Beyond Shapes: Lists with Ordered Data

Kshitij Bansal*[1], Rémi Brochenin**[2], and Etienne Lozes[2]

[1] Chennai Mathematical Institute
kshitij@cmi.ac.in
[2] LSV, ENS Cachan, CNRS
{brocheni,lozes}@lsv.ens-cachan.fr

Abstract. Standard analysis on recursive data structures restrict their attention to shape properties (for instance, a program that manipulates a list returns a list), excluding properties that deal with the actual content of these structures. For instance, these analysis would not establish that the result of merging two ordered lists is an ordered list. Separation logic, one of the prominent framework for these kind of analysis, proposed a heap model that could represent data, but, to our knowledge, no predicate dealing with data has ever been integrated to the logic while preserving decidability. We establish decidability for (first-order) separation logic with a predicate that allows to compare two successive data in a list. We then consider the extension where two data in arbitrary positions may be compared, and establish the undecidability in general. We define a guarded fragment that turns out to be both decidable and sufficiently expressive to prove the preservation of the loop invariant of a standard program merging ordered lists. We finally consider the extension with the magic-wand and prove that, by constrast with the data-free case, even a very restricted use of the magic wand already introduces undecidability.

1 Introduction

Data-ordering and shape analysis Providing automatic methods for faults detection in programs manipulating recursive mutable data structures is a long-standing problem. Shape analysis is a well established approach that may detect faults due to in-depth properties of the heap, like creating a cycle in an acyclic list. Prominent logics that integrate such an analysis are separation logic [1], pointer assertion logic PAL [9], TVLA [10], LRP (logic of reachable patterns) [16], or alias logic [4], to quote a few examples. A common feature in these analyses is that they completely forget the data held in the recursive structures, focusing on the shape of the structure. As a consequence, ordering properties are out of the scope of these analyses: for instance, one cannot check or even specify that the reverse of a sorted list is a sorted list. Extensions of shape analysis have been proposed for ordering properties, stability properties,

* Supported by P2R MODISTE-COVER/Timed-DISCOVERI, a project under the Indo-French Networking Programme.
** Supported by a DGA/CNRS fellowship

and size properties, in shape graphs [3], in the TVLA approach [11], and in the separation logic approach [12] to cite a few. This paper studies the rather more theoretical issue of the decidability of the satisfiability problem. It proposes a general approach for reducing the shapes handling ordering properties to pure shapes, and stress some natural limitations we should put on the data properties we would like to check automatically.

Data-ordering in separation logic Our approach lies in the framework of separation logic [14]. In essence, separation logic extends first order logic with two substructural connectives: the *separation* connective ($*$) and its adjoint (the separating implication $-\!\!*$, also known as the *magic wand*). These connectives are convenient to express pre and post conditions of all standard heap-manipulating instructions. For instance, the strongest post condition $\mathbf{Post}(\mathsf{x} := \mathsf{new}, A)$ of a memory allocation instruction can be expressed by $\mathbf{x}\mapsto - * \exists x.A\{^x/_{\mathbf{x}}\}$. This formula involves two more ingredients : the use of first-order logical variables, that here quantify over the memory location of $\mathbf{x}$ before allocation, and the points-to predicate $.\hookrightarrow.$ (or its precise version in this example). We extend the logic with the predicate $\mathsf{val}(x) \leq \mathsf{val}(y)$ that asserts that the value stored at the location x is smaller than the one stored at y, which in particular allows to define the predicate

$$x\overset{\leq}{\hookrightarrow}y \quad \overset{\mathrm{def}}{\equiv} \quad x\hookrightarrow y \ \wedge \ \mathsf{val}(x) \leq \mathsf{val}(y)$$

and $x\overset{\geq}{\hookrightarrow}y$ accordingly. We call these predicates *short-distance comparisons*, and by contrast $\mathsf{val}(x) \leq \mathsf{val}(y)$ is called *long-distance comparison*. We moreover say that such a long-distance comparison is *guarded* if x or y is an open variable.

Separation logic's decidability The decidability of the satisfiability problem for separation logic has been intensively studied so far: first-order separation logic over heap models with at least two selectors (record fields) is known to be undecidable [7] by containment of finite satisfiability for classical predicate logic with one binary relation [15] (even with no separating connectives). On the other hand, first-order separation logic over heaps with one selector has been proved to be decidable when the magic-wand is dropped [6], by reduction to monadic second order logic over functional graphs, but becomes undecidable in presence of magic wand. To our knowledge, nothing was known about first-order separation logic with data. The following table summarizes the results we present in this paper:

Undecidable	long distance comparison without $-\!\!*$
	short distance comparison with (restricted) $-\!\!*$
Decidable	short + guarded long distance comparisons without $-\!\!*$

The decidability result comes from a reduction to monadic second order logic over functional graphs. The translation is strongly inspired by the one for separation logic over lists without data [6], but involves some non-trivial complications for ensuring the coherence of data abstraction. The undecidability results are obtained by reduction to first-order logic over (finite) data words, which was proved undecidable [2, 8].

Case study In order to illustrate the practical relevance of our results, we consider a very standard merge-sort program. Checking that *any* formula is a correct loop invariant requires in general to deal with the magic-wand connective, which leads to our undecidable fragments. However, for the loop invariant one may think about (that is, all working lists are ordered lists) the magic wand can be eliminated, and the formula considered falls into the decidable fragment.

Outline of the paper Section 2 introduces our separation logic over lists with data. In Section 3, we illustrate on the merge program how the logic can deal with relevant loop invariants. In Section 4, we establish the decidability of the short distance comparison. Section 5 deals with the case of guarded and non-guarded long-distance comparison, whereas Section 6 explains the undecidability of the logic in the presence of the magic wand.

2 Preliminaries

In this section, we introduce first the separation logic with data considered in this work, then the monadic second order logic to which we reduce to. These logics are based on different classes of models: our separation logic deals with lists with data, whereas the monadic second order logic deals with shapes, e.g. lists without data.

2.1 A Separation Logic for Lists with Ordered Data

Memory model We assume an infinite, totally ordered set $(\texttt{Dat}, \leq)$ of data, and range over a particular datum with α, β. We moreover assume an infinite set $\texttt{Loc}$ of locations, ranged over with l, l' etc. and an infinite set $\texttt{Var}$ of variables, ranged over with either x, y, z or $\mathbf{x}, \mathbf{y}, \mathbf{z}$ etc. Variables can be interpreted as both variables from the programs or logical variables quantifying over locations; the main difference between both is that program variables are never quantified in the formula. We safely identify them and will use the font convention $\mathbf{x}$, $\mathbf{y}$ to emphasize that a variable should be understood as a program variable. In the latter, we may use the standard notation $A\{{}^{y}/_{x}\}$ for the formula A in which x replaces y.

