Separation Logic with Linearly Compositional Inductive Predicates and Set Data Constraints

Chong Gao[1,2], Taolue Chen[3], 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 and Information Systems, Birkbeck, University of London, London, UK

Abstract. We identify difference-bound set constraints (DBS), an analogy of difference-bound arithmetic constraints for sets. DBS can express not only set constraints but also arithmetic constraints over set elements. We integrate DBS into separation logic with linearly compositional inductive predicates, obtaining a logic thereof where set data constraints of linear data structures can be specified. We show that the satisfiability of this logic is decidable. A crucial step of the decision procedure is to compute the transitive closure of DBS-definable set relations, to capture which we propose an extension of quantified set constraints with Presburger Arithmetic (RQSPA). The satisfiability of RQSPA is then shown to be decidable by harnessing advanced automata-theoretic techniques.

1 Introduction

Separation Logic (SL) is a well-established approach for deductive verification of programs that manipulate dynamic data structures [25,28]. Typically, SL is used in combination with inductive definitions (SLID), which provides a natural and convenient means to specify dynamic data structures. To reason about the property (e.g. sortedness) of data values stored in data structures, it is also necessary to incorporate data constraints into the inductive definitions.

One of the most fundamental questions for a logical theory is whether its satisfiability is decidable. SLID with data constraints is no exception. This problem becomes more challenging than one would probably expect, partially due to the inherent intricacy brought up by inductive definitions and data constraints. It is somewhat surprising that only disproportional research has addressed this question (cf. *Related work*). In practice, most available tools based on SLID only support heuristics without giving completeness guarantees, especially when data

Partially supported by the NSFC grants (No. 61472474, 61572478, 61872340), UK EPSRC grant (EP/P00430X/1), and the INRIA-CAS joint research project VIP.

B. Catania et al. (Eds.): SOFSEM 2019, LNCS 11376, pp. 206–220, 2019.
https://doi.org/10.1007/978-3-030-10801-4_17

constraints are involved. *Complete* decision procedures for satisfiability, however, have been found important in software engineering tasks such as symbolic execution, specification debugging, counterexample generation, etc., let along the theoretical insights they usually shed on the logic system.

The dearth of complete decision procedures for SLID with data constraints has prompted us to launch a research program as of 2015, aiming to identify decidable *and* sufficiently expressive instances. We have made encouraging progress insofar. In [15], we set up a general framework, but could only tackle linear data structures with data constraints in difference-bound arithmetic. In [34], we were able to tackle tree data structures by exploiting machineries such as order graphs and counter machines, though the data constraints therein remained to be in difference-bound arithmetic.

An important class of data constraints that is currently elusive in our investigations is set constraints. They are mandatory for reasoning about, e.g., invariants of data collections stored in data structures. For instance, when specifying the correctness of a sorting algorithm on input lists, whilst the sortedness of the list can be described by difference-bound arithmetic constraints, the property that the sorting algorithm does not change the set of data values on the list requires inductive definitions with *set* data constraints. Indeed, reviewers of the papers [15,34] constantly raised the challenge of set constraints, which compelled us to write the current paper.

Main Contributions. Our *first* contribution is to carefully design the difference-bound set constraints ($\mathcal{DBS}$), and to integrate them into the linearly compositional inductive predicates introduced in [15], yielding $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$: *SL with linearly compositional inductive predicates and set data constraints*. The rationale of $\mathcal{DBS}$ is two-fold: (1) it must be sufficiently expressive to represent common set data constraints as well as arithmetic constraints over set elements one usually needs when specifying linear data structures, (2) because of the inductive predicates, it must be sufficiently "simple" to be able to capture the *transitive closure* of $\mathcal{DBS}$-definable set relations[1] in an effective means, in order to render the satisfiability of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$ decidable. As the *second* contribution, we show that the transitive closure of $\mathcal{DBS}$ can indeed be captured in the *restricted extension of quantified set constraints with Presburger arithmetic* ($\mathcal{RQSPA}$) introduced in this paper. Finally, our *third* contribution is to show that the satisfiability of $\mathcal{RQSPA}$ is decidable by establishing a connection of $\mathcal{RQSPA}$ with Presburger automata [29]. This extends the well-known connection of Monadic Second-Order logic on words (MSOW) and finite-state automata *a la* Büchi and Elgot [5,11]. These contributions, together with a procedure which constructs an abstraction (as an $\mathcal{RQSPA}$ formula) from a given $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$ formula and which we adapt from our previous work [15], show the satisfiability of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$ is decidable.

We remark that sets are conceptually related to second—rather than first—order logics. While the transitive closure of logic formulae with *first-order variables* is somehow well-studied (especially for simple arithmetic; cf. *Related Work*), the transitive closure of logic formulae with *second-order variables* is

[1] This shall be usually referred to as "transitive closure of $\mathcal{DBS}$" to avoid clumsiness.

rarely addressed in literature. (They easily lead to undecidability.) To our best knowledge, the computation of transitive closures of $\mathcal{DBS}$ here represents one of the first practically relevant examples of the computation of this type for a class of logic formulae with second-order variables, which may be of independent interests.

Related Work. We first review the work on SLID with data constraints. (Due to space limit, the work on SLID *without* data constraints will be skipped.) In [7,8,23], SLID with set/multiset/size data constraints were considered, but only (incomplete) heuristics were provided. To reason about invariants of data values stored in lists, SL with list segment predicates and data constraints in universally quantified Presburger arithmetic was considered [1]. The work [26,27] provided decision procedures for SLID with data constraints by translating into many-sorted first-order logic with reachability predicates. In particular, in [27, Section 6], extensions of basic logic GRIT are given to cover set data constraints as well as order constraints over set elements. However, it seems that this approach does not address arithmetic constraints over set elements (cf. the "Limitations" paragraph in the end of Sect. 6 in [27]). For instance, a list where the data values in adjacent positions are consecutive can be captured in $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$ (see the predicate *plseg* in Sect. 3), but appears to go beyond the work [26,27]. Moreover, there is no precise characterisation of the limit of extensions under which the decidability retains. The work [13] introduced the concept of compositional inductive predicates, which may alleviate the difficulties of the entailment problem for SLID. Nevertheless, [13] only provided sound heuristics rather than decision procedures. More recently, the work [21,31] investigated SLID with Presburger arithmetic data constraints.

Furthermore, several logics other than separation logic have been considered to reason about both shape properties and data constraints of data structures. The work [30] proposed a generic decision procedure for recursive algebraic data types with abstraction functions encompassing lengths (sizes) of data structures, sets or multisets of data values as special cases. Nevertheless, the work [30] focused on functional programs while this work aims to verify imperative programs, which requires to reason about *partial* data structures such as list segments (rather than complete data structures such as lists). It is unclear how the decision procedure in [30] can be generalised to partial data structures. The work [22] introduced STRAND, a fragment of monadic second-order logic, to reason about tree structures. Being undecidable in general, several decidable fragments were identified. STRAND does not provide an explicit means to describe sets of data values, although it allows using set variables to represent sets of locations.

