorem 1. *If x, y, x', y' in $avlth(E, x, y, h; F, x', y', h')$ are assumed to be of the type $\mathbb{N}$, then $\mathsf{lfp}(avlth_D)$ is inexpressible in Presburger arithmetic.*

The intuition of Theorem 1 is explained as follows: If the data values in AVL-trees are assumed to be natural numbers, then in $avlth(E, x, y, h; \mathsf{nil}, x, x, 0)$, the predicate atom for AVL trees, $y - x$ correlates with h and is at least exponential in h. This relationship goes beyond Presburger arithmetic.

Next, for a tree predicate P in CSLTP, we show that a linear arithmetic formula can be computed to represent $\mathsf{lfp}(P_D)$.

Theorem 2. *A linear arithmetic formula can be computed in 5-fold exponential time to represent $\mathsf{lfp}(P_D)$. In addition, if the natural-number constants in the inductive definition of P_D are encoded in unary, then the complexity is reduced to 4-fold exponential time.*

The rest of this section is devoted to the proof of Theorem 2. We start with two simpler cases, i.e., **dense order constraints** and **single size parameter**.

4.1 Dense Order Constraints

In this subsection, we fix a tree predicate $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$ where all parameters in $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are of the type $\mathbb{D}$. As a result, only value formulae $\Delta_{\mathbb{D}}$ are used in $P_D(\boldsymbol{\alpha}; \boldsymbol{\beta})$. Let $\mathcal{C}(P_D)$ denote the set of constants occurring in the rules of P_D.

Definition 2 (Order graphs). *Let V be a finite subset of $\mathsf{DVars} \cup \mathbb{D}$. An order graph G on V is an edge-labelled graph (V, E), where $E \subseteq V \times \{\leqslant, <\} \times V$.*

It is evident that order graphs are simply another representation of value formulae, which are dense order constraints on $\mathbb{D}$. More specifically, from an order graph G on V, a dense order constraint $\Delta_{\mathbb{D}}(G)$ can be naturally defined. On the other hand, an order graph $G_{\Delta_{\mathbb{D}}}$ can be constructed from a value formula $\Delta_{\mathbb{D}}$. For two order graphs G_1, G_2, we will use $G_1 \models G_2$ to denote $\Delta_{\mathbb{D}}(G_1) \models \Delta_{\mathbb{D}}(G_2)$.

Definition 3 (Saturated order graphs). *Assume an order graph $G = (V, E)$. The saturated graph of G, denoted by $\mathsf{Sat}(G)$, is computed from G by the following procedure:*

1. *Initially, let $\mathsf{Sat}(G) := G$.*
2. *Repeat the following procedure until no more edges can be added to $\mathsf{Sat}(G)$.*
   - *If there are two edges $(v_1, \mathfrak{o}_1, v_2)$ and $(v_2, \mathfrak{o}_2, v_3)$ in $\mathsf{Sat}[G]$ such that $\mathfrak{o}_1$ and $\mathfrak{o}_2$ are both $\leqslant$ and $(v_1, \leqslant, v_3)$ is not an edge in $\mathsf{Sat}(G)$, then add $(v_1, \leqslant, v_3)$ into $\mathsf{Sat}(G)$.*
   - *If there are two edges $(v_1, \mathfrak{o}_1, v_2)$ and $(v_2, \mathfrak{o}_2, v_3)$ in $\mathsf{Sat}(G)$ such that at least one of $\mathfrak{o}_1$ and $\mathfrak{o}_2$ is $<$ and $(v_1, <, v_3)$ is not an edge in $\mathsf{Sat}(G)$, then add $(v_1, <, v_3)$ into $\mathsf{Sat}(G)$.*

$\mathsf{Sat}(G)$ *is said to be* consistent *if it does not contain edges of the form $(v, <, v)$ for $v \in V$. Otherwise, it is said to be* inconsistent.

Proposition 2. *Let $\varDelta_{\mathbb{D}}$ be a value formula. Then $\varDelta_{\mathbb{D}}$ is satisfiable iff $\mathsf{Sat}(G_{\varDelta_{\mathbb{D}}})$ is consistent.*

For a finite set $V \subseteq \mathsf{DVars} \cup \mathbb{D}$, we use $\mathcal{G}_{\mathsf{ord}}(V)$ to denote the set of *consistent saturated* order graphs on V. Note that the cardinality of $\mathcal{G}_{\mathsf{ord}}(V)$ is exponential in the size of V.

To compute $\mathsf{lfp}(P_D)$, let $V = \boldsymbol{\alpha} \cup \boldsymbol{\beta} \cup \mathcal{C}(P_D)$. We define a monotone function $\mathcal{T}_{P_D} : 2^{\mathcal{G}_{\mathsf{ord}}(V)} \to 2^{\mathcal{G}_{\mathsf{ord}}(V)}$ to capture $P_D(\boldsymbol{\alpha}; \boldsymbol{\beta})$, and compute $\mathsf{lfp}(\mathcal{T}_{P_D})$ by a standard iteration: let $\mathcal{G}_0 = \varnothing$, and $\mathcal{G}_i := \mathcal{T}_{P_D}(\mathcal{G}_{i-1})$ until the iteration stabilises. The algorithm terminates in exponential time, since $\mathcal{T}_{P_D}$ is monotone and the cardinality of $\mathcal{G}_{\mathsf{ord}}(V)$ is exponential in the size of V.

Suppose $|\boldsymbol{\alpha}| = k$. For a vector $\boldsymbol{d}, \boldsymbol{d'} \in \mathbb{D}^k$, define an order graph $\mathcal{G}_{\boldsymbol{d},\boldsymbol{d'}} = (V, E_{\boldsymbol{d},\boldsymbol{d'}})$ as as follows: Let $\eta : V \to \boldsymbol{d} \cup \boldsymbol{d'} \cup \mathcal{C}(P_D)$ such that $\eta(\boldsymbol{\alpha}(i)) = \boldsymbol{d}(i)$ and $\eta(\boldsymbol{\beta}(i)) = \boldsymbol{d}'(i)$ for each $i \in [k]$, and $\eta(d'') = d''$ for each $d'' \in \mathcal{C}(P_D)$. Then for each $z, z' \in V$ and $\mathfrak{o} \in \{<, \leqslant\}$, $(z, \mathfrak{o}, z') \in E_{\boldsymbol{d},\boldsymbol{d'}}$ iff $\eta(z)\ \mathfrak{o}\ \eta(z')$ holds in $\mathbb{D}$.