Following the standard semantics of separation logic, we define a memory state as a pair of a store s and a heap h such that:

- $s : \texttt{Var} \to \texttt{Loc}$,
- $h : \texttt{Loc} \rightharpoonup (\texttt{Loc} \times \texttt{Dat})$ is a partial function with finite domain.

We write $\texttt{dom}(h)$ to denote the domain of h and $\texttt{ran}(h)$ to denote its range. For $\mathcal{Z} \subseteq \texttt{dom}(h)$, We write $h_{|\mathcal{Z}}$ to denote the restriction of h to $\mathcal{Z}$. We write fst and snd to denote the first and second projection on a product set. We write $h \perp h'$ if $\texttt{dom}(h) \cap \texttt{dom}(h') = \emptyset$, and the heap composition $h * h'$ is defined as $h \cup h'$ when $h \perp h'$.

Example 1 (Ordered lists). Programs manipulating ordered lists of integers can be modeled choosing $\mathtt{Dat} = \mathbb{Z}$ with the standard order. The same holds for lists of reals, lists of naturals, and so on.

Example 2 (Fine-grained concurrent lists). $\mathtt{Dat}$ could be thought as the state of a lock at the current node, that is the identifier of the thread holding the node (or some constant for an available lock). Here, the ordering on data is not relevant, but the equality between data is. For such a model, one may want to express, for instance, that every thread holds the locks of at most two nodes of a list, and that these nodes are necessarily consecutive.

Separation logic We now define our assertion language $\mathtt{SL}_<$ by extending the standard separation logic with a comparison predicate. We assume a set $\mathtt{DVar}$ of data variables, ranged over with v, w, etc. A valuation interpreting data variables is a function $\rho : \mathtt{DVar} \to \mathtt{Dat}$.

Formulae of $\mathtt{SL}_<$ are defined by the grammar below.

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi \mid \exists v.\phi \mid x \hookrightarrow y \mid \mathsf{val}(x) \leq v \mid \mathsf{val}(x) \geq v \mid x = y \mid \phi * \phi \mid \phi \mathrel{-\!\!*} \phi$$

The semantics of the formulae is defined as usual, with the expected definition for the predicates $\mathsf{val}(x) \leq v$ and $\mathsf{val}(x) \geq v$.

$(s,h),\rho \models_{\mathsf{SL}} \phi \wedge \psi$	iff	$(s,h),\rho \models_{\mathsf{SL}} \phi$ and $(s,h),\rho \models_{\mathsf{SL}} \psi$
$(s,h),\rho \models_{\mathsf{SL}} \neg\phi$	iff	not $(s,h),\rho \models_{\mathsf{SL}} \phi$
$(s,h),\rho \models_{\mathsf{SL}} \exists x.\ \phi$	iff	there is $l \in \mathtt{Loc}$ such that $(s[x \mapsto l], h),\rho \models_{\mathsf{SL}} \phi$
$(s,h),\rho \models_{\mathsf{SL}} \exists v.\ \phi$	iff	there is $\alpha \in \mathtt{Dat}$ such that $(s,h),\rho[v \mapsto \alpha] \models_{\mathsf{SL}} \phi$
$(s,h),\rho \models_{\mathsf{SL}} x \hookrightarrow y$	iff	there is $\alpha \in \mathtt{Dat}$ such that $h(s(x)) = (s(y), \alpha)$
$(s,h),\rho \models_{\mathsf{SL}} \mathsf{val}(x) \leq v$	iff	there is $\alpha \in \mathtt{Dat}$ and $l \in \mathtt{Loc}$ such that $h(s(x)) = (l, \alpha)$, and $\alpha \leq \rho(v)$
$(s,h),\rho \models_{\mathsf{SL}} \mathsf{val}(x) \geq v$	iff	there is $\alpha \in \mathtt{Dat}$ and $l \in \mathtt{Loc}$ such that $h(s(x)) = (l, \alpha)$, and $\alpha \geq \rho(v)$
$(s,h),\rho \models_{\mathsf{SL}} x = y$	iff	$s(x) = s(y)$
$(s,h),\rho \models_{\mathsf{SL}} \phi_1 * \phi_2$	iff	there are two heaps h_1, h_2 such that $h = h_1 * h_2$ and $(s,h_i),\rho \models_{\mathsf{SL}} \phi_i,\ i = 1,2$
$(s,h),\rho \models_{\mathsf{SL}} \phi_1 \mathrel{-\!\!*} \phi_2$	iff	for all h', if $h \perp h'$ and $(s,h'),\rho \models_{\mathsf{SL}} \phi_1$, then $(s, h * h'),\rho \models_{\mathsf{SL}} \phi_2$

Note that, due to our memory model, the natural semantics of $\mathsf{val}(x) \leq v$ implies in particular $\exists z.x \hookrightarrow z$.

Derived formulae We use standard notations $\vee, \forall, \Rightarrow$, and write $\mathsf{val}(x) = v$, $\mathsf{val}(x) \leq \mathsf{val}(y)$,... for the obvious combinations of comparison predicates. We write $\mathsf{precisely}(A)$ to denote $A \wedge \neg(A * \exists x, y.x \hookrightarrow y)$. We also abbreviate $\phi \mathrel{-\!\!\circledast} \psi$ for the sometimes called septraction connective defined by $\neg(\phi \mathrel{-\!\!*} \neg\psi)$. We use the wildcard notation, e.g. $x \hookrightarrow -$ for $\exists y.x \hookrightarrow y$, the so-called precise predicates $\mapsto$ (e.g. $x \mapsto y$ abbreviates $\mathsf{precisely}(x \hookrightarrow y)$), and equality over vectors $(x_1, .., x_n) =$

$(y_1, .., y_n)$. We will also use the following shorthands:

$$\begin{array}{rl}
x \overset{\leq}{\hookrightarrow} y & \text{for } x \hookrightarrow y \wedge \mathsf{val}(x) \leq \mathsf{val}(y), \text{ and } x \overset{\geq}{\hookrightarrow} y \text{ accordingly,} \\
x \mapsto (y, v) & \text{for } x \mapsto y \wedge \mathsf{val}(x) = v \\
x \hookrightarrow^* y & \text{for } x = y \vee \Big(\top * \big(\ (x \hookrightarrow -) \wedge (- \hookrightarrow y) \wedge \neg(- \hookrightarrow x) \wedge \neg(y \hookrightarrow -) \\
 & \qquad\qquad \wedge \forall z \notin \{x, y\}.\ \big((z \hookrightarrow -) \Leftrightarrow (- \hookrightarrow z)\big)\big)\Big) \\
x \hookrightarrow^+ y & \text{for } \exists z. x \hookrightarrow z \wedge z \hookrightarrow^* y, \\
\mathsf{decls}(x, y) & \text{for } \quad \mathsf{precisely}(x = y) \quad \vee \quad x \mapsto y \\
 & \quad \vee\ \mathsf{precisely}\big(\exists y'.\ x \hookrightarrow^+ y' \ \wedge \ y' \hookrightarrow y \wedge \forall z. (z \hookrightarrow^+ y') \Rightarrow (z \overset{\geq}{\hookrightarrow} -)\big)
\end{array}$$

We christen the $\overset{\leq}{\hookrightarrow}$ predicate *short-distance comparison*, and by contrast refer to $\mathsf{val}(x) \leq \mathsf{val}(y)$ as *long-distance comparison*. The binary predicate $x \hookrightarrow^* y$ is the accessibility relation (see [6]); it asserts that $(\mathsf{fst} \circ h)^n(s(x)) = s(y)$ for some $n \geq 0$.