Our work is also related to classical logics with set constraints, for which we can only give a brief (but by no means comprehensive) summary. Presburger arithmetic extended with sets was studied dating back to 80's, with highly undecidability results [6,16]. However, decidable fragments do exist: [33] studied the non-disjoint combination of theories that share set variables and set operations. [20] considered $\mathrm{QFBAPA}^{<}_{\infty}$, a quantifier-free logic of sets of real numbers supporting integer sets and variables, linear arithmetic, the cardinality operator,

infimum and supremum. [17,32] investigated two extensions of the Bernays-Schönfinkel-Ramsey fragment of first-order predicate logic (BSR) with simple linear arithmetic over integers and difference-bound constraints over reals (but crucially, the ranges of the universally quantified variables must be bounded). Since the unary predicate symbols in BSR are uninterpreted and represent sets over integers or reals, the two extensions of BSR can also be used to specify the set constraints on integers or reals. [10] presented a decision procedure for quantifier-free constraints on restricted intensional sets (i.e., sets given by a property rather than by enumerating their elements). None of these logics are able to capture the transitive closure of $\mathcal{DBS}$ as $\mathcal{RQSPA}$ does. MSOW extended with linear cardinality constraints was investigated in [18]. Roughly speaking, $\mathcal{RQSPA}$ can be considered as an extension of MSOW with linear arithmetic expressions on the maximum or minimum value of free set variables. Therefore, the two extensions in [18] and this paper are largely incomparable.

In contrast to set constraints, the computation of transitive closures of relations definable in first-order logic (in particular, difference-bound and octagonal arithmetic constraints) has been considered in for instance, [2–4,9,19].

2 Logics for Sets

We write $\mathbb{Z}$, $\mathbb{N}$ for the set of integers and natural numbers; $\mathbb{S}_\mathbb{Z}$ and $\mathbb{S}_\mathbb{N}$ for *finite* subsets of $\mathbb{Z}$ and $\mathbb{N}$. For $n \in \mathbb{N}$, $[n]$ stands for $\{1, \cdots, n\}$. We shall work exclusively on finite subsets of $\mathbb{Z}$ or $\mathbb{N}$ unless otherwise stated. For any finite $A \neq \varnothing$, we write $\min(A)$ and $\max(A)$ for the minimum and maximum element of A. These functions, however, are *not* defined over empty sets.

In the sequel, we introduce a handful of logics for sets which will be used later in this paper. We mainly consider two data types, i.e., integer type $\mathbb{Z}$ and (finite) set type $\mathbb{S}_\mathbb{Z}$. Typically, $c, c', \cdots \in \mathbb{Z}$ and $A, A', \cdots \in \mathbb{S}_\mathbb{Z}$. Accordingly, two types of variables occur: integer variables (ranged over by $x, y, \cdots$) and set variables (ranged over by $S, S', \cdots$). Furthermore, we reserve $\bowtie \in \{=, \leq, \geq\}$ for comparison operators between integers,[2] and $\asymp \in \{=, \subseteq, \supseteq, \subset, \supset\}$ for comparison operators between sets. We start with *difference-bound* set constraints ($\mathcal{DBS}$).

Definition 1 (Difference-bound set constraints). *Formulae of $\mathcal{DBS}$ are defined by the rules:*

$$\begin{array}{lll} \varphi & ::= S = S' \cup T_s \mid T_i \ \bowtie \ T_i + c \mid \varphi \wedge \varphi & \\ T_s & ::= \varnothing \mid \{\min(S)\} \mid \{\max(S)\} \mid T_s \cup T_s & (\textit{s}\text{et terms}) \\ T_i & ::= \min(S) \mid \max(S) & (\textit{i}\text{nteger terms}) \end{array}$$

Remark. $\mathcal{DBS}$ is a rather limited logic, but it has been carefully devised to serve the data formulae in inductive predicates of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$ (cf. Sect. 3). In particular, we remark that only conjunction, but not disjunction, of atomic constraints is

[2] The operators $<$ and $>$ can be seen as abbreviations, for instance, $x < y$ is equivalent to $x \leq y - 1$, which will be used later on as well.

allowed. The main reason is, once the disjunction is introduced, the computation of transitive closures becomes infeasible simply because one would be able to encode the computation of Minsky's two-counter machines. □

To capture the transitive closure of $\mathcal{DBS}$, we introduce *Restricted extension of Quantified Set constraints with Presburger Arithmetic*[3] ($\mathcal{RQSPA}$). Intuitively, an $\mathcal{RQSPA}$ formula is a *quantified* set constraint extended with Presburger Arithmetic satisfying the following restriction: each atomic formula containing *quantified* variables must be a difference-bound arithmetic constraint.

Definition 2 (Restricted extension of Quantified Set constraints with Presburger Arithmetic). *Formulae of $\mathcal{RQSPA}$ are defined by the rules:*

$$\begin{aligned}
\Phi &::= T_s \asymp T_s \mid T_i \bowtie T_i + c \mid T_m \bowtie 0 \mid \Phi \wedge \Phi \mid \neg\Phi \mid \forall x.\ \Phi \mid \forall S.\ \Phi,\\
T_s &::= \varnothing \mid S \mid \{T_i\} \mid T_s \cup T_s \mid T_s \cap T_s \mid T_s \setminus T_s,\\
T_i &::= c \mid x \mid \min(T_s) \mid \max(T_s),\\
T_m &::= c \mid x \mid \max(T_s) \mid \min(T_s) \mid T_m + T_m \mid T_m - T_m.
\end{aligned}$$

Here, T_s (resp. T_i) represents set (resp. integer) terms which are more general than those in $\mathcal{DBS}$, and T_m terms are *Presburger arithmetic expressions*. Let $\mathsf{Vars}(\Phi)$ (resp. $\mathsf{free}(\Phi)$) denote the set of variables (resp. free variables) occurring in Φ. We require that **all set variables in atomic formulae $T_m \bowtie 0$ are free**. To make the free variables explicit, we usually write $\Phi(\boldsymbol{x}, \boldsymbol{S})$ for a $\mathcal{RQSPA}$ formula Φ. Free variable names are assumed *not* to clash with the quantified ones.

Example 1. $\max(S_1 \cup S_2) - \min(S_1) - \max(S_2) < 0$ and $\forall S_1 \forall S_2.(S_2 \neq \varnothing \rightarrow \max(S_2) \leq \max(S_1 \cup S_2))$ are $\mathcal{RQSPA}$ formulae, while $\forall S_2.\ \max(S_1 \cup S_2) - \min(S_1) - \max(S_2) < 0$ is *not*. □

The work [6], among others, studied *Presburger arithmetic extended with Sets* ($\mathcal{PS}$), which is *quantifier-free* $\mathcal{RQSPA}$ formulae. In this paper, $\mathcal{PS}$ will serve the data formula part of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$, and we reserve $\Delta, \Delta', \ldots$ to denote formulae from $\mathcal{PS}$ (see Sect. 3).

Semantics. All of these logics ($\mathcal{DBS}$, $\mathcal{RQSPA}$, $\mathcal{PS}$) can be considered as instances of weak monadic second-order logic, and thus their semantics are largely self-explanatory. In particular, set variables are interpreted as *finite* subsets of $\mathbb{Z}$ and integer variables are interpreted as integers. We emphasize that, if a set term T_s is interpreted as $\varnothing$, $\min(T_s)$ and $\max(T_s)$ are undefined. As a result, we stipulate that **any atomic formula containing an undefined term is interpreted as `false`**.