Proposition 3. *For any two vectors $\boldsymbol{d}, \boldsymbol{d}' \in \mathbb{D}^k$, $\mathsf{lfp}(P_D)(\boldsymbol{d}; \boldsymbol{d}')$ holds iff there exists $G \in \mathsf{lfp}(\mathcal{T}_{P_D})$ such that $G_{\boldsymbol{d},\boldsymbol{d'}} \models G$.*

4.2 Single Size Parameter

In this subsection, we fix a tree predicate P where all (data) parameters are of type $\mathbb{N}$. Then according to **C4**, the parameters of P are of the form $(E, \alpha; F, \beta)$, where α, β are of type $\mathbb{N}$, in addition, each inductive rule of the associated data predicate $P_D(\alpha; \beta)$ is of the form

$$P_D(\alpha; \beta) ::= \exists \boldsymbol{h}.\ \varDelta_{\mathbb{N}} \wedge P_D(\delta; \beta) \wedge P_D(\gamma; n). \tag{1}$$

Let $\mathcal{N}(P_D)$ denote the set of all constants n occurring in the predicate atom $P_D(\gamma; n)$ of the body $P_D(\alpha; \beta)$. By **C3** and **C4**, δ and γ are the *only* existentially quantified variables, that is, $\exists \boldsymbol{h} = \exists \delta \exists \gamma$. For each $n \in \mathcal{N}(P_D)$, we introduce a new predicate $P_{D,n}(\alpha)$, the definition of which is as follows:

- base rule: $P_{D,n}(\alpha) ::= \alpha = n$,
- inductive rules: $P_{D,n}(\alpha) ::= \exists \delta \exists \gamma.\ \Delta \wedge P_{D,n}(\delta) \wedge P_{D,n'}(\gamma)$, if there is an inductive rule $P_D(\alpha; \beta) ::= \exists \delta \exists \gamma.\ \Delta \wedge P_D(\delta; \beta) \wedge P_D(\gamma; n')$.

The general strategy to solve (1) is to first compute $\mathsf{lfp}(P_{D,n})$ as a quantifier-free Presburger formula $\varphi_{P_{D,n}}(\alpha)$ for the predicates $P_{D,n}$ with $n \in \mathcal{N}(P_D)$. We then substitute $P_D(\gamma, n')$ in the body of the inductive rule of $P_D(\alpha; \beta)$ with $\varphi_{P_{D,n'}}(\gamma)$, resulting in a new collection of inductive rules for $P_D(\alpha; \beta)$. Finally, we compute the least fixed point of the function induced by this new collection of rules of $P_D(\alpha; \beta)$.

Computation of $\mathsf{lfp}(P_{D,n})$. We will reduce the problem to the computation of the reachability sets of an A1CS $\mathcal{A}_{P_D} = (Q, \Theta)$, where Q is the union of $\{P_{D,n} \mid n \in \mathcal{N}(P_D)\}$ and a set of auxiliary states (see below), and Θ is defined according to the inductive rules of the predicates $P_{D,n}$ for $n \in \mathcal{N}(P_D)$.

Let us fix a predicate $P_{D,n}$ and an inductive rule of $P_{D,n}$

$$P_{D,n}(\alpha) ::= \exists \delta \exists \gamma.\ \Delta_{\mathbb{N}} \wedge P_{D,n}(\delta) \wedge P_{D,n'}(\gamma). \tag{2}$$

By **C4**, the size formula $\Delta_{\mathbb{N}}$ must be of the form $\alpha = \delta + m \wedge \Delta'$ or $\alpha = \gamma + m \wedge \Delta'$ such that α does not occur in Δ'. W.l.o.g., we assume that $\alpha = \delta + m$ holds. It follows that Δ' is a conjunction of difference bound constraints over δ and γ. Hence, we may constraint γ in terms of α (rather than δ; this is possible because $\alpha = \delta + m$). Namely, we may assume that $\Delta' = \Delta'_1(\alpha) \wedge \Delta'_2(\alpha, \gamma) \wedge \Delta'_3(\gamma)$, where $\Delta'_1, \Delta'_2, \Delta'_3$ are defined by the following rules,

1. $\Delta'_1(\alpha) ::= \mathtt{true} \mid \alpha \geqslant l \mid \alpha \leqslant u \mid l \leqslant \alpha \leqslant u$, where $l, u \in \mathbb{N}$,
2. $\Delta'_2(\alpha, \gamma) ::= \mathtt{true} \mid \gamma \geqslant \alpha + l \mid \gamma \leqslant \alpha + u \mid \alpha + l \leqslant \gamma \leqslant \alpha + u$, where $l, u \in \mathbb{Z}$,
3. $\Delta'_3(\gamma) ::= \mathtt{true} \mid \gamma \geqslant l \mid \gamma \leqslant u \mid l \leqslant \gamma \leqslant u$, where $l, u \in \mathbb{N}$.

Θ comprises the transition rules for each predicate $P_{D,n}$ and each inductive rule of $P_{D,n}$ as in Eq. (2), defined as follows:

- the transition rules for $\Delta'_1(\alpha)$:
  - if $\Delta'_1 = \mathtt{true}$, then $P_{D,n} \hookrightarrow \{(+0, q_1)\}$,
  - if $\Delta'_1 = \alpha \geqslant l$, then $P_{D,n} \hookrightarrow \{(\geqslant l, q_1)\}$,
  - if $\Delta'_1 = \alpha \leq u$, then $P_{D,n} \hookrightarrow \{(\leqslant u, q_1)\}$,
  - if $\Delta'_1 = l \leqslant \alpha \leqslant u$, then $P_{D,n} \hookrightarrow \{(\geqslant l, q'_1)\}$, $q'_1 \hookrightarrow \{(\leqslant u, q_1)\}$;
- the transition rules for $\alpha = \delta + m \wedge \Delta'_2(\alpha, \gamma)$:
  - if $\Delta'_2 = \mathtt{true}$, then $q_1 \hookrightarrow \{(-m, P_{D,n}), (\mathsf{reset}(0), q'_2)\}$, $q'_2 \hookrightarrow \{(+1, q'_2)\}$, and $q'_2 \hookrightarrow \{(+0, q_2)\}$,
  - if $\Delta'_2 = \gamma \geqslant \alpha + l$, then $q_1 \hookrightarrow \{(-m, P_{D,n}), (l, q'_2)\}$, $q'_2 \hookrightarrow \{(+1, q'_2)\}$, $q'_2 \hookrightarrow \{(+0, q_2)\}$,
  - if $\Delta'_2 = \gamma \leqslant \alpha + u$, then $q_1 \hookrightarrow \{(-m, P_{D,n}), (u, q'_2)\}$, $q'_2 \hookrightarrow \{(-1, q'_2)\}$, $q'_2 \hookrightarrow \{(+0, q_2)\}$,
  - if $\Delta'_2 = \alpha + l \leqslant \gamma \leqslant \alpha + u$, then $q_1 \hookrightarrow \{(-m, P_{D,n}), (m', q_2)\}$ for each $l \leqslant m' \leqslant u$;