The binary predicate $\mathsf{decls}(x, y)$ characterises a heap composed of a single list segment with data sorted in the decreasing order.

2.2 A Monadic Second Order Logic over Memory Shapes

Memory shapes We define memory shapes as the abstraction of a memory heap forgetting the whole data component of all cells, while retaining the graphical aspect. A *memory shape* is hence a pair composed of a *store* and a *heap shape*, $(\mathfrak{s}, \mathfrak{h})$ such that

- $\mathfrak{s}$ is a variable valuation of the form $\mathfrak{s} : \mathtt{Var} \rightarrow \mathtt{Loc}$,
- $\mathfrak{h}$ is a partial function $\mathfrak{h} : \mathtt{Loc} \rightharpoonup \mathtt{Loc}$ with finite domain.

We will use the typographic convention to differentiate a memory state (s, h) from a memory shape $(\mathfrak{s}, \mathfrak{h})$. Note that concrete stores can be safely identified to abstract stores. We will write $\mathsf{shape}(.)$ for the obvious map from concrete heaps to heap shapes:

$$\mathsf{shape}(h) \overset{\text{def}}{\equiv} \begin{array}{l} \mathsf{Loc} \rightharpoonup \mathsf{Loc} \\ \ \ l \ \ \mapsto \mathsf{fst}(h(l)) \end{array} \quad \text{with } \ \mathtt{dom}(\mathsf{shape}(h)) = \mathtt{dom}(h)$$

MSO over memory shapes We assume a set $\mathtt{VAR}$ of monadic second-order variables, denoted by $\mathtt{P}, \mathtt{Q}, \mathtt{R}, \ldots$. An *environment* $\mathcal{E}$ is a map $\mathcal{E} : \mathtt{VAR} \rightarrow \mathcal{P}_{fin}(\mathtt{Loc})$ that associates to every second order variable a finite set of locations. Since we require finiteness of models, the version of monadic second-order logic we shall consider is usually called *weak*.

Formulae of (weak) monadic second-order logic ($\mathtt{MSO}$) are defined by the grammar below:

$$\phi := \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, x \hookrightarrow y \,|\, x = y \,|\, \exists \mathtt{P}.\phi \,|\, \mathtt{P}(x)$$

and are interpreted with the expected semantics:

$$\begin{array}{ll}
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \neg\phi & \text{iff not } (\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \wedge \psi & \text{iff } (\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \text{ and } (\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \psi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \exists x.\ \phi & \text{iff there is } l \in \mathtt{Loc} \text{ such that } (\mathfrak{s}[x \mapsto l],\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \phi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} x \hookrightarrow y & \text{iff } \mathfrak{h}(\mathfrak{s}(x)) = \mathfrak{s}(y) \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} x = y & \text{iff } \mathfrak{s}(x) = \mathfrak{s}(y) \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \exists \mathtt{P}.\ \phi & \text{iff there is a finite subset } \mathcal{P} \text{ of } \mathtt{Loc}, \\
 & \quad \text{such that } (\mathfrak{s},\mathfrak{h}),\mathcal{E}[\mathtt{P} \mapsto \mathcal{P}] \models_{\mathtt{MSO}} \phi \\
(\mathfrak{s},\mathfrak{h}),\mathcal{E} \models_{\mathtt{MSO}} \mathtt{P}(x) & \text{iff } \mathfrak{s}(x) \in \mathcal{E}(\mathtt{P})
\end{array}$$

As usual, we will write $\mathtt{P} \subseteq \mathtt{Q}$ for $\forall x.\mathtt{P}(x) \Rightarrow \mathtt{Q}(x)$, $\mathtt{P} \subsetneq \mathtt{Q}$ for $\mathtt{P} \subseteq \mathtt{Q} \wedge \exists x.\mathtt{P}(x) \wedge \neg\mathtt{Q}(x)$, and all set operators $\mathtt{P} \cap \mathtt{Q}, \mathtt{P} \cup \mathtt{Q}$, etc.

The following result is an almost straightforward consequence of the decidability of monadic second-order logic over structures with one function symbol [13] (see also [6] for details):

Theorem 1. *The satisfiability of MSO formulae interpreted over memory shapes is a decidable problem.*

3 Motivations

The merge function, that builds an ordered list from two ordered lists, will be our running example in this section. We consider the following C-like code :

```
struct cell {
  int val;
  struct cell *next;
};

function merge(cell *x, cell *y){
  cell *z, *head;
  if (x==NULL) return y;
  if (y==NULL) return x;
  if ((x->val) >= (y->val)){
    head = x; x=x->next;
  else {
    head = y; y=y->next;
  }
  z= head;
```

```
while((x!=NULL)&&(y!=NULL)) {
  /* MAIN LOOP P */
  if ((x->val) >= (y->val)) {
    z->next = x;
    x = x->next;
  } else {
    z->next = y;
    y = y->next;
  }
  z = z->next;
  /* END OF LOOP P*/
}
[...]
}
```

Let P denote the instruction block of the while loop. In order to prove the merge program, one usually needs at some point to provide a loop invariant A. This invariant may be automatically found, using some acceleration techniques, or might be provided by the user. In both cases, proving that the invariant is preserved is equivalent to showing that

$$\mathbf{Post}(P, A \wedge \mathbf{x} \neq \mathsf{null} \wedge \mathbf{y} \neq \mathsf{null}) \ \Rightarrow\ A \tag{1}$$

is a valid formula, where **Post** denotes the strongest postcondition. There are several ways to compute the strongest postcondition of a loop-free sequence of instructions. We sketch here two approaches: the original one in separation logic theory [14], and the one followed by the tool SMALLFOOT [1].