For an $\mathcal{RQSPA}$ formula $\Phi(\boldsymbol{x}, \boldsymbol{S})$ with $\boldsymbol{x} = (x_1, \cdots, x_k)$ and $\boldsymbol{S} = (S_1, \cdots, S_l)$, $\mathcal{L}(\Phi(\boldsymbol{x}, \boldsymbol{S}))$ denotes

$$\{(n_1, \cdots, n_k, A_1, \cdots, A_l) \in \mathbb{Z}^k \times \mathbb{S}^l_{\mathbb{Z}} \mid \Phi(n_1, \cdots, n_k, A_1, \cdots, A_l)\}.$$

[3] An unrestricted extension of quantified set constraints with Presburger Arithmetic is undecidable, as shown in [6].

As expected, typically we use $\mathcal{DBS}$ formulae to define relations between (tuples of) sets from $\mathbb{S}_{\mathbb{Z}}^k$. We say a relation $R \subseteq \mathbb{S}_{\mathbb{Z}}^k \times \mathbb{S}_{\mathbb{Z}}^k$ a *difference-bound set relation* if there is a $\mathcal{DBS}$ formula $\varphi(\boldsymbol{S}, \boldsymbol{S}')$ over set variables $\boldsymbol{S}$ and $\boldsymbol{S}'$ such that $R = \{(\boldsymbol{A}, \boldsymbol{A}') \in \mathbb{S}_{\mathbb{Z}}^k \times \mathbb{S}_{\mathbb{Z}}^k \mid \varphi(\boldsymbol{A}, \boldsymbol{A}')\}$. The *transitive closure* of R is defined in a standard way, viz., $\bigcup_{i \geq 0} R^i$, where $R^0 = \{(\boldsymbol{A}, \boldsymbol{A}) \mid \boldsymbol{A} \in \mathbb{S}_{\mathbb{Z}}^k\}$ and $R^{i+1} = R^i \cdot R$.

3 Linearly Compositional SLID with Set Data Constraints

In this section, we introduce separation logic with *linearly compositional* inductive predicates and *set data* constraints, denoted by $\mathsf{SLID}_{\mathsf{LC}}^{\mathsf{S}}[P]$, where P is an *inductive predicate*. In addition to the integer and set data types introduced in Sect. 2, we also consider the *location* data type $\mathbb{L}$. As a convention, $l, l', \cdots \in \mathbb{L}$ denote locations and $E, F, X, Y, \cdots$ range over location variables. We consider location fields associated with $\mathbb{L}$ and data fields associated with $\mathbb{Z}$.

$\mathsf{SLID}_{\mathsf{LC}}^{\mathsf{S}}[P]$ formulae may contain inductive predicates, each of which is of the form $P(\boldsymbol{\alpha}; \boldsymbol{\beta}; \boldsymbol{\xi})$ and has an associated inductive definition. The parameters are classified into three groups: *source parameters* $\boldsymbol{\alpha}$, *destination parameters* $\boldsymbol{\beta}$, and *static parameters* $\boldsymbol{\xi}$. We require that the source parameters $\boldsymbol{\alpha}$ and the destination parameters $\boldsymbol{\beta}$ are *matched* in type, namely, the two tuples have the same length $\ell > 0$ and for each $i \in [\ell]$, α_i and β_i have the same data type. Static parameters are typically used to store some static (global) information of dynamic data structures, e.g., the target location of tail pointers. Moreover, we assume that for each $i \in [\ell]$, α_i is of either the location type, or the *set type*. (There are no parameters of the integer type.) Without loss of generality, it is assumed that the first components of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are location variables; we usually explicitly write $E, \boldsymbol{\alpha}$ and $F, \boldsymbol{\beta}$.

$\mathsf{SLID}_{\mathsf{LC}}^{\mathsf{S}}[P]$ formulae comprise three types of formulae: *pure formulae* Π, *data formulae* Δ, and *spatial formulae* Σ. The data formulae are simply $\mathcal{PS}$ introduced in Sect. 2, while Π and Σ are defined by the following rules,

$$\begin{array}{rll} \Pi ::= & E = F \mid E \neq F \mid \Pi \wedge \Pi & \text{(pure formulae)} \\ \Sigma ::= & \mathtt{emp} \mid E \mapsto (\rho) \mid P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi}) \mid \Sigma * \Sigma & \text{(spatial formulae)} \\ \rho ::= & (f, X) \mid (d, T_i) \mid \rho, \rho & \text{(fields)} \end{array}$$

where T_i is an integer term as in Definition 2, and f (resp. d) is a location (resp. data) field. For spatial formulae Σ, formulae of the form $\mathtt{emp}$, $E \mapsto (\rho)$, or $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi})$ are called *spatial atoms*. In particular, formulae of the form $E \mapsto (\rho)$ and $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi})$ are called *points-to* and *predicate* atoms respectively. Moreover, E is *the root* of these points-to or predicate atoms.

Linearly Compositional Inductive Predicates. An inductive predicate P is *linearly compositional* if the inductive definition of P is given by the following two rules,

– base rule $R_0 : P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi}) ::= E = F \wedge \boldsymbol{\alpha} = \boldsymbol{\beta} \wedge \mathtt{emp}$,
– inductive rule $R_1 : P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi}) ::= \exists \boldsymbol{X} \exists \boldsymbol{S}.\ \varphi \wedge E \mapsto (\rho) * P(Y, \boldsymbol{\gamma}; F, \boldsymbol{\beta}; \boldsymbol{\xi})$.

The left-hand (resp. right-hand) side of a rule is called the *head* (resp. *body*) of the rule.

In the sequel, we specify some constraints on the inductive rule R_1 which are vital to obtain *complete* decision procedures for the satisfiability problem.

C1 None of the variables from $F, \boldsymbol{\beta}$ occur elsewhere in the right-hand side of R_1, that is, in φ, $E \mapsto (\rho)$.
C2 The data constraint φ in the body of R_1 is a $\mathcal{DBS}$ formula.
C3 For each atomic formula in φ, there is i such that all the variables in the atomic formula are from $\{\alpha_i, \gamma_i\}$.
C4 Each variable occurs in each of $P(Y, \boldsymbol{\gamma}; F, \boldsymbol{\beta}; \boldsymbol{\xi})$ and ρ at most once.
C5 $\boldsymbol{\xi}$ contains *only* location variables and all location variables from $\boldsymbol{\alpha} \cup \boldsymbol{\xi} \cup \boldsymbol{X}$ occur in ρ.
C6 $Y \in \boldsymbol{X}$ and $\boldsymbol{\gamma} \subseteq \{E\} \cup \boldsymbol{X} \cup \boldsymbol{S}$.