- the transition rules for $\Delta'_3(\gamma)$:
  - if $\Delta'_3 = \texttt{true}$, then $q_2 \hookrightarrow \{(+0, P_{D,n'})\}$,
  - if $\Delta'_3 = \gamma \geqslant l$, then $q_2 \hookrightarrow \{(\geqslant l, P_{D,n'})\}$,
  - if $\Delta'_3 = \gamma \leqslant u$, then $q_2 \hookrightarrow \{(\leqslant u, P_{D,n'})\}$,
  - if $\Delta'_3 = l \leqslant \gamma \leqslant u$, then $q_2 \hookrightarrow \{(\geqslant l, q'_3)\}$, $q'_3 \hookrightarrow \{(\leqslant u, P_{D,n'})\}$,

where $q_1, q_2, q'_1, q'_2, q'_3$ are the auxiliary (control) states.

For each predicate $P_{D,n}$, we use $\mathcal{P}(P_{D,n})$ to denote the set of predicates $P_{D,n'}$ such that $P_{D,n'}$ occurs in the body of some inductive rule of $P_{D,n}$. In particular, $P_{D,n} \in \mathcal{P}(P_{D,n})$. Then for each $P_{D,n}$, we define a set of *goal configurations* $\mathsf{GConf}(P_{D,n}) = \{(P_{D,n'}, n') \mid P_{D,n'} \in \mathcal{P}(P_{D,n})\}$.

Proposition 4. *For each predicate $P_{D,n}$ and $m \in \mathbb{N}$, $\mathsf{lfp}(P_{D,n})(m)$ holds iff $(P_{D,n}, m) \Rightarrow_{\mathcal{A}_{P_D}} \mathsf{GConf}(P_{D,n})$.*

Thanks to Proposition 4, we have $\mathsf{lfp}(P_{D,n}) = \mathsf{Pre}^*_{\mathcal{A}_{P_D}}(P_{D,n}, \mathsf{GConf}(P_{D,n}))$. According to Proposition 1, for each predicate $P_{D,n}$, a quantifier-free Presburger formula $\varphi_{P_{D,n}}(\alpha)$ in disjunctive normal form to represent $\mathsf{lfp}(P_{D,n})$, can be computed in doubly exponential time w.r.t. the size of $\mathcal{A}_{P_D}$ (thus in doubly exponential time w.r.t. the size of the inductive definition of P_D as well). In addition, if the constants in the inductive definition of P_D are encoded in unary, then the complexity is dropped to singly exponential time.

Computation of $\mathsf{lfp}(P_D)$. The main idea is to reduce the computation of $\mathsf{lfp}(P_D)$ to solving the reachability problem of an N1CS.

From the previous step, the solution of $P_{D,n}(\gamma)$ is expressed by the formula $\varphi_{P_{D,n}}(\gamma)$ in disjunctive normal form, say $\varphi_{P_{D,n}}(\gamma) = \bigvee\limits_{1 \leqslant i \leqslant \ell_n} \varphi^{(i)}_{P_{D,n}}(\gamma)$, where each $\varphi^{(i)}_{P_{D,n}}(\gamma)$ is of the form $\gamma = n_1$ or $\gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod n_2$. Let $N \in \mathbb{N}$ be the least common multiplier of the divisors n_2 occurring in $\varphi_{P_{D,n}}(\alpha)$ for $n \in \mathcal{N}(P_D)$.

It follows that $P_D(\alpha; \beta) ::= \exists \delta \exists \gamma.\ \Delta_{\mathbb{N}} \wedge P_D(\delta; \beta) \wedge P_D(\gamma; n) \equiv \bigvee\limits_{1 \leqslant i \leqslant \ell_n} \exists \delta \exists \gamma.\ \Delta_{\mathbb{N}} \wedge \varphi^{(i)}_{P_{D,n}}(\gamma) \wedge P_D(\delta; \beta)$. Namely, it suffices to consider $P_D(\alpha; \beta)$ with multiple rules of the form

$$P_D(\alpha; \beta) ::= \exists \delta \exists \gamma.\ (\Delta_{\mathbb{N}} \wedge \varphi^{(i)}_{P_{D,n}}(\gamma)) \wedge P_D(\delta; \beta), \tag{3}$$

for $1 \leqslant i \leq \ell_n$, where each $\varphi^{(i)}_{P_{D,n}}(\gamma)$ is of the form $\gamma = n_1$ or $\gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod N$. This new collection of rules is *linear* in that the predicate P_D occurs at most once in the body of each rule, which is simpler than (2).

$\mathsf{lfp}(P_D)$ can now be computed by appealing to an N1CS $\mathcal{B}_{P_D} = (Q', \Theta')$. The N1CS $\mathcal{B}_{P_D}$ is constructed according to the new collection of rules of P_D. The states of $\mathcal{B}_{P_D}$ are of the form (q, r), where q is a location and $r \in \{0, \ldots, N-1\}$. In $\mathcal{B}_{P_D}$, a special location q_0 is used to represent the predicate P_D.