The original approach does not make any assumption on the invariant A, and fully exploits the magic wand connective. To give an idea, in our case, the post-condition of the loop P of the merge program would look like

$$\begin{array}{l}
\mathbf{Post}(P,B) \;=\; \exists x', y', z'.\ z' \hookrightarrow \mathbf{z} \\
\wedge \mathsf{val}(x') \geq \mathsf{val}(y') \Rightarrow \Big(\quad x' \hookrightarrow \mathbf{x} \ \wedge\ y' = \mathbf{y} \\
\qquad\qquad\qquad\qquad \wedge \Big(\exists v.\ \ z' \mapsto (x', v) * \big(z' \mapsto (-, v) \twoheadrightarrow B\{{}^{x',z'}/_{\mathbf{x},\mathbf{z}}\}\big)\Big)\Big) \\
\wedge \mathsf{val}(x') < \mathsf{val}(y') \Rightarrow \Big(\quad y' \hookrightarrow \mathbf{y} \ \wedge\ x' = \mathbf{x} \\
\qquad\qquad\qquad\qquad \wedge \Big(\exists v.\ \ z' \mapsto (y', v) * \big(z' \mapsto (-, v) \twoheadrightarrow B\{{}^{x',z'}/_{\mathbf{x},\mathbf{z}}\}\big)\Big)\Big).
\end{array}$$

where primed variables quantify over the value of the corresponding program variable before the execution of the loop. What should be underlined concerning this approach is that automatically checking (1) would involve to solve the satisfiability of the logic in presence of magic wand, which is known to be undecidable even with only one selector [6].

In the SMALLFOOT *approach,* on the contrary, the symbolic computation is not parametric in the invariant. Usually, symbolic memory states are represented by (disjunctions of) formulae of the form $\exists x_1, \ldots, x_n . \Pi \wedge \Sigma$, where Σ (the "spatial" part) is a $*$-conjunct of the elementary list segments present in memory, and Π (the "pure" part) handles all other informations that are not properly adressed by local reasoning. For instance, a reasonable loop invariant following this format could be:

$$A \;\overset{\text{def}}{\equiv}\; \exists z_1, z_1' \quad \begin{pmatrix} \quad (\ z_1 = \mathbf{x} \ \vee \ z_1 = \mathbf{y} \ ) \\ \wedge\ z_1' \overset{\geq}{\hookrightarrow} \mathbf{z} \wedge \mathsf{val}(\mathbf{z}) \geq \mathsf{val}(\mathbf{x}) \wedge \mathsf{val}(\mathbf{z}) \geq \mathsf{val}(\mathbf{y}) \end{pmatrix} \\ \wedge\ \mathsf{decls}(\mathtt{head}, \mathbf{z}) * \mathbf{z} \mapsto z_1 * \mathsf{decls}(\mathbf{x}, \mathsf{null}) * \mathsf{decls}(\mathbf{y}, \mathsf{null}) \tag{2}$$

Symbolic computation over lists with values has not been defined in SMALLFOOT, but taking inspiration from it, we may consider that for such an invariant the result of the symbolic computation would look like:

$$\begin{array}{l}
\mathbf{Post}(P, A \wedge \mathbf{x} \neq \mathsf{null} \wedge \mathbf{y} \neq \mathsf{null}) \;\overset{\text{def}}{\equiv}\; \exists z_1', z_2, z_3, z_4, x', y', z'. \\
\begin{pmatrix} \quad ((z_2, z_3, z_4, y') = (x', \mathbf{x}, \mathbf{y}, \mathbf{y}) \vee (z_2, z_3, z_4, x') = (y', \mathbf{y}, \mathbf{x}, \mathbf{x})) \\ \wedge\ \mathbf{z} = z_2 \ \wedge\ z_1' \overset{\geq}{\hookrightarrow} z' \ \wedge\ z_2 \overset{\geq}{\hookrightarrow} z_3 \\ \wedge\ \mathsf{val}(z') \geq \mathsf{val}(x') \wedge \mathsf{val}(z') \geq \mathsf{val}(y') \wedge \mathsf{val}(z_2) \geq \mathsf{val}(z_4) \end{pmatrix} \\
\wedge\ \ \mathsf{decls}(\mathtt{head}, z') * z' \mapsto z_2 * z_2 \mapsto z_3 * \mathsf{decls}(\mathbf{x}, \mathsf{null}) * \mathsf{decls}(\mathbf{y}, \mathsf{null})
\end{array} \tag{3}$$

where again primed variables should be thought as the past values of the corresponding program variables. We may underline that, unlike for (2), there are long-distance comparisons in the pure part of (3) that involve two quantified variables. As we will see in Section 5, this formula belongs to a fragment for which we obtain an undecidability result. However, looking more carefully to it, one may notice that, out of z' and z'_1, all quantified variables are aliased to program variables, which allows to rewrite the formula so that each long-distance comparison involves at least one open variable. Up to that, one may then use our decidability result of Section 5 to automatically check (1).

4 Decidability of Short-Distance Comparisons

In this section, we establish the decidability of the short-distance fragment of $\mathtt{SL}_<$. This fragment is defined by the following grammar:

$$\phi ::= \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi \mid x \hookrightarrow y \mid x \overset{\leq}{\hookrightarrow} y \mid x \overset{\geq}{\hookrightarrow} y \mid x = y \mid \phi * \phi$$
(short-distance fragment)

The decidability of satisfiability for this fragment is obtained by reduction to the satisfiability of $\mathtt{MSO}$ over shapes.

Colored shapes. We hence have to abstract the values taking care of their local comparisons. To do so, we use a colored shape, with three colors on the edges[3]: '<', '>', and '='. In logical terms, these colors will be defined by two second order variables, noted X and Y, and we will observe the color '=' if both X and Y holds for the source location of the edge, '<' if X holds but not Y, and '>' if Y holds but not X. The case where neither X nor Y holds is irrelevant since we assumed a total order on data values, so we should constrain the possible choices for X and Y to avoid this situation. Moreover, some extra constraints will be involved by the necessity to manipulate only colored shapes for which it is possible to assign data respecting the colors (for instance, a cycle of '<' cannot be assigned data).

The graph of constraints. Given a shape $(\mathfrak{s}, \mathfrak{h})$, and the interpretations $\mathcal{X}, \mathcal{Y} \subseteq \mathtt{dom}(\mathfrak{h})$ of the second-order variables mentioned before, we define the associated graph of constraints $G = (V, E)$ where:

- V is $\mathtt{dom}(\mathfrak{h})$ quotiented by the equivalence $l \sim l'$ relating locations connected by a non oriented, '='-labeled path in the colored shape. Note that each $\sim$-equivalence class contains at most one location l whose image under $\mathfrak{h}$ lies outside the equivalence class of l. In such a situation, $[l]$ denotes this equivalence class.
- E is the set of pairs of equivalence classes $([l], [l'])$ such that
  - either $h(l) = l'$ and the color on l is '>'
  - or $h(l') = l$ and the color on l' is '<'

[3] Formally, on vertices, but each edge can be non-ambiguously identified to its source vertex in a shape.