Note that, by **C6**, none of the variables from $\boldsymbol{\alpha} \cup \boldsymbol{\xi}$ occur in $\boldsymbol{\gamma}$. Moreover, from **C5** and **C6**, Y occurs in ρ, which guarantees that in each model of $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi})$, the sub-heap represented by $P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi})$, seen as a directed graph, is connected. We also note that the body of R_1 does *not* contain pure formulae. We remark that these constraints are undeniably technical. However, in practice the inductive predicates satisfying these constraints are usually sufficient to define linear data structures with set data constraints, cf. Example 2.

For an inductive predicate P, let $\mathrm{Flds}(P)$ denote the set of all fields occurring in the inductive rules of P. 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 $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$ for the collection of separation logic formulae $\phi = \Pi \wedge \Delta \wedge \Sigma$ satisfying the following constraints: (1) P is a linearly compositional inductive predicate, and (2) each predicate atom of Σ is of the form $P(-)$, and for each points-to atom occurring in Σ, the set of fields of this atom is $\mathrm{Flds}(P)$.

For an $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$ formula ϕ, let $\mathsf{Vars}(\phi)$ (resp. $\mathsf{LVars}(\phi)$, resp. $\mathsf{DVars}(\phi)$, resp. $\mathsf{SVars}(\phi)$) denote the set of (resp. location, resp. integer, resp. set) variables occurring in ϕ. Moreover, we use $\phi[\boldsymbol{\mu}/\boldsymbol{\alpha}]$ to denote the simultaneous replacement of the variables α_j by μ_j in ϕ. We adopt the standard *classic, precise semantics* of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$ in terms of *states*. In particular, a *state* is a pair (s, h), where s is an assignment and h is a heap. The details can be found in [14].

Example 2. We collect a few examples of linear data structures with set data constraints definable in $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$:

sdllseg for sorted doubly linked list segments, $sdllseg(E,P,S;F,L,S') ::= E = F \wedge P = L \wedge S = S' \wedge \mathtt{emp}$, $sdllseg(E,P,S;F,L,S') ::= \exists X, S''.\ S = S'' \cup \{\min(S)\} \wedge$ $E \mapsto ((\mathtt{next},X),(\mathtt{prev},P),(\mathtt{data},\min(S))) * sdllseg(X,E,S'';F,L,S')$.
plseg for list segments where the data values are consecutive, $plseg(E,S;F,S') ::= E = F \wedge S = S' \wedge \mathtt{emp}$, $plseg(E,S;F,S') ::= \exists X, S''.\ S = S'' \cup \{\min(S)\} \wedge \min(S'') = \min(S)+1 \wedge$ $E \mapsto ((\mathtt{next},X),(\mathtt{data},\min(S))) * plseg(X,S'';F,S')$.
ldllseg for doubly list segments, to mimic lengths with sets, $ldllseg(E,P,S;F,L,S') ::= E = F \wedge P = L \wedge S = S' \wedge \mathtt{emp}$, $ldllseg(E,P,S;F,L,S') ::= \exists X, S''.\ S = S'' \cup \{\max(S)\} \wedge \max(S'') = \max(S)-1 \wedge$ $E \mapsto ((\mathtt{next},X),(\mathtt{prev},P)) * ldllseg(X,E,S'';F,L,S')$.

4 Satisfiability of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$

The satisfiability problem is to decide whether there is a state (an assignment-heap pair) satisfying ϕ for a given $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$ formula ϕ. We shall follow the approach adopted in [12,15], i.e., to construct $\mathsf{Abs}(\phi)$, an abstraction of ϕ that is equisatisfiable to ϕ. The key ingredient of the construction is to compute the transitive closure of the data constraints extracted from the inductive rule of P.

Let $\phi = \Pi \wedge \Delta \wedge \Sigma$ be an $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$ formula. Suppose $\Sigma = a_1 * \cdots * a_n$, where each a_i is either a points-to atom or a predicate atom. For predicate atom $a_i = P(Z_1, \boldsymbol{\mu}; Z_2, \boldsymbol{\nu}; \boldsymbol{\chi})$ we assume that the inductive rule for P is

$$R_1 : P(E, \boldsymbol{\alpha}; F, \boldsymbol{\beta}; \boldsymbol{\xi}) ::= \exists \boldsymbol{X} \exists \boldsymbol{S}.\ \varphi \wedge E \mapsto (\rho) * P(Y, \boldsymbol{\gamma}; F, \boldsymbol{\beta}; \boldsymbol{\xi}). \tag{1}$$

We extract the data constraint $\varphi_P(\mathsf{dt}(\boldsymbol{\alpha}), \mathsf{dt}(\boldsymbol{\beta}))$ out of R_1. Formally, we define $\varphi_P(\mathsf{dt}(\boldsymbol{\alpha}), \mathsf{dt}(\boldsymbol{\beta}))$ as $\varphi[\mathsf{dt}(\boldsymbol{\beta})/\mathsf{dt}(\boldsymbol{\gamma})]$, where $\mathsf{dt}(\boldsymbol{\alpha})$ (resp. $\mathsf{dt}(\boldsymbol{\gamma})$, $\mathsf{dt}(\boldsymbol{\beta})$) is the projection of $\boldsymbol{\alpha}$ (resp. $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$) to data variables. For instance, $\varphi_{ldllseg}(S, S') := (S = S'' \cup \{\max(S)\} \wedge \max(S'') = \max(S) - 1)\,[S'/S''] = S = S' \cup \{\max(S)\} \wedge \max(S') = \max(S) - 1$.

We can construct $\mathsf{Abs}(\phi)$ with necessary adaptations from [15]. For each spatial atom a_i, $\mathsf{Abs}(\phi)$ introduces a Boolean variable to denote whether a_i corresponds to a nonempty heap or not. With these Boolean variables, the semantics of separating conjunction are encoded in $\mathsf{Abs}(\phi)$. Moreover, for each predicate atom a_i, $\mathsf{Abs}(\phi)$ contains an abstraction of a_i, where the formulae $\mathsf{Ufld}_1(a_i)$ and $\mathsf{Ufld}_{\geq 2}(a_i)$ are used. Intuitively, $\mathsf{Ufld}_1(a_i)$ and $\mathsf{Ufld}_{\geq 2}(a_i)$ correspond to the separation logic formulae obtained by unfolding the rule R_1 *once* and *at least twice* respectively. We include the construction here so one can see the role of the transitive closure in $\mathsf{Abs}(\phi)$. The details of $\mathsf{Abs}(\phi)$ can be found in [14].

Let $a_i = P(Z_1, \boldsymbol{\mu}; Z_2, \boldsymbol{\nu}; \boldsymbol{\chi})$ and R_1 be the inductive rule in Eq. (1). If E occurs in $\boldsymbol{\gamma}$ in the body of R_1, we use $\mathsf{idx}_{(P,\boldsymbol{\gamma},E)}$ to denote the unique index j such that $\gamma_j = E$. (The uniqueness follows from **C4**.)