Let us fix an inductive rule of $P_D(\alpha; \beta)$, say

$$P_D(\alpha; \beta) ::= \exists \delta \exists \gamma.\ (\Delta_{\mathbb{N}} \wedge \varphi^{(i)}_{P_{D,n}}(\gamma)) \wedge P_D(\delta; \beta). \tag{4}$$

We will demonstrate how to construct the transition rules of $\mathcal{B}_{P_D}$ according to this rule. We will only illustrate the construction for the case that each $\varphi^{(i)}_{P_{D,n}}(\gamma)$ is of the form $\gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod N$. The construction for the case $\gamma = n_1$ is (much) simpler.

For (4), as before by **C4**, $\Delta_{\mathbb{N}}$ must be of the form $\alpha = \delta + m \wedge \Delta'$ or $\alpha = \gamma + m \wedge \Delta'$ such that α does *not* occur in Δ'. We will illustrate the construction by considering the former case, that is, $\alpha = \delta + m \wedge \Delta'$.

Since $\delta = \alpha - m$, we can assume that Δ' is a formula involving only α, γ (instead of δ, γ). As before, Δ' can be written as $\Delta'_1(\alpha) \wedge \Delta'_2(\alpha, \gamma) \wedge \Delta'_3(\gamma)$. Therefore,

$$\Delta' \wedge \varphi^{(i)}_{P_{D,n}}(\gamma) = \Delta'_1(\alpha) \wedge \Delta'_2(\alpha, \gamma) \wedge (\Delta'_3(\gamma) \wedge \gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod N).$$

For each $r \in \{0, \ldots, N-1\}$, Θ' includes the transition rules defined below. Let us assume that the formula $\Delta'_3(\gamma) \wedge \gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod N$ is satisfiable (since otherwise, no transition rules should be included into Θ' in this case).

- The transition rules for Δ'_1:
  - if $\Delta'_1 = \texttt{true}$, then $(q_0, r) \hookrightarrow (+0, (q_1, r))$,
  - if $\Delta'_1 = \alpha \geqslant l$, then $(q_0, r) \hookrightarrow (\geqslant l, (q_1, r))$,
  - if $\Delta'_1 = \alpha \leqslant u$, then $(q_0, r) \hookrightarrow (\leqslant u, (q_1, r))$,
  - if $\Delta'_1 = l \leqslant \alpha \leqslant u$, then $(q_0, r) \hookrightarrow (\geqslant l, (q'_1, r))$, $(q'_1, r) \hookrightarrow (\leqslant u, (q_1, r))$;
- the transition rules for

$$\Delta'' = \Delta'_2(\alpha, \gamma) \wedge (\Delta'_3(\gamma) \wedge \gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod N):$$

  - if $\Delta'_2 = \texttt{true}$, then $(q_1, r) \hookrightarrow (+0, (q_2, r))$, since $\Delta'_3(\gamma) \wedge \gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod N$ is satisfiable (by assumption),
  - if $\Delta'_2 = \gamma \geqslant \alpha + l$, then
    * if $\Delta'_3 = \texttt{true}$ or $\Delta'_3 = \gamma \geqslant l'$, then

    $$\exists \gamma.\ \Delta'' = \exists \gamma.\ \gamma \geqslant \alpha + l \wedge \Delta'_3 \wedge \gamma \geqslant n_1 \wedge \gamma \equiv n_3 \bmod N$$

    is satisfiable for every value of α, therefore, we have $(q_1, r) \hookrightarrow (+0, (q_2, r))$,
    * if $\Delta'_3 = \gamma \leqslant u'$ or $\Delta'_3 = l' \leqslant \gamma \leqslant u'$, let $l'' = n_1$ or $l'' = \max(l', n_1)$ respectively, then

    $$\exists \gamma.\ \Delta'' = \exists \gamma.\ \gamma \geqslant \alpha + l \wedge l'' \leqslant \gamma \leqslant u' \wedge \gamma \equiv n_3 \bmod N,$$

    from this, we have that for each $s \in \mathbb{N}$ such that $l'' \leqslant s \leqslant u'$ and $s \equiv n_3 \bmod N$, $(q_1, r) \hookrightarrow (\leqslant s - l, (q_2, r))$,
  - if $\Delta'_2 = \gamma \leqslant \alpha + u$,
    * if $\Delta'_3 = \texttt{true}$ or $\Delta'_3 = \gamma \geqslant l'$, let $l'' = n_1$ or $l'' = \max(l', n_1)$ respectively, then

    $$\exists \gamma.\ \Delta'' = \exists \gamma.\ \gamma \leqslant \alpha + u \wedge \gamma \geqslant l'' \wedge \gamma \equiv n_3 \bmod N,$$

    which is equivalent to $\alpha + u \geqslant l'' + s$, where s is the minimum natural number satisfying $0 \leqslant s < N$ and $l'' + s \equiv n_3 \bmod N$, therefore, we have $(q_1, r) \hookrightarrow (\geqslant l'' + s - u, (q_2, r))$,

$*$ if $\Delta'_3 = \gamma \leqslant u'$ or $\Delta'_3 = l' \leqslant \gamma \leqslant u'$, let $l'' = n_1$ or $l'' = \max(l', n_1)$ respectively, then

$$\exists \gamma.\ \Delta'' = \exists \gamma.\ \gamma \leqslant \alpha + u \wedge l'' \leqslant \gamma \leqslant u' \wedge \gamma \equiv n_3 \bmod N,$$

from this, we have that for each $s \in \mathbb{N}$ such that $l'' \leqslant s \leqslant u'$ and $s \equiv n_3 \bmod N$, $(q_1, r) \hookrightarrow (\geqslant s - u, (q_2, r))$,

- if $\Delta'_2 = \alpha + l \leqslant \gamma \leqslant \alpha + u$, then

$*$ if $\Delta'_3 = \texttt{true}$ or $\Delta'_3 = \gamma \geqslant l'$, let $l'' = n_1$ or $l'' = \max(l', n_1)$ respectively, then

$$\exists \gamma.\ \Delta'' = \exists \gamma.\ \alpha + l \leqslant \gamma \leqslant \alpha + u \wedge \gamma \geqslant l'' \wedge \gamma \equiv n_3 \bmod N,$$

which is equivalent to $\alpha + s \geqslant l''$, provided that $\alpha \equiv r \bmod N$, where s is the maximum natural number such that $l \leqslant s \leqslant u$ and $r + s \equiv n_3 \bmod N$, therefore, we have $(q_1, r) \hookrightarrow (\geqslant l'' - s, (q_2, r))$,