Figure 1 A concrete heap (1), its colored abstraction (2), and the associated graph of constraints (3). Here $\mathcal{X} = \{c, d, f, g\}$ and $\mathcal{Y} = \{a, d, e, f, g\}$.

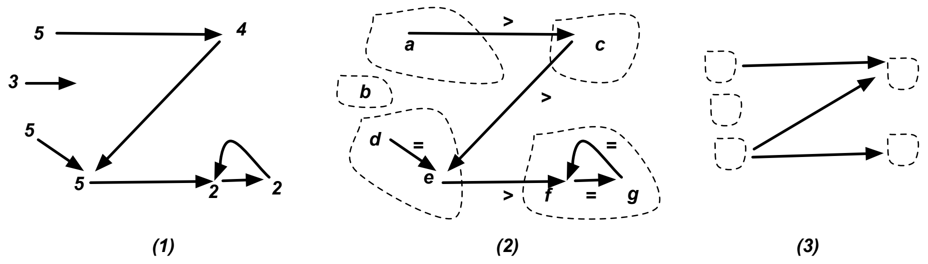

Figure 1 gives an example of a colored shape and its associated graph of constraints. Note that an edge towards a dangling pointer cannot be colored, and this is in fact the unique situation in which one allows $\neg X \wedge \neg Y$. The graph of constraints helps us to decide whether or not it is possible to assign values to a colored shape: indeed, this problem is equivalent to defining a topological order on the graph of constraints, which is known to be equivalent to this graph being acyclic. What remains to be explained now is: (1) how to define the graph of constraints in MSO, (2) how to express acyclicity, (3) how to treat separating conjunction.

The reduction. The reduction from $\texttt{SL}_<$ over memory states to $\texttt{MSO}$ over shapes is defined by $\mathsf{rd}_{\texttt{SL}_< \to \texttt{MSO}}(\phi) = \exists X.\exists Y.\exists Z.\mathrm{Cons}(X, Y, Z) \wedge \mathsf{rd}(\phi, X, Y, Z)$ where:

- Z is an extra second-order variable that is needed to define the current focus, that is the subheap of the original heap on which the (sub)formula is currently evaluated.
- rd is an auxiliary reduction that works assuming that X, Y and Z have been correctly guessed, updating these parameters appropriately when $*$ is translated.
- Cons are constraints imposed on X, Y and Z to guarantee that the first guess is a valid one: $\mathcal{Z}$ is the domain of the heap, and $\mathcal{X}$ and $\mathcal{Y}$ define a colored shape to which one may assign values.

Constraints. We impose three contraints : $\mathrm{Cons}(X, Y, Z) \overset{\mathrm{def}}{\equiv} \mathrm{Cons1}(X, Y, Z) \wedge \mathrm{Cons2}(X, Y, Z) \wedge \mathrm{Cons3}(X, Y, Z)$

1. the only admitted color on a monochromatic cycle is '=' (this is indeed equivalent to the acyclicity condition on the graph of constraints):

$$\mathrm{Cons1}(X, Y, Z) \quad \overset{\mathrm{def}}{\equiv} \quad \forall U \subseteq Z.\ \ \mathrm{Loop}(U) \ \Rightarrow\ \ (U \subseteq X \Leftrightarrow U \subseteq Y)$$

where $\mathrm{Loop}(U)$ is defined as $\mathrm{SetOfLoops}(U) \wedge \forall V \subsetneq U. \neg\mathrm{SetOfLoops}(V)$ and $\mathrm{SetOfLoops}(U)$ is $\forall x.U(x) \Rightarrow \exists y.U(y) \wedge y \hookrightarrow x$

2. every edge that should be colored is colored with '<, '>' or '='

$$\mathrm{Cons2}(X,Y,Z) \overset{\text{def}}{\equiv} \forall x.\ \ (Z(x) \wedge (\exists y.Z(y) \wedge x \hookrightarrow y)) \ \Leftrightarrow\ (X(x) \vee Y(x))$$

3. $\mathcal{Z}$ is the domain of the heap: $\mathrm{Cons3}(X,Y,Z) \overset{\text{def}}{\equiv} \forall x.(x \hookrightarrow -) \ \Leftrightarrow\ Z(x)$.

Let us now state the results we may derive from these definitions. We say that a location l is an increasing (resp. decreasing) node if there are $l', l'' \in \mathtt{Loc}$ and $\alpha, \beta \in \mathtt{Dat}$ such that $h(l) = (l', \alpha)$, $h(l') = (l'', \beta)$, and $\alpha \leq \beta$ (resp. $\alpha \geq \beta$). We write $\mathtt{dom}^+(h)$ (resp. $\mathtt{dom}^-(h)$) to denote the set of increasing (resp. decreasing) nodes of h, and $\mathcal{E}_h$ denotes $[X \mapsto \mathtt{dom}^+(h),\ Y \mapsto \mathtt{dom}^-(h),\ Z \mapsto \mathtt{dom}(h)]$.

Lemma 1 (Constraints soundness). *If $(\mathfrak{s}, \mathfrak{h}), \mathcal{E} \models_{\mathtt{MSO}} \mathrm{Cons}(X,Y,Z)$ then there is a $h : \mathsf{Loc} \rightharpoonup \mathsf{Loc} \times \mathsf{Dat}$ such that $\mathsf{shape}(h) = \mathfrak{h}$, $\mathcal{E}(Z) = \mathtt{dom}(h)$, $\mathcal{E}(X) = \mathtt{dom}^+(h)$ and $\mathcal{E}(Y) = \mathtt{dom}^-(h)$.*

Lemma 2 (Constraints completeness). *For all models with data (s,h):*

$$(s, \mathsf{shape}(h)), \mathcal{E}_h \models_{\mathtt{MSO}} \mathrm{Cons}(X,Y,Z).$$