Definition 3 ($\mathsf{Ufld}_1(a_i)$ **and** $\mathsf{Ufld}_{\geq 2}(a_i)$)**.** $\mathsf{Ufld}_1(a_i)$ *and* $\mathsf{Ufld}_{\geq 2}(a_i)$ *are defined by distinguishing the following two cases:*

– *If* E *occurs in* $\boldsymbol{\gamma}$ *in the body of* R_1*, then* $\mathsf{Ufld}_1(a_i) := (E = \beta_{\mathsf{idx}_{(P,\boldsymbol{\gamma},E)}} \wedge \varphi_P(\mathsf{dt}(\boldsymbol{\alpha}), \mathsf{dt}(\boldsymbol{\beta})))[Z_1/E, \boldsymbol{\mu}/\boldsymbol{\alpha}, Z_2/F, \boldsymbol{\nu}/\boldsymbol{\beta}, \boldsymbol{\chi}/\boldsymbol{\xi}]$ *and* $\mathsf{Ufld}_{\geq 2}(a_i) :=$

$$\begin{pmatrix} E \neq \beta_{\mathsf{idx}_{(P,\boldsymbol{\gamma},E)}} \wedge E \neq \gamma_{2,\mathsf{idx}_{(P,\boldsymbol{\gamma},E)}} \wedge \\ \varphi_P[\mathsf{dt}(\boldsymbol{\gamma_1})/\mathsf{dt}(\boldsymbol{\beta})] \wedge \varphi_P[\mathsf{dt}(\boldsymbol{\gamma_1})/\mathsf{dt}(\boldsymbol{\alpha}), \mathsf{dt}(\boldsymbol{\gamma_2})/\mathsf{dt}(\boldsymbol{\beta})] \wedge \\ (\mathsf{TC}[\varphi_P])[\mathsf{dt}(\boldsymbol{\gamma_2})/\mathsf{dt}(\boldsymbol{\alpha})] \end{pmatrix} [Z_1/E, \boldsymbol{\mu}/\boldsymbol{\alpha}, Z_2/F, \boldsymbol{\nu}/\boldsymbol{\beta}, \boldsymbol{\chi}/\boldsymbol{\xi}],$$

where $\boldsymbol{\gamma_1}$ *and* $\boldsymbol{\gamma_2}$ *are fresh variables.*

– *Otherwise, let* $\mathsf{Ufld}_1(a_i) := \varphi_P[Z_1/E, \boldsymbol{\mu}/\boldsymbol{\alpha}, Z_2/F, \boldsymbol{\nu}/\boldsymbol{\beta}, \boldsymbol{\chi}/\boldsymbol{\xi}]$ *and*

$$\mathsf{Ufld}_{\geq 2}(a_i) := \begin{pmatrix} \varphi_P[\mathsf{dt}(\boldsymbol{\gamma_1})/\mathsf{dt}(\boldsymbol{\beta})] \wedge \\ \varphi_P[\mathsf{dt}(\boldsymbol{\gamma_1})/\mathsf{dt}(\boldsymbol{\alpha}), \mathsf{dt}(\boldsymbol{\gamma_2})/\mathsf{dt}(\boldsymbol{\beta})] \wedge \\ (\mathsf{TC}[\varphi_P])[\mathsf{dt}(\boldsymbol{\gamma_2})/\mathsf{dt}(\boldsymbol{\alpha})] \end{pmatrix} [Z_1/E, \boldsymbol{\mu}/\boldsymbol{\alpha}, Z_2/F, \boldsymbol{\nu}/\boldsymbol{\beta}, \boldsymbol{\chi}/\boldsymbol{\xi}],$$

where $\boldsymbol{\gamma_1}$ *and* $\boldsymbol{\gamma_2}$ *are fresh variables.*

Here, $\mathsf{TC}[\varphi_P](\mathsf{dt}(\boldsymbol{\alpha}), \mathsf{dt}(\boldsymbol{\beta}))$ denotes the transitive closure of φ_P. In Sect. 5, it will be shown that $\mathsf{TC}[\varphi_P](\mathsf{dt}(\boldsymbol{\alpha}), \mathsf{dt}(\boldsymbol{\beta}))$ can be written as an $\mathcal{RQSPA}$ formula. As a result, since we are only concerned with satisfiability and can treat the location data type $\mathbb{L}$ simply as integers $\mathbb{Z}$, $\mathsf{Abs}(\phi)$ can also be read as an $\mathcal{RQSPA}$ formula. In Sect. 6, we shall show that the satisfiability of $\mathcal{RQSPA}$ is decidable. Following this chain of reasoning, we conclude that *the satisfiability of* $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$ *formulae is decidable.*

5 Transitive Closure of Difference-Bound Set Relations

In this section, we show how to compute the transitive closure of the difference-bound set relation R given by a $\mathcal{DBS}$ formula $\varphi_R(\boldsymbol{S}, \boldsymbol{S}')$. Our approach is, in a nutshell, to encode $TC[\varphi_R](\boldsymbol{S}, \boldsymbol{S}')$ into $\mathcal{RQSPA}$. We shall only sketch part of a simple case, i.e., in $\varphi_R(S, S')$ only one source and destination set parameter are present. The details are however given in [14].

Recall that, owing to the simplicity of $\mathcal{DBS}$, the integer terms T_i in $\varphi_R(S, S')$ can only be $\min(S)$, $\max(S)$, $\min(S')$ or $\max(S')$, whereas the set terms T_s are $\varnothing$, $\{\min(S)\}$, $\{\min(S')\}$, $\{\max(S)\}$, $\{\max(S')\}$, or their union. For reference, we write $\varphi_R(S, S') = \varphi_{R,1} \wedge \varphi_{R,2}$, where $\varphi_{R,1}$ is an equality of set terms (i.e., they are of the form $S = S' \cup T_s$ or $S' = S \cup T_s$), and $\varphi_{R,2}$ is a conjunction of constraints over integer terms (i.e., a conjunction of formulae $T_i \leq T_i + c$). $\varphi_{R,1}$ and $\varphi_{R,2}$ will be referred to as the *set* and *integer subformula* of $\varphi_R(S, S')$ respectively. We shall focus on the case $\varphi_{R,1} := S = S' \cup T_s$. The symmetrical case $\varphi_{R,1} := S' = S \cup T_s$ can be adapted easily.

The integer subformula $\varphi_{R,2}$ can be represented by an *edge-weighted directed graph* $\mathcal{G}(\varphi_{R,2})$, where the vertices are all integer terms appearing in $\varphi_{R,2}$, and there is an edge from T_1 to T_2 with weight c iff $T_1 = T_2 + c$ (equivalent to $T_2 = T_1 - c$), or $T_1 \leq T_2 + c$, or $T_2 + c \geq T_1$ appears in $\varphi_{R,2}$. The weight of a path in $\mathcal{G}(\varphi_{R,2})$ is the sum of the weights of the edges along the path. A *negative cycle* in $\mathcal{G}(\varphi_{R,2})$ is a cycle with negative weight. It is known that $\varphi_{R,2}$ is satisfiable iff $\mathcal{G}(\varphi_{R,2})$ contains no negative cycles [24]. Suppose $\varphi_{R,2}$ is satisfiable. We define the *normal form* of $\varphi_{R,2}$, denoted by $\mathsf{Norm}(\varphi_{R,2})$, as the

conjunction of the formulae $T_1 \leq T_2 + c$ such that $T_1 \neq T_2$, T_2 is reachable from T_1 in $\mathcal{G}(\varphi_{R,2})$, and c is path from T_1 to T_2 with the minimal weight in $\mathcal{G}(\varphi_{R,2})$.