$*$ if $\Delta'_3 = \gamma \leqslant u'$ or $\Delta'_3 = l' \leqslant \gamma \leqslant u'$, let $l'' = n_1$ or $l'' = \max(l', n_1)$ respectively, then

$$\exists \gamma.\ \Delta'' = \exists \gamma.\ \alpha + l \leqslant \gamma \leqslant \alpha + u \wedge l'' \leqslant \gamma \leqslant u' \wedge \gamma \equiv n_3 \bmod N,$$

from this, we have that for each $s \in \mathbb{N}$ such that $l'' \leqslant s \leqslant u'$ and $s \equiv n_3 \bmod N$, $(q_1, r) \hookrightarrow (\leqslant s - l, (q'_2, r))$ and $(q'_2, r) \hookrightarrow (\geqslant s - u, (q_2, r))$,

- the transition rule for $\alpha = \delta + m$: $(q_2, r) \hookrightarrow (-m, (q_0, (r - m) \bmod N))$,

where q_1, q_2, q'_1, q'_2 are the freshly introduced locations.

We have the following result:

Proposition 5. *For $m, n \in \mathbb{N}$, let $r = m \bmod N$ and $r' = n \bmod N$. Then $\mathsf{lfp}(P_D)(m, n)$ holds iff $((q_0, r), m) \Rightarrow_{\mathcal{B}_{P_D}} ((q_0, r'), n)$.*

From Proposition 1, for each $r, r' \in \{0, \ldots, N - 1\}$, a quantifier-free Presburger formula $\varphi_{(q_0,r),(q_0,r')}(\alpha, \beta)$ can be computed in triply exponential time w.r.t. the size of $\mathcal{B}_{P_D}$ to represent $\{(m, n) \in \mathbb{N}^2 \mid ((q_0, r), m) \Rightarrow_{\mathcal{B}_{P_D}} ((q_0, r'), n)\}$. Therefore, from Proposition 5, $\mathsf{lfp}(P_D)$ can be expressed with $\varphi_{P_D}(\alpha, \beta) \equiv \bigvee\limits_{0 \leqslant r, r' < N} \varphi_{(q_0,r),(q_0,r')}(\alpha, \beta)$. Since the size of the new collection of inductive rules of P_D—thus the size of $\mathcal{B}_{P_D}$—is at most doubly exponential in the size of the (original) inductive definition of P_D, we conclude that the size of $\varphi_{P_D}(\alpha, \beta)$ is 5-fold exponential in the size of the (original) inductive definition of P_D. In addition, the size of $\varphi_{P_D}(\alpha, \beta)$ is 4-fold exponential if the constants in the inductive definition of P_D are encoded in unary.

4.3 The General Case

In the subsection, we show how to combine the techniques developed in the preceding sections to tackle the general case. Without loss of generality, we assume that the data predicate $P_D(\boldsymbol{\alpha}, \boldsymbol{\beta})$ satisfies that $|\boldsymbol{\alpha}| = k > 1$, $\boldsymbol{\alpha}(1), \cdots, \boldsymbol{\alpha}(k-1)$

are of type $\mathbb{D}$, and $\boldsymbol{\alpha}(k)$ is of type $\mathbb{N}$. For convenience, we write $\boldsymbol{\alpha} = (\boldsymbol{\alpha}', \alpha'')$ where $\boldsymbol{\alpha}' = (\boldsymbol{\alpha}(1), \ldots, \boldsymbol{\alpha}(k-1))$ and $\alpha'' = \boldsymbol{\alpha}(k)$. Similarly, $\boldsymbol{\beta} = (\boldsymbol{\beta}', \beta'')$. Then each inductive rule for P_D is of the form

$P_D(\boldsymbol{\alpha}', \alpha''; \boldsymbol{\beta}', \beta'') ::= \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \Delta_{\mathbb{D}} \wedge \Delta_{\mathbb{N}} \wedge P_D(\boldsymbol{\delta}', \delta''; \boldsymbol{\beta}', \beta'') \wedge P_D(\boldsymbol{\gamma}', \gamma''; \boldsymbol{\epsilon}', n)$. We split each inductive rule of P_D into two rules,

$$P_{D,\mathbb{D}}(\boldsymbol{\alpha}'; \boldsymbol{\beta}') ::= \exists \boldsymbol{x}.\ \Delta_{\mathbb{D}} \wedge P_{D,\mathbb{D}}(\boldsymbol{\delta}'; \boldsymbol{\beta}') \wedge P_{D,\mathbb{D}}(\boldsymbol{\gamma}'; \boldsymbol{\epsilon}'),$$

$$P_{D,\mathbb{N}}(\alpha''; \beta'') ::= \exists \boldsymbol{h}.\ \Delta_{\mathbb{N}} \wedge P_{D,\mathbb{N}}(\delta''; \beta'') \wedge P_{D,\mathbb{N}}(\gamma''; n).$$

The computation of $\mathsf{lfp}(P_D)$ proceeds as follows. Intuitively, we first deal with $P_{D,\mathbb{D}}(\boldsymbol{\alpha}'; \boldsymbol{\beta}')$ and $P_{D,\mathbb{N}}(\alpha''; \beta'')$ separately by the constructions in Sects. 4.1 and 4.2. More specifically, $\mathsf{lfp}(\mathcal{T}_{P_{D,\mathbb{D}}})$, a set of order graphs on V, is computed, and the A1CS $\mathcal{A}_{P_{D,\mathbb{N}}}$ and the N1CS $\mathcal{B}_{P_{D,\mathbb{N}}}$ are constructed. We then integrate the order graphs from $\mathsf{lfp}(\mathcal{T}_{P_{D,\mathbb{D}}})$ into the states of $\mathcal{A}_{P_{D,\mathbb{N}}}$ and $\mathcal{B}_{P_{D,\mathbb{N}}}$.

As the first step, we use the algorithm in Sect. 4.1 to compute $\mathsf{lfp}(\mathcal{T}_{P_{D,\mathbb{D}}})$. As a result, we obtain a set of order graphs over $V = \boldsymbol{\alpha}' \cup \boldsymbol{\beta}' \cup \mathcal{C}(P_{D,\mathbb{D}})$, where $\mathcal{C}(P_{D,\mathbb{D}})$ is the set of constants occurring in the body of the rules of $P_{D,\mathbb{D}}(\boldsymbol{\alpha}'; \boldsymbol{\beta}')$.