Auxiliary recursive translation. The auxiliary recursive translation rd is defined as follows: (1) it is isomorphic on the cases of $\phi \wedge \psi$, $\neg\phi$, $\exists x.\phi$, and $x = y$, and (2) for other connectives, parameters X, Y, Z come into play:

$$\begin{aligned}
&\mathsf{rd}(x \hookrightarrow y, X, Y, Z) \overset{\text{def}}{\equiv} Z(x) \wedge x \hookrightarrow y\\
&\mathsf{rd}(x \overset{\leq}{\hookrightarrow} y, X, Y, Z) \ \ \overset{\text{def}}{\equiv} Z(x) \wedge Z(y) \wedge X(x) \wedge x \hookrightarrow y\\
&\mathsf{rd}(x \overset{\geq}{\hookrightarrow} y, X, Y, Z) \ \ \overset{\text{def}}{\equiv} Z(x) \wedge Z(y) \wedge Y(x) \wedge x \hookrightarrow y\\
&\mathsf{rd}(\phi_1 * \phi_2, X, Y, Z) \overset{\text{def}}{\equiv} \exists Z_1, Z_2.\\
&\qquad \mathsf{rd}(\phi_1, X, Y, Z_1) \ \wedge\ \mathsf{rd}(\phi_2, X, Y, Z_2) \ \wedge\ Z = Z_1 \cup Z_2 \ \wedge\ Z_1 \cap Z_2 = \emptyset
\end{aligned}$$

Lemma 3 (Reduction Lemma). *For all s, h, for all $\mathcal{Z} \subseteq \mathtt{dom}(h)$,*

$$(s, \mathsf{shape}(h)), \mathcal{E}_h[Z \mapsto \mathcal{Z}] \models_{\mathtt{MSO}} \mathsf{rd}(\phi, X, Y, Z) \text{ if and only if } (s, h_{|\,\mathcal{Z}}) \models_{\mathtt{SL}} \phi.$$

Theorem 2. *For all formulae ϕ of $\mathtt{SL}_<$, there exists (s,h) such that $(s,h) \models_{\mathtt{SL}} \phi$ if and only if there exists $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathtt{MSO}} \mathsf{rd}_{\mathtt{SL}_< \to \mathtt{MSO}}(\phi)$*

Thanks to Theorem 2 and Theorem 1, we have established the announced result:

Corollary 1. *The satisfiability problem for the fragment of $\mathtt{SL}_<$ with short-distance comparisons is decidable.*

5 Long-Distance Comparisons

5.1 An Undecidablity Result

We consider now the fragment of $\mathtt{SL}_<$ where magic wand is still dropped, but long-distance comparison is considered:

$$\phi ::= \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, \exists v.\phi \,|\, x \hookrightarrow y \,|\, \mathsf{val}(x) \leq v \,|\, \mathsf{val}(x) \geq v \,|\, x = y \,|\, \phi * \phi.$$
(long-distance fragment)

We show that, without any further restriction, long-distance comparisons yield undecidability, even for a simpler fragment:

$$\phi ::= \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, x \hookrightarrow y \,|\, \mathsf{val}(x) = \mathsf{val}(y) \,|\, x = y \,|\, \phi * \phi.$$
(equality long-distance fragment)

Theorem 3. *The satisfiability problem for the equality long-distance fragment is undecidable.*

The proof goes by reduction to the satisfiability problem of first-order formulae over data words. Before giving the intuition of the reduction, we first recall this logic.

First-order logic over data words We assume a finite set Σ. A finite *data word* is a sequence $w = w_1..w_n$, where $w_i = (a_i, \alpha_i) \in \Sigma \times \mathtt{Dat}$; we write $|\,w\,|$ to denote the length $n \in \mathbb{N}$ of w. Note that, so far, we assumed a total order on $\mathtt{Dat}$, but this aspect is not essential for this reduction, and one may think of $\mathtt{Dat}$ as any arbitrary infinite set. The first-order formulae we will evaluate over these models are defined by the following grammar:

$$\text{(FO over data words)} \quad \phi ::= \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, a(x) \,|\, x = y + 1 \,|\, x \sim_{\mathtt{Dat}} y$$

where $a \in \Sigma$. Variables are interpreted as positions in the word through a valuation $\sigma : \mathtt{Var} \to \{1..\,|\,w\,|\}$, $+1$ is the standard addition over $\mathbb{N}$, and $\sim_{\mathtt{Dat}}$ relates positions holding the same datum. More formally

$$\begin{array}{ll}
w, \sigma & \models_{FO} \exists x.\phi \text{ if there is } n \in \{1..\,|\,w\,|\} \text{ s.t. } w, \sigma[x \mapsto n] \models_{FO} \phi \\
w, \sigma & \models_{FO} a(x) \text{ if } a_{\sigma(x)} = a \\
w, \sigma & \models_{FO} x = y + 1 \text{ if } \sigma(x) = \sigma(y) + 1 \\
w, \sigma & \models_{FO} \; x \sim_{\mathtt{Dat}} y \text{ if } \alpha_{\sigma(x)} = \alpha_{\sigma(y)}
\end{array}$$

Theorem 4 (see [2], Prop. 27). *The satisfiability problem for a closed sentence of FO over data words is undecidable.*

The reduction To prove Theorem 3, we define a translation from FO to the long-distance fragment such that a formula ϕ admits a data word model if and only if its translation admits a memory state model. A data word of length n is encoded as a list segment of length $2n$, placing the sequence of letters of Σ in the even positions, and the data sequence in odd positions. Then $x = y + 1$ can be encoded by $y \hookrightarrow^2 x$, and $x \sim_{\mathtt{Dat}} y$ can be encoded by $\mathsf{val}(x) = \mathsf{val}(y)$.

5.2 Decidability of Guarded Long-Distance Comparisons

We now consider the fragment of formulae where every quantification over values is restricted to values stored in the cells that are pointed by the program variables:

$$\begin{aligned}\phi ::= {} & \neg\phi \mid \phi \wedge \phi \mid \exists x.\phi \mid \exists v.\mathsf{val}(\mathbf{x}) = v \wedge \phi \\ & \mid x \overset{\leq}{\hookrightarrow} y \mid x \overset{\geq}{\hookrightarrow} y \mid x \hookrightarrow y \mid \mathsf{val}(x) \leq v \mid \mathsf{val}(x) \geq v \mid x = y \mid \phi * \phi.\end{aligned}$$

(guarded long-distance fragment)

Note that guarded long-distance comparisons are quite weak, and we need to add short-distance comparisons as basic predicates if we still want to use them.