S (resp. S') is said to be *surely nonempty* in φ_R if $\min(S)$ or $\max(S)$ (resp. $\min(S')$ or $\max(S')$) occurs in φ_R; otherwise, S (resp. S') is *possibly empty* in φ_R. Recall that, according to the semantics, an occurrence of $\min(S)$ or $\max(S)$ (resp. $\min(S')$ or $\max(S')$) in φ_R implies that S (resp. S') is interpreted as a nonempty set in every satisfiable assignment. Provided that S' is nonempty, we know that $\min(S')$ and $\max(S')$ belong to S'. Therefore, for simplicity, here we assume that in $S = S' \cup T_s$, T_s contains neither $\min(S')$ nor $\max(S')$. The situation that T_s contains $\min(S')$ and $\max(S')$ can be dealt with in a similar way.

Saturation. For technical convenience, we introduce a concept of saturation. The main purpose of saturation is to regularise T_s and $\varphi_{R,2}$, which would make the transitive closure construction more "syntactic".

Definition 4. *Let $\varphi_R(S, S') := S = S' \cup T_s \wedge \varphi_{R,2}$ be a $\mathcal{DBS}$ formula. Then $\varphi_R(S, S')$ is saturated if $\varphi_R(S, S')$ satisfies the following conditions*

- *$\varphi_{R,2}$ is satisfiable and in normal forms,*
- *$T_s \subseteq \{\max(S), \min(S)\}$,*
- *if S (resp. S') is surely nonempty in φ_R, then $\varphi_{R,2}$ contains a conjunct $\min(S) \leq \max(S) - c$ for some $c \geq 0$ (resp. $\min(S') \leq \max(S') - c'$ for some $c' \geq 0$),*
- *if both S and S' are surely nonempty in φ_R, then*
  - *$\varphi_{R,2}$ contains two conjuncts $\min(S) \leq \min(S') - c$ and $\max(S') \leq \max(S) - c'$ for some $c, c' \geq 0$,*
  - *$\min(S) \notin T_s$ iff $\varphi_{R,2}$ contains the conjuncts $\min(S) \leq \min(S')$ and $\min(S') \leq \min(S)$,*
  - *$\max(S) \notin T_s$ iff $\varphi_{R,2}$ contains the conjuncts $\max(S') \leq \max(S)$ and $\max(S) \leq \max(S')$,*
- *if $\varphi_{R,2}$ contains the conjuncts $\min(S) \leq \max(S)$ and $\max(S) \leq \min(S)$, then $\max(S) \notin T_s$ (possibly $\min(S) \in T_s$).*

For a formula $\varphi_R(S, S') := S = S' \cup T_s \wedge \varphi_{R,2}$, one can easily saturate φ_R, yielding a saturated formula $\mathsf{Strt}(\varphi_R(S, S'))$. (It is possible, however, to arrive at an unsatisfiable formula, then we are done.)

Proposition 1. *Let $\varphi_R(S, S') := \varphi_{R,1} \wedge \varphi_{R,2}$ be a $\mathcal{DBS}$ formula such that $\varphi_{R,1} := S = S' \cup T_s$ and $\varphi_{R,2}$ is satisfiable. Then φ_R can be transformed, in polynomial time, to an equisatisfiable formula $\mathsf{Strt}(\varphi_R(S, S'))$, and if the integer subformula of $\mathsf{Strt}(\varphi_R(S, S'))$ is satisfiable, then $\mathsf{Strt}(\varphi_R(S, S'))$ is saturated.*

In the sequel, we assume that $\varphi_R(S, S') := \varphi_{R,1} \wedge \varphi_{R,2}$ is satisfiable and *saturated*. For notational convenience, for $A \subseteq \{\min(S), \max(S), \min(S'), \max(S')\}$ with $|A| = 2$, let $\lfloor \varphi_{R,2} \rfloor_A$ denote the conjunction of atomic formulae in $\varphi_{R,2}$ where *all* the elements of A occur.

Evidently, $\lfloor\varphi_{R,2}\rfloor_A$ gives a partition of atomic formulae of $\varphi_{R,2}$. Namely,

$$\varphi_{R,2} = \bigwedge\nolimits_{A\subseteq\{\min(S),\max(S),\min(S'),\max(S')\},|A|=2} \lfloor\varphi_{R,2}\rfloor_A.$$

We proceed by a case-by-case analysis of $\varphi_{R,1}$. There are four cases: (I) $\varphi_{R,1} := S = S'$, (II) $\varphi_{R,1} := S = S' \cup \{\min(S)\}$, (III) $\varphi_{R,1} = S = S' \cup \{\max(S)\}$ and (IV) $\varphi_{R,1} = S = S' \cup \{\min(S), \max(S)\}$. Case (I) is trivial, and Case (III) is symmetrical to (II). However, both (II) and (IV) are technically involved. We shall only give a "sample" treatment of these cases, i.e., part of arguments for Case (II); the full account of Case (II) and (IV) are given in [14].

To start with, Case (II) can be illustrated schematically as $\underbrace{|-\overbrace{|-----|}^{S'}}_{S}$. We observe that S is surely nonempty in φ_R. We then distinguish two subcases depending on whether S' is possibly empty or surely nonempty in φ_R. Here we give the details of the latter subcase because it is more interesting. In this case, both S and S' are surely nonempty in φ_R. By Definition 4 (4–5), $\varphi_{R,2}$ contains a conjunct $\min(S) \leq \min(S') - c$ for some $c \geq 0$, as well as $\max(S') \leq \max(S)$ and $\max(S) \leq \max(S')$ (i.e., $\max(S') = \max(S)$). Therefore, we can assume

$$\varphi_{R,2} = \max(S') \leq \max(S) \wedge \max(S) \leq \max(S') \wedge \lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')} \wedge \lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)} \wedge \lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S')}.$$

Note that in $\varphi_{R,2}$ above, the redundant subformulae $\lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S')}$ and $\lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S)}$ have been omitted.

The formula $\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}$ is said to be *strict* if it contains a conjunct $\min(S) \leq \min(S') - c$ for some $c > 0$. Otherwise, it is said to be *non-strict*. Intuitively, if $\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}$ is strict, then for $n, n' \in \mathbb{Z}$, the validity of $(\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')})[n/\min(S), n'/\min(S')]$ implies that $n < n'$. For the sketch we only present *the case that* $\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}$ *is strict*; the other cases are similar and can be found in [14].

Evidently, $\mathsf{TC}[\varphi_R](S, S')$ can be written as $(S = S') \vee \bigvee_{n\geq 1} \varphi_R^{(n)}$, where $\varphi_R^{(n)}$ is obtained by unfolding φ_R for n times, that is,

$$\varphi_R^{(n)} = \exists S_1, \cdots, S_{n+1}. \left(\begin{array}{l} S_1 = S \wedge S_{n+1} = S' \wedge \\ \bigwedge_{i\in[n]} (S_i = S_{i+1} \cup \{\min(S_i)\} \wedge \varphi_{R,2}[S_i/S, S_{i+1}/S']) \end{array} \right),$$

where $\varphi_{R,2}[S_i/S, S_{i+1}/S']$ is obtained from $\varphi_{R,2}$ by replacing S (resp. S') with S_i (resp. S_{i+1}).