Suppose $\mathcal{A}_{P_{D,\mathbb{N}}} = (Q, \Theta)$ is the A1CS constructed for $P_{D,\mathbb{N}}(\alpha''; \beta'')$ as in Sect. 4.2. Recall that Q is the union of $\{P_{D,\mathbb{N},n} \mid n \in \mathcal{N}(P_{D,\mathbb{N}})\}$ and a set of auxiliary states. We shall construct a new A1CS $\mathcal{A}'_{P_D}$. The state space of $\mathcal{A}'_{P_D}$ is $\mathsf{lfp}(\mathcal{T}_{P_{D,\mathbb{D}}}) \times Q$. As before, for each $n \in \mathcal{N}(P_{D,\mathbb{N}})$, we consider a predicate $P_{D,n}(\boldsymbol{\alpha}', \alpha''; \boldsymbol{\beta}')$ whose inductive definition is obtained from that of $P_D(\boldsymbol{\alpha}', \alpha''; \boldsymbol{\beta}', \beta'')$ by replacing β'' with n. Specifically, each inductive rule of $P_{D,n}$ is of the form,

$$P_{D,n}(\boldsymbol{\alpha}', \alpha''; \boldsymbol{\beta}') ::= \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \Delta_{\mathbb{D}} \wedge \Delta_{\mathbb{N}} \wedge P_{D,n}(\boldsymbol{\delta}', \delta''; \boldsymbol{\beta}') \wedge P_{D,n'}(\boldsymbol{\gamma}', \gamma''; \boldsymbol{\epsilon}'). \quad (5)$$

Considering the inductive rule of $P_{D,\mathbb{N},n}(\alpha'')$ corresponding to (5),

$$P_{D,\mathbb{N},n}(\alpha'') ::= \exists \boldsymbol{h}.\ \Delta_{\mathbb{N}} \wedge P_{D,\mathbb{N},n}(\delta'') \wedge P_{D,\mathbb{N},n'}(\gamma''). \quad (6)$$

We lift the transition rules of $\mathcal{A}_{P_{D,\mathbb{N}}}$ for the inductive rule (6) of $P_{D,\mathbb{N},n}(\alpha'')$ to the ones of $\mathcal{A}'_{P_D}$ for the rule (5) of $P_{D,n}(\boldsymbol{\alpha}', \alpha''; \boldsymbol{\beta}')$ as follows: For every $G, G_1, G_2 \in \mathsf{lfp}(\mathcal{T}_{P_{D,\mathbb{D}}})$ satisfying the proper constraints induced by some inductive rule of $P_{D,\mathbb{D}}$, add G, G_1, G_2 as the first-component of states. For instance, the transitions $P_{D,\mathbb{N},n} \hookrightarrow (+0, q_1)$, $q_1 \hookrightarrow \{(-m, P_{D,\mathbb{N},n}), (\mathsf{reset}(0), q_2')\}$, $q_2' \hookrightarrow \{(+1, q_2')\}$, $q_2' \hookrightarrow \{(+0, q_2)\}$, and $q_2 \hookrightarrow \{(+0, P_{D,\mathbb{N},n'})\}$ in $\mathcal{A}_{P_{D,\mathbb{N}}}$ are changed to the following transitions in $\mathcal{A}'_{P_D}$ respectively: $(G, P_{D,\mathbb{N},n}) \hookrightarrow (+0, (G, q_1))$, $(G, q_1) \hookrightarrow \{(-m, (G_1, P_{D,\mathbb{N},n})), (\mathsf{reset}(0), (G, q_2'))\}$, $(G, q_2') \hookrightarrow \{(+1, (G, q_2'))\}$, $(G, q_2') \hookrightarrow \{(+0, (G, q_2))\}$, and $(G, q_2) \hookrightarrow \{(+0, (G_2, P_{D,\mathbb{N},n'}))\}$.

Recall that $\mathcal{P}(P_{D,\mathbb{N},n})$ is the set of predicates $P_{D,\mathbb{N},n'}$ occurring in the body of the inductive rules of $P_{D,\mathbb{N},n}$. Let $\mathsf{GConf}'(P_{D,n}) = \{((G_0, P_{D,\mathbb{N},n'}), n') \mid P_{D,\mathbb{N},n'} \in \mathcal{P}(P_{D,\mathbb{N},n})\}$, where G_0 is the order graph corresponding to the value formula $\boldsymbol{\alpha}' = \boldsymbol{\beta}'$. Again, from Proposition 1, for each state $(G, P_{D,\mathbb{N},n})$ of $\mathcal{A}'_{P_D}$, a quantifier-free Presburger formula $\varphi_{(G, P_{D,\mathbb{N},n})}$ can be computed to represent the set of natural numbers $\mathsf{Pre}^*_{\mathcal{A}'_{P_D}}((G, P_{D,\mathbb{N},n}), \mathsf{GConf}'(P_{D,n}))$. As a result, $\mathsf{lfp}(P_{D,n})$ is given by

$$\varphi_{P_{D,n}}(\boldsymbol{\alpha}', \alpha''; \boldsymbol{\beta}') = \bigvee_{G \in \mathsf{lfp}(\mathcal{T}_{P_{D,\mathbb{D}}})} (\varDelta(G) \wedge \varphi_{(G, P_{D,\mathbb{N}}, n)}).$$

Next, we replace each predicate atom $P_D(\boldsymbol{\gamma}', \gamma''; \boldsymbol{\epsilon}', n)$ in the body of each inductive rule by the formula $\varphi_{P_{D,n}}(\boldsymbol{\gamma}', \gamma''; \boldsymbol{\epsilon}')$ and rewrite $\varphi_{P_{D,n}}(\boldsymbol{\gamma}', \gamma''; \boldsymbol{\epsilon}')$ into a disjunctive normal form, resulting into a new collection of *linear* inductive rules for $P_D(\boldsymbol{\alpha}', \alpha''; \boldsymbol{\beta}', \beta'')$.