Theorem 5. *The satisfiability problem for the guarded long-distance fragment is decidable.*

Proof of Theorem 5 We only sketch the proof. We adapt the proof of Theorem 2 by extending the notions of colored shapes and graphs of constraints. Let $\mathtt{ProgVar} = \{\mathbf{x}_1, .., \mathbf{x}_n\} \subsetneq \mathtt{Var}$ be a finite set of variables such that every formula to be translated will have all its open variables in $\mathtt{ProgVar}$. To every variable $\mathbf{x} \in \mathtt{ProgVar}$, we associate two second-order variable $X_\mathbf{x}, Y_\mathbf{x}$. A colored shape is then a tuple

$$CS \;=\; \Big((\mathfrak{s}, \mathfrak{h}) \;,\; \mathcal{X}, \mathcal{Y},\; \mathcal{X}_{\mathbf{x}_1}, \mathcal{Y}_{\mathbf{x}_1}, \ldots, \mathcal{X}_{\mathbf{x}_n}, \mathcal{Y}_{\mathbf{x}_n} \Big)$$

where $\mathcal{X}_\mathbf{x}, \mathcal{Y}_\mathbf{x}$ are finite sets of locations; it is *well defined* if $\mathcal{X} \cup \mathcal{Y} = \mathtt{dom}(h) \cap h^{-1}(\mathtt{dom}(h))$ and $\mathcal{X}_\mathbf{x} \cup \mathcal{Y}_\mathbf{x} = \mathtt{dom}(h)$ for every program variable $\mathbf{x}$ such that $\mathfrak{s}(\mathbf{x}) \in \mathtt{dom}(h)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a fixed shape. We define the relation $\sim$ on $\mathtt{dom}(\mathfrak{h})$ as the smallest equivalence relation such that:

- if $l \in \mathcal{X}_\mathbf{x} \cap \mathcal{Y}_\mathbf{x}$ and $\mathfrak{s}(\mathbf{x}) \in \mathtt{dom}(\mathfrak{h})$, then $\mathfrak{s}(\mathbf{x}) \sim l$;
- if $\mathfrak{h}(l) = l'$, and $l \in \mathcal{X} \cap \mathcal{Y}$, then $l \sim l'$.

The graph of constraints associated to CS is the pair (V, E) where the vertex set V is the quotient of $\mathtt{dom}(\mathfrak{h})$ by $\sim$, and there is an edge from the equivalence class c_1 to c_2 if at least one of the following conditions holds:

- either there is $\mathfrak{s}(\mathbf{x}) \in c_1$ and $l \in c_2$ such that $l \in \mathcal{Y}_\mathbf{x} - \mathcal{X}_\mathbf{x}$;
- or there is $\mathfrak{s}(\mathbf{x}) \in c_2$ and $l \in c_1$ such that $l \in \mathcal{X}_\mathbf{x} - \mathcal{Y}_\mathbf{x}$;
- or there is $l \in c_1, l' \in c_2$ such that $\mathfrak{h}(l) = l'$ and $l \in \mathcal{Y} - \mathcal{X}$;
- or there is $l \in c_1, l' \in c_2$ such that $\mathfrak{h}(l') = l$ and $l' \in \mathcal{X} - \mathcal{Y}$.

It is possible to check that the graph of constraints and the acyclicity condition on it are MSO definable. We may then adapt the reduction of Section 4: we guess the $\mathcal{X}_\mathbf{x}$s and $\mathcal{Y}_\mathbf{x}$s at start and check we made a valid guess, and we extend the recursive translation $\mathsf{rd}(\phi)$ with the following cases:

$$\begin{aligned}
\mathsf{rd}(\exists v.\mathsf{val}(\mathbf{x}) = v \wedge \phi) &\overset{\text{def}}{\equiv} \mathsf{rd}(\phi\{^{\mathsf{val}(\mathbf{x})}/_{v}\}) \\
\mathsf{rd}(\mathsf{val}(x) \leq \mathsf{val}(\mathbf{x})) &\overset{\text{def}}{\equiv} Z(x) \wedge Z(\mathbf{x}) \wedge Y_\mathbf{x}(x) \wedge \neg X_\mathbf{x}(x) \\
\mathsf{rd}(\mathsf{val}(x) \geq \mathsf{val}(\mathbf{x})) &\overset{\text{def}}{\equiv} Z(x) \wedge Z(\mathbf{x}) \wedge X_\mathbf{x}(x) \wedge \neg Y_\mathbf{x}(x)
\end{aligned}$$

Perspectives We expect this decidability result to extend to more complex data structures that would have a decidable MSO theory (trees, doubly-linked lists, lists of lists, and more generally tree-width bounded structures), and to more complex short-distance comparisons (n-th successor, brothers,...). Moreover, such restrictions may be sufficient to handle other interesting applications, for instance search-trees. In this sense, we claim that the graph of constraints is the "right" general concept for logics dealing with sorted data structures.

6 Magic Wand and Restricted Magic Wand

Even without data, the logic with the operator $-\!\!*$ is proved to be undecidable in [6]. In the technical report [5] corresponding to the paper, a decidable separation logic with a restricted magic wand is presented. Let us write the definition of this binary operator, $-\!\!*_n$ (for n an integer). Unlike the plain operator $-\!\!*$, the quantification on disjoint heaps of $-\!\!*_n$ considers only heaps for which the cardinality of the domain is bounded by n. More formally, we define that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 -\!\!*_n \phi_2$ if and only if for all $\mathfrak{h}'$ such that $\mathfrak{h}' \perp \mathfrak{h}$ and $|\,\mathtt{dom}(\mathfrak{h}')\,| \leq n$, if $(\mathfrak{s}, \mathfrak{h}') \models \phi_1$ then $(\mathfrak{s}, \mathfrak{h} * \mathfrak{h}') \models \phi_2$. It can be seen as an abbreviation of $(\phi_1 \wedge \neg\exists x_1, \ldots, x_{n+1}. \bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_i \exists y. x_i \hookrightarrow y) -\!\!* \phi_2$. In the sequel, we will prove that, in the context of heaps with data, $-\!\!*_1$ is sufficient to obtain undecidability.

Let R denote an arbitrary binary relation on $\mathtt{Dat}$. Let us call $\sim_R$ the equivalence relation defined as $\alpha \sim_R \alpha'$ iff $\{\beta, \beta R R \alpha\} = \{\beta, \beta R R \alpha'\}$. We consider the atomic formula $\mathsf{val}(x) R \mathsf{val}(y)$ stating that values stored in x and y compare through R. Formally, $(s, h) \models_{\mathtt{SL}} \mathsf{val}(x) R \mathsf{val}(y)$ iff there are $\alpha, \beta \in \mathtt{Val}$ and $l, l' \in \mathtt{Loc}$ such that $h(s(x)) = (l, \alpha)$, $h(s(y)) = (l', \beta)$, and $\alpha R \beta$.