Clearly, $\varphi_R^{(1)} = \varphi_R$, and

$\varphi_R^{(2)} = \exists S_2.\ (S = S_2 \cup \{\min(S)\} \wedge S_2 = S' \cup \{\min(S_2)\} \wedge \varphi_{R,2}[S_2/S'] \wedge \varphi_{R,2}[S_2/S])$.

For $\varphi_R^{(n)}$ where $n \geq 3$, we first simplify $\varphi_R^{(n)}$ to construct a finite formula for $\mathsf{TC}[\varphi_R](S, S')$. The subformula $\bigwedge_{i\in[n]} (S_i = S_{i+1} \cup \{\min(S_i)\} \wedge \varphi_{R,2}[S_i/S, S_{i+1}/S'])$ can be rewritten as

$$\bigwedge_{i\in[n]} \begin{pmatrix} S_i = S_{i+1} \cup \{\min(S_i)\} \wedge \max(S_i) = \max(S_{i+1}) \wedge \\ (\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}[S_i/S, S_{i+1}/S']) \wedge (\lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_i/S]) \wedge \\ (\lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S')}[S_{i+1}/S']) \end{pmatrix}.$$

Because $S_i = S_{i+1} \cup \{\min(S_i)\}$ for each $i \in [n]$, we have $\max(S_1) = \cdots = \max(S_n)$ and $\min(S_1) \leq \cdots \leq \min(S_n)$. Since $\lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}$ is a conjunction of difference-bound constraints involving $\min(S)$ and $\max(S)$ only, we have $\bigwedge_{i\in[n]} \lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_i/S]$ is equivalent to $\lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_1/S] \wedge \lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_n/S]$. To see this, assume, for instance,

$$\lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)} \equiv c \leq \max(S) - \min(S) \leq c'$$

for some constants $c, c' \geq 0$ with $c \leq c'$. Then $\max(S_1) - \min(S_1) \leq c'$ implies $\max(S_i) - \min(S_i) \leq c'$ for each $i \in [n]$, and $c \leq \max(S_n) - \min(S_n)$ implies $c \leq \max(S_i) - \min(S_i)$ for each $i \in [n]$. Therefore, $\lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_1/S] \wedge \lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_n/S] \equiv c \leq \max(S_1) - \min(S_1) \leq c' \wedge c \leq \max(S_n) - \min(S_n) \leq c'$ implies that $\bigwedge_{i\in[n]} \lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_i/S]$, thus they are equivalent. (The other direction is trivial.) Likewise, one has $\lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S')}[S_2/S'] \wedge \lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S')}[S_{n+1}/S']$ implies $\bigwedge_{i\in[n]} \lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S')}[S_{i+1}/S']$, thus they are equivalent. Therefore, $\varphi_R^{(n)}$ can be transformed into

$$\exists S_2, S_n. \begin{pmatrix} \lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)} \wedge (\lfloor\varphi_{R,2}\rfloor_{\min(S),\max(S)}[S_n/S]) \wedge \\ (\lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S')}[S_2/S']) \wedge \lfloor\varphi_{R,2}\rfloor_{\min(S'),\max(S')} \wedge S = S_2 \cup \{\min(S)\} \wedge \\ S_n = S' \cup \{\min(S_n)\} \wedge \max(S) = \max(S_2) \wedge \max(S_n) = \max(S') \wedge \\ (\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}[S_2/S']) \wedge (\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}[S_n/S]) \wedge \\ \exists S_3, \cdots, S_{n-1}. \bigwedge_{2\leq i\leq n-1} \begin{pmatrix} S_i = S_{i+1} \cup \{\min(S_i)\} \wedge \max(S_i) = \max(S_{i+1}) \wedge \\ (\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}[S_i/S, S_{i+1}/S']) \end{pmatrix} \end{pmatrix}.$$

Claim. *Suppose $n \geq 3$ and $\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}$ is strict. Then*

$$\exists S_3, \cdots, S_{n-1}. \bigwedge_{2\leq i\leq n-1} \begin{pmatrix} S_i = S_{i+1} \cup \{\min(S_i)\} \wedge \max(S_i) = \max(S_{i+1}) \wedge \\ (\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}[S_i/S, S_{i+1}/S']) \end{pmatrix}$$

is equivalent to

$$S_n \neq \emptyset \wedge S_2 \setminus S_n \neq \emptyset \wedge S_n \subseteq S_2 \wedge |S_2 \setminus S_n| = n - 2 \wedge \max(S_2 \setminus S_n) < \min(S_n) \wedge \\ \forall y, z.\ \mathit{succ}((S_2 \setminus S_n) \cup \{\min(S_n)\}, y, z) \rightarrow (\lfloor\varphi_{R,2}\rfloor_{\min(S),\min(S')}[y/\min(S), z/\min(S')]),$$

where $\mathsf{succ}(S, x, y)$ specifies intuitively that y is the successor of x in S, that is,

$$\mathsf{succ}(S, x, y) = x \in S \wedge y \in S \wedge x < y \wedge \forall z \in S.\ (z \leq x \vee y \leq z).$$

Note that $|\cdot|$ denotes the set cardinality which can be easily encoded into $\mathcal{RQSPA}$. ([14] gives the proof of the claim.) It follows that $\mathsf{TC}[\varphi_R](S, S') =$

$$(S = S') \vee \varphi_R(S, S') \vee \varphi_R^{(2)}(S, S') \vee \exists S_1, S_2. \left(\begin{array}{l} S = S_1 \cup \{\min(S)\} \wedge S_2 = S' \cup \{\min(S_2)\} \wedge \\ \max(S) = \max(S_1) \wedge \max(S_2) = \max(S') \wedge \\ S_2 \neq \emptyset \wedge S_1 \setminus S_2 \neq \emptyset \wedge S_2 \subseteq S_1 \wedge \max(S_1 \setminus S_2) < \min(S_2) \wedge \\ \lfloor \varphi_{R,2} \rfloor_{\min(S),\max(S)} \wedge (\lfloor \varphi_{R,2} \rfloor_{\min(S),\max(S)}[S_2/S]) \wedge \\ (\lfloor \varphi_{R,2} \rfloor_{\min(S'),\max(S')}[S_1/S']) \wedge \lfloor \varphi_{R,2} \rfloor_{\min(S'),\max(S')} \wedge \\ (\lfloor \varphi_{R,2} \rfloor_{\min(S),\min(S')}[S_1/S']) \wedge (\lfloor \varphi_{R,2} \rfloor_{\min(S),\min(S')}[S_2/S]) \wedge \\ \forall y, z. \left(\begin{array}{r} \mathsf{succ}((S_1 \setminus S_2) \cup \{\min(S_2)\}, y, z) \rightarrow \\ (\lfloor \varphi_{R,2} \rfloor_{\min(S),\min(S')}[y/\min(S), z/\min(S')]) \end{array} \right) \end{array} \right).$$

6 Satisfiability of $\mathcal{RQSPA}$

In this section, we focus on the second ingredient of the procedure for deciding satisfiability of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}[P]$, i.e., the satisfiability of $\mathcal{RQSPA}$. We first note that $\mathcal{RQSPA}$ is defined over $\mathbb{Z}$. To show the decidability, it turns to be much easier to work on $\mathbb{N}$. We shall write $\mathcal{RQSPA}_{\mathbb{Z}}$ and $\mathcal{RQSPA}_{\mathbb{N}}$ to differentiate them when necessary. Moreover, for technical reasons, we also introduce $\mathcal{RQSPA}^{-}$, the fragment of $\mathcal{RQSPA}$ excluding formulae of the form $T_m \bowtie 0$.