We can then define the N1CS $\mathcal{B}'_{P_D}$ by adapting the construction of the N1CS $\mathcal{B}_{P_{D,\mathbb{N}}}$ for $P_{D,\mathbb{N}}$. Roughly speaking, this is done by adding the order graphs as components of the states of $\mathcal{B}_{P_{D,\mathbb{N}}}$. Finally, a linear arithmetic formula $\varphi_{P_D}(\boldsymbol{\alpha}; \boldsymbol{\beta})$, which is a mixture of dense order constraints and quantifier-free Presburger formulae, is computed from $\mathcal{B}'_{P_D}$ to represent $\mathsf{lfp}(P_D)$, by using Proposition 1.

5 Satisfiability

Let $\varphi = \varPi \wedge \varDelta \wedge \varSigma$ be a $\mathsf{CSLTP}[P]$ formula. Suppose $\varSigma = a_1 * \cdots * a_n$, where each a_i is either a points-to atom or a predicate atom. Let $P_D(\boldsymbol{\alpha}; \boldsymbol{\beta})$ be the data predicate induced by P and $\varphi_{P_D}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ be the formula constructed in Sect. 4 to represent $\mathsf{lfp}(P_D)$. For each inductive rule R of $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta})$, we define $\varDelta_R^{\geqslant 1}(\boldsymbol{\alpha}; \boldsymbol{\beta})$ as follows.

- If R is a left-hole inductive rule

$$\begin{aligned} P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}) ::= \exists X \exists Y \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \varDelta \wedge E \mapsto ((\texttt{left}, X), (\texttt{right}, Y), \rho_{\mathsf{d}}) * \\ P(X, \boldsymbol{\delta}; F, \boldsymbol{\beta}) * P(Y, \boldsymbol{\gamma}; \mathsf{nil}, \boldsymbol{\epsilon}), \end{aligned}$$

  then $\varDelta_R^{\geqslant 1}(\boldsymbol{\alpha}; \boldsymbol{\beta}) := \exists \boldsymbol{x} \exists \boldsymbol{h}.\ \varDelta \wedge \varphi_{P_D}[\boldsymbol{\delta}/\boldsymbol{\alpha}] \wedge \varphi_{P_D}[(\boldsymbol{\gamma}, \boldsymbol{\epsilon})/(\boldsymbol{\alpha}, \boldsymbol{\beta})]$.
- If R is a right-hole inductive rule, then $\varDelta_R^{\geqslant 1}(\boldsymbol{\alpha}; \boldsymbol{\beta})$ is defined similarly.

In addition, we define $\varDelta_P^{\geqslant 1}(\boldsymbol{\alpha}; \boldsymbol{\beta}) := \bigvee_{R:\text{ inductive rule of } P} \varDelta_R^{\geqslant 1}(\boldsymbol{\alpha}; \boldsymbol{\beta})$.

For each predicate atom $a_i = P(Z_1, \boldsymbol{\mu}; Z_2, \boldsymbol{\nu})$, we define the formula $\mathsf{Ufld}^{\geqslant 1}(a_i)$ as $\varDelta_P^{\geqslant 1}(\boldsymbol{\mu}, \boldsymbol{\nu})$. Intuitively, $\mathsf{Ufld}^{\geqslant 1}(a_i)$ is the data constraint obtained by unfolding a_i at least once (with the inductive rules of P).

For each location variable E and atom a_i in $\varSigma$, we introduce a Boolean variable $[E, i]$ to represent whether E is allocated in a_i. Let $\mathsf{BVars}(\varphi)$ denote the set of introduced Boolean variables. We define *an abstraction of* φ [12,14] to be $\mathsf{Abs}(\varphi) ::= \varPi \wedge \varDelta \wedge \phi_\varSigma \wedge \phi_*$ over $\mathsf{BVars}(\varphi) \cup \mathsf{Vars}(\varphi)$, where

- $\phi_\varSigma = \bigwedge_{1 \leqslant i \leqslant n} \mathsf{Abs}(a_i)$ is an abstraction of $\varSigma$ where
  - if $a_i = E \mapsto \rho$, then $\mathsf{Abs}(a_i) = [E, i] \wedge E \neq \mathsf{nil}$,
  - if $a_i = P(Z_1, \boldsymbol{\mu}; Z_2, \boldsymbol{\nu})$, then

$$\mathsf{Abs}(a_i) = (\neg[Z_1, i] \wedge Z_1 = Z_2 \wedge \boldsymbol{\mu} = \boldsymbol{\nu}) \vee ([Z_1, i] \wedge Z_1 \neq \mathsf{nil} \wedge \mathsf{Ufld}^{\geqslant 1}(a_i)).$$

- ϕ_* states the separation constraint of spatial atoms,

$$\phi_* = \bigwedge_{[Z_1,i],[Z'_1,j] \in \mathsf{BVars}(\varphi), i \neq j} (Z_1 = Z'_1 \wedge [Z_1,i]) \rightarrow \neg [Z'_1,j].$$

Proposition 6. *For* $\mathsf{CSLTP}[P]$ *formula* φ*,* φ *and* $\mathsf{Abs}(\varphi)$ *are equisatisfiable.*

The formula $\mathsf{Abs}(\varphi)$ can be turned into a quantifier-free formula $\mathsf{Abs}_{\mathsf{qf}}(\varphi)$ by removing all the existential quantifiers in $\mathsf{Ufld}^{\geqslant 1}(a_i)$ and replace the existentially quantified variables with some freshly introduced variables. The formula $\mathsf{Abs}_{\mathsf{qf}}(\varphi)$ can be seen as a mixed real and integer linear arithmetic constraint, thus its satisfiability can be decided in nondeterministic polynomial time in theory, and can be solved by using the state-of-the-art SMT solvers, e.g. Z3 [34], in practice.

Theorem 3. *The satisfiability of* $\mathsf{CSLTP}[P]$ *formulae can be decided in 6-fold exponential time. In addition, if the natural-number constants in* P *are encoded in unary, the satisfiability can be decided in 5-fold exponential time.*

Remark 1. The decision procedure for the satisfiability problem can be easily generalised to n-ary trees, and to separation logic formulae where several inductive predicates, e.g., $lseg(E;F)$ and $bsth(E,x,y;F,x',y')$, occur simultaneously.