We now introduce the relation $x \overset{R}{\hookrightarrow} y$ for $x \hookrightarrow y \wedge \mathsf{val}(x) R \mathsf{val}(y)$, and define the logic $\mathtt{SL}(R, -\!\!*_1)$ with the grammar:

$$\phi ::= \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, x \hookrightarrow y \,|\, x \overset{R}{\hookrightarrow} y \,|\, x = y \,|\, \phi * \phi \,|\, \phi -\!\!*_1 \phi.$$

We are going to prove that satisfiability and validity problems are undecidable for $\mathtt{SL}(R, -\!\!*_1)$, for any $R \in \{\leq, \geq, =, <, >\}$. We will rely on the previous section, especially Theorem 3, by simulating a long-distance equality. We first need the following fact:

Lemma 4. *Let $R \in \{\leq, \geq, =, <, >\}$. Then $\sim_R$ has an infinite number of equivalence classes.*

Let $\sim$ be an equivalence relation on $\mathtt{Dat}$ with infinitely many equivalence classes. Let us define the following fragment:

$$\phi ::= \neg\phi \,|\, \phi \wedge \phi \,|\, \exists x.\phi \,|\, x \hookrightarrow y \,|\, \mathsf{val}(x) \sim \mathsf{val}(y) \,|\, x = y \,|\, \phi * \phi.$$
(equivalence long-distance fragment)

Then the following lemma, a slight variation of Theorem 3, also holds in this generalised framework:

Lemma 5. *The satisfiability problem for the equivalence long-distance fragment is undecidable.*

Proof By the same encoding as the one of Theorem 3, one may reduce a satisfiability problem of an FO sentence over data words, where data taken from the infinite quotient set $\mathtt{Dat}/\sim_{\mathtt{R}}$, to the satisfiability problem for the equivalence long-distance fragment. □

Lemma 6. *There is a formula $\phi_R(x, x') \in \mathtt{SL}(R, \multimap_1)$ such that for all (s, h) with $\{s(x), s(x')\} \subseteq \mathtt{dom}(h)$:*

$$(s, h) \models \phi_R(x, x') \textit{ iff } (s, h) \models \mathsf{val}(x) \sim_R \mathsf{val}(x')$$

We only sketch the proof. $\phi \overline{\multimap}_1 \psi$ will abbreviate $\neg(\phi \multimap_1 \neg\psi)$. Then $(s, h) \models \phi \overline{\multimap}_1 \psi$ iff there is h' such that $(s, h') \models \phi$, $(s, h * h') \models \psi$ and $|\,\mathtt{dom}(\mathfrak{h}')\,| \leq 1$. The operators $\multimap_1$ and $\overline{\multimap}_1$ will be used to simulate restricted quantifications over $\mathtt{Dat}$, respectively universal and existential. Consider the formula ϕ

$$\begin{gathered}\exists x_1.\exists x_2.\big(\neg\exists x_3.x_1 \hookrightarrow x_3 \vee x_2 \hookrightarrow x_3\big) \\ \wedge(x_1 \hookrightarrow x_2) \multimap_1 \big((\mathsf{val}(x_1)\ RR\ \mathsf{val}(x)) \quad \Leftrightarrow \quad (\mathsf{val}(x_1)\ RR\ \mathsf{val}(x'))\big)\end{gathered}$$

where $\mathsf{val}(x_1) RR \mathsf{val}(x)$ abbreviates $(x_2 \hookrightarrow x) \overline{\multimap}_1 [(x_1 \overset{R}{\hookrightarrow} x_2) \wedge x_2 \overset{R}{\hookrightarrow} x]$. The formula ϕ expresses that for all α, there is β such that $\alpha R \beta R \mathsf{snd}(h(s(x)))$ if and only if there is β such that $\alpha R \beta R \mathsf{snd}(h(s(x')))$, that is $\mathsf{val}(x) \sim_R \mathsf{val}(x')$. As a consequence:

Theorem 6. *For any $R \in \{\leq, \geq, <, >, =\}$, the validity and satisfiability problems for $\mathtt{SL}(R, \multimap_1)$ are undecidable.*

7 Conclusion

Our results give a wide picture of the decidability status of the satisfiability problem for separation logic dealing with data.

With the ability to describe lists and quantify over locations, allowing long-distance comparisons brings undecidability, and so does allowing the operator $\multimap$, even strongly restricted. Yet, there is a very positive result: dropping these two features makes the satisfiability problem decidable, still being able to do local reasoning and express properties about ordered recursive structures. The decidability even holds when a finite set of references can be compared to all the rest of the memory.

Some ways to restrict the full language are still unexplored, for instance bounding the amount of quantified variables. With the same hope to obtain decidability for satisfiability problems, one may look at extension of our decidable fragment. For instance, our results are general for any totally ordered infinite set, and questions remain open about partially ordered sets.

References

1. J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO'05*, volume 4111 of *LNCS*, pages 115–137. Springer, 2005.
2. Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *Proceedings of 21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA*, pages 7–16, 2006.
3. A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In *CAV'06*, volume 4144 of *LNCS*, pages 517–531. Springer, 2006.
4. M. Bozga, R. Iosif, and Y. Lakhnech. On logics of aliasing. In *SAS'04*, volume 3148 of *LNCS*, pages 344–360. Springer, 2004.
5. R. Brochenin, S. Demri, and E. Lozes. On the almighty wand. Technical report, LSV, ENS de Cachan, 2008.
6. Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. In Michael Kaminski and Simone Martini, editors, *Proceedings of the 16th Annual EACSL Conference on Computer Science Logic (CSL'08)*, volume 5213 of *Lecture Notes in Computer Science*, Bertinoro, Italy, September 2008. Springer. To appear.
7. C. Calcagno, H. Yang, and P. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FST&TCS'01*, volume 2245 of *LNCS*, pages 108–119. Springer, 2001.
8. Stéphane Demri, Ranko Lazić, and David Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. In *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 113–121, Burlington, Vermont, USA, June 2005. IEEE Computer Society Press.
9. J. Jensen, M. Jorgensen, N. Klarlund, and M. Schwartzbach. Automatic verification of pointer programs using monadic second-order logic. In *PLDI'97*, pages 226–236. ACM, 1997.
10. T. Lev-Ami and M. Sagiv. TVLA: A system for implementing static analyses. In *SAS'00*, volume 1824 of *LNCS*, pages 280–301. Springer, 2000.
11. A. Loginov, T. Reps, and M. Sagiv. Refinement-based verification for possibly-cyclic lists. In *Program Analysis and Compilation, Theory and Practice: Essays Dedicated to Reinhard Wilhelm*. Springer-Verlag, 2007.
12. Huu Hai Nguyen1, Cristina David, Shengchao Qin, and Wei-Ngan Chin. Automated verification of shape and size properties via separation logic. In *Verification, Model Checking, and Abstract Interpretation 2007*, volume 4349/2007 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2007.
13. M. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 41:1–35, 1969.
14. J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.
15. B.A. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Dokl. Akad. Nauk SSSR*, 70:596–572, 1950. English translation in: AMS Transl. Ser. 2, vol.23 (1063), 1–6.
16. G. Yorsh, A. M. Rabinovich, M. Sagiv, A. Meyer, and A. Bouajjani. A logic of reachable patterns in linked data structures. In *FOSSACS'05*, volume 3441 of *LNCS*, pages 94–110. Springer, 2005.