The decision procedure for the satisfiability of $\mathcal{RQSPA}$ proceeds with the following three steps:

Step I. Translate $\mathcal{RQSPA}_{\mathbb{Z}}$ to $\mathcal{RQSPA}_{\mathbb{N}}$,

Step II. Normalize an $\mathcal{RQSPA}_{\mathbb{N}}$ formula $\Phi(\boldsymbol{x}, \boldsymbol{S})$ into $\bigvee_i (\Phi^{(i)}_{\text{core}} \wedge \Phi^{(i)}_{\text{count}})$, where $\Phi^{(i)}_{\text{core}}$ is an $\mathcal{RQSPA}^{-}_{\mathbb{N}}$ formula, and $\Phi^{(i)}_{\text{count}}$ is a conjunction of formulae of the form $T_m \bowtie 0$ which contain only variables from $\boldsymbol{x} \cup \boldsymbol{S}$,

Step III. For each disjunct $\Phi^{(i)}_{\text{core}} \wedge \Phi^{(i)}_{\text{count}}$, construct a Presburger automaton (PA) $\mathcal{A}^{(i)}_{\Phi}$ which captures the models of $\Phi^{(i)}_{\text{core}} \wedge \Phi^{(i)}_{\text{count}}$. Satisfiability is thus reducible to the nonemptiness of PA, which is decidable [29].

These steps are technically involved. In particular, the third step requires exploiting Presburger automata [29]. The details can be found in [14].

7 Conclusion

In this paper, we have defined $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$, SL with linearly compositional inductive predicates and set data constraints. The main feature is to identify $\mathcal{DBS}$ as a special class of set data constraints in the inductive definitions. We encoded the transitive closure of $\mathcal{DBS}$ into $\mathcal{RQSPA}$, which was shown to be decidable. These together yield a complete decision procedure for the satisfiability of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$.

The precise complexity of the decision procedure—NONELEMENTARY is the best upper-bound we have now—is left open for further studies. Furthermore, the entailment problem of $\mathsf{SLID}^{\mathsf{S}}_{\mathsf{LC}}$ is an immediate future work.

References

1. 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, vol. 7561, pp. 167–182. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33386-6_14
2. Bozga, M., Gîrlea, C., Iosif, R.: Iterating octagons. In: TACAS, pp. 337–351 (2009)
3. Bozga, M., Iosif, R., Konecný, F.: Fast acceleration of ultimately periodic relations. In: CAV, pp. 227–242 (2010)
4. Bozga, M., Iosif, R., Lakhnech, Y.: Flat parametric counter automata. Fundam. Inf. **91**(2), 275–303 (2009)
5. Büchi, R.J.: Weak Second-Order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik **6**(1–6), 66–92 (1960)
6. Cantone, D., Cutello, V., Schwartz, J.T.: Decision problems for tarski and presburger arithmetics extended with sets. In: Börger, E., Kleine Büning, H., Richter, M.M., Schönfeld, W. (eds.) CSL 1990. LNCS, vol. 533, pp. 95–109. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-54487-9_54
7. 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)
8. Chu, D.-H., Jaffar, J., Trinh, M.-T.: Automatic induction proofs of data-structures in imperative programs. In: PLDI, pp. 457–466 (2015)
9. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and presburger arithmetic. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0028751
10. Cristiá, M., Rossi, G.: A decision procedure for restricted intensional sets. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 185–201. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_12
11. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Am. Math. Soc. **98**(1), 21–51 (1961)
12. Enea, C., Lengál, O., Sighireanu, M., Vojnar, T.: Compositional entailment checking for a fragment of separation logic. In: APLAS, pp. 314–333 (2014)
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). https://doi.org/10.1007/978-3-319-24953-7_7
14. Gao, C., Chen, T., Wu, Z.: Separation logic with linearly compositional inductive predicates and set data constraints (full version). http://arxiv.org/abs/1811.00699
15. Gu, X., Chen, T., Wu, Z.: A complete decision procedure for linearly compositional separation logic with data constraints. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 532–549. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_36
16. Halpern, J.Y.: Presburger arithmetic with unary predicates is Π_1^1-complete. J. Symb. Logic **56**(2), 637–642 (1991)
17. Horbach, M., Voigt, M., Weidenbach, C.: On the combination of the Bernays–Schönfinkel–Ramsey fragment with simple linear integer arithmetic. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 77–94. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_6
18. Klaedtke, F., Rueß, H.: Monadic second-order logics with cardinalities. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 681–696. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45061-0_54

19. Konečný, F.: PTIME computation of transitive closures of octagonal relations. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 645–661. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_42
20. Kuncak, V., Piskac, R., Suter, P.: Ordered sets in the calculus of data structures. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 34–48. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15205-4_5
21. 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). https://doi.org/10.1007/978-3-319-41528-4_21
22. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: POPL 2011, pp. 611–622. ACM (2011)
23. Madhusudan, P., Qiu, X., Stefanescu, A.: Recursive proofs for inductive tree data-structures. In: POPL, pp. 123–136 (2012)
24. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Danvy, O., Filinski, A. (eds.) PADO 2001. LNCS, vol. 2053, pp. 155–172. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44978-7_10
25. 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). https://doi.org/10.1007/3-540-44802-0_1
26. 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). https://doi.org/10.1007/978-3-642-39799-8_54
27. 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). https://doi.org/10.1007/978-3-319-08867-9_47
28. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS, pp. 55–74 (2002)
29. 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). https://doi.org/10.1007/978-3-540-27836-8_94
30. Suter, P., Dotta, M., Kuncak, V.: Decision procedures for algebraic data types with abstractions. In: POPL 2010, pp. 199–210. ACM (2010)
31. 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). https://doi.org/10.1007/978-3-319-47958-3_22
32. Voigt, M.: The Bernays–Schönfinkel–Ramsey fragment with bounded difference constraints over the reals is decidable. In: Dixon, C., Finger, M. (eds.) FroCoS 2017. LNCS (LNAI), vol. 10483, pp. 244–261. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66167-4_14
33. Wies, T., Piskac, R., Kuncak, V.: Combining theories with shared set operations. In: Ghilardi, S., Sebastiani, R. (eds.) FroCoS 2009. LNCS (LNAI), vol. 5749, pp. 366–382. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04222-5_23
34. Xu, Z., Chen, T., Wu, Z.: Satisfiability of compositional separation logic with tree predicates and data constraints. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 509–527. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_31