6 Conclusion

In this paper, we proposed CSLTP, the compositional separation logic with tree predicates. We gave a complete decision procedure for the satisfiability problem. To our best knowledge, this is one of the most expressive fragments of SLID with data/size constraints that is equipped with a *complete* decision procedure. The main ingredient of the decision procedure is to compute the least fixed point of data predicates involving dense order constraints and difference-bound size constraints, by utilising an automata-theoretical approach.

For the future work, the decision procedure for the satisfiability problem paves the way towards a compete decision procedure for the entailment problem of CSLTP. In addition, we plan to implement the decision procedure and apply it to the analysis and verification of programs manipulating tree data structures.

References

1. Abdulla, P.A., Holík, L., Jonsson, B., Lengál, O., Trinh, C.Q., Vojnar, T.: Verification of heap manipulating programs with ordered data by extended forest automata. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 224–239. Springer, Cham (2013). doi:10.1007/978-3-319-02444-8_17
2. Berdine, J., Calcagno, C., O'Hearn, P.W.: Symbolic execution with separation logic. In: Yi, K. (ed.) APLAS 2005. LNCS, vol. 3780, pp. 52–68. Springer, Heidelberg (2005). doi:10.1007/11575467_5

3. Bouajjani, A., Drăgoi, C., Enea, C., Sighireanu, M.: Accurate invariant checking for programs manipulating lists and arrays with infinite data. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, pp. 167–182. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33386-6_14
4. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997). doi:10.1007/3-540-63141-0_10
5. Brotherston, J., Distefano, D., Petersen, R.L.: Automated cyclic entailment proofs in separation logic. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 131–146. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22438-6_12
6. Brotherston, J., Fuhs, C., Perez, J.A.N., Gorogiannis, N.: A decision procedure for satisfiability in separation logic with inductive predicates. In: LICS, pp. 25:1–25:10 (2014)
7. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (1981)
8. Chin, W.-N., David, C., Nguyen, H.H., Qin, S.: Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Sci. Comput. Program. **77**(9), 1006–1036 (2012)
9. Chu, D.-H., Jaffar, J., Trinh, M.-T.: Automatic induction proofs of data-structures in imperative programs. In: PLDI, pp. 457–466 (2015)
10. Comon-Lundh, H., Jacquemard, F., Perrin, N.: Visibly tree automata with memory and constraints. Logical Methods Comput. Sci. **4**(2), 1–36 (2008)
11. Creus, C., Godoy, G.: Tree automata with height constraints between brothers. In: RTA-TLCA, pp. 149–163 (2014)
12. Enea, C., Lengál, O., Sighireanu, M., Vojnar, T.: Compositional entailment checking for a fragment of separation logic. In: Garrigue, J. (ed.) APLAS 2014. LNCS, vol. 8858, pp. 314–333. Springer, Cham (2014). doi:10.1007/978-3-319-12736-1_17
13. Enea, C., Sighireanu, M., Wu, Z.: On automated lemma generation for separation logic with inductive definitions. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 80–96. Springer, Cham (2015). doi:10.1007/978-3-319-24953-7_7
14. Gu, X., Chen, T., Wu, Z.: A complete decision procedure for linearly compositional separation logic with data constraints. In: IJCAR, pp. 532–549 (2016)
15. Haase, C., Kreutzer, S., Ouaknine, J., Worrell, J.: Reachability in succinct and parametric one-counter automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 369–383. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04081-8_25
16. Habermehl, P., Iosif, R., Vojnar, T.: Automata-based verification of programs with tree updates. Acta Inf. **47**(1), 1–31 (2010)
17. Hóu, Z., Goré, R., Tiu, A.: Automated theorem proving for assertions in separation logic with all connectives. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS, vol. 9195, pp. 501–516. Springer, Cham (2015). doi:10.1007/978-3-319-21401-6_34
18. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. In: Bonacina, M.P. (ed.) CADE 2013. LNCS, vol. 7898, pp. 21–38. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38574-2_2
19. Iosif, R., Rogalewicz, A., Vojnar, T.: Deciding entailments in inductive separation logic with tree automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 201–218. Springer, Cham (2014). doi:10.1007/978-3-319-11936-6_15

20. Le, Q.L., Sun, J., Chin, W.-N.: Satisfiability modulo heap-based programs. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 382–404. Springer, Cham (2016). doi:10.1007/978-3-319-41528-4_21
21. Manna, Z., Sipma, H.B., Zhang, T.: Verifying balanced trees. In: Artemov, S.N., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 363–378. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72734-7_26
22. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, pp. 1–19. Springer, Heidelberg (2001). doi:10.1007/3-540-44802-0_1
23. Pek, E., Qiu, X., Madhusudan, P.: Natural proofs for data structure manipulation in C using separation logic. In: PLDI, pp. 440–451 (2014)
24. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic using SMT. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 773–789. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_54
25. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic with trees and data. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 711–728. Springer, Cham (2014). doi:10.1007/978-3-319-08867-9_47
26. Piskac, R., Wies, T., Zufferey, D.: GRASShopper - complete heap verification with mixed specifications. In: TACAS, pp. 124–139 (2014)
27. Qiu, X., Garg, P., Stefănescu, A., Madhusudan, P.: Natural proofs for structure, data, and separation. In: PLDI, pp. 231–242 (2013)
28. Revesz, P.Z.: A closed-form evaluation for datalog queries with integer (gap)-order constraints. Theor. Comput. Sci. **116**(1), 117–149 (1993)
29. Reynolds, A., Iosif, R., Serban, C., King, T.: A decision procedure for separation logic in SMT. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 244–261. Springer, Cham (2016). doi:10.1007/978-3-319-46520-3_16
30. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS, pp. 55–74 (2002)
31. Rümmer, P., Hojjat, H., Kuncak, V.: Disjunctive interpolants for horn-clause verification. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 347–363. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_24
32. Seidl, H., Schwentick, T., Muscholl, A., Habermehl, P.: Counting in trees for free. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 1136–1149. Springer, Heidelberg (2004). doi:10.1007/978-3-540-27836-8_94
33. Tatsuta, M., Le, Q.L., Chin, W.-N.: Decision procedure for separation logic with inductive definitions and presburger arithmetic. In: Igarashi, A. (ed.) APLAS 2016. LNCS, vol. 10017, pp. 423–443. Springer, Cham (2016). doi:10.1007/978-3-319-47958-3_22
34. Z3. http://rise4fun.com/z3