Draft of paper to appear at LICS 2014

Expressive Completeness of Separation Logic With Two Variables and No Separating Conjunction [*]

Stéphane Demri
New York University, USA & CNRS, France
demri@cs.nyu.edu

Morgan Deters
New York University, USA
mdeters@cs.nyu.edu

Abstract

We show that first-order separation logic with one record field restricted to two variables and the separating implication (no separating conjunction) is as expressive as weak second-order logic, substantially sharpening a previous result. Capturing weak second-order logic with such a restricted form of separation logic requires substantial updates to known proof techniques. We develop these, and as a by-product identify the smallest fragment of separation logic known to be undecidable: first-order separation logic with one record field, two variables, and no separating conjunction.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic

Keywords separation logic, expressive completeness

1. Introduction

Expressive completeness The literature is rich with results comparing the expressive power of non-classical logics with first-order or second-order logic. For instance, the celebrated Kamp's Theorem amounts to stating that linear-time temporal logic (LTL) is equal in expressive power to first-order logic. More generally, we know the expressive completeness of Stavi connectives for general linear time, see e.g. [16]. This has been refined to the restriction to two variables, leading to the equivalence between unary LTL and FO2, see e.g. [14, 30]. In addition, there is a wealth of results relating first-order logic with two variables and non-classical logics, providing a neat characterization of the expressive power of many formalisms since first-order logic and second-order logic are queen logics. For instance, Boolean modal logic with converse and identity is as expressive as FO2 [26]. In the realm of interval temporal logics, we also know expressive completeness of metric propositional neighborhood logic with respect to the two-variable fragment of first-order logic for linear orders with successor function, interpreted over natural numbers [5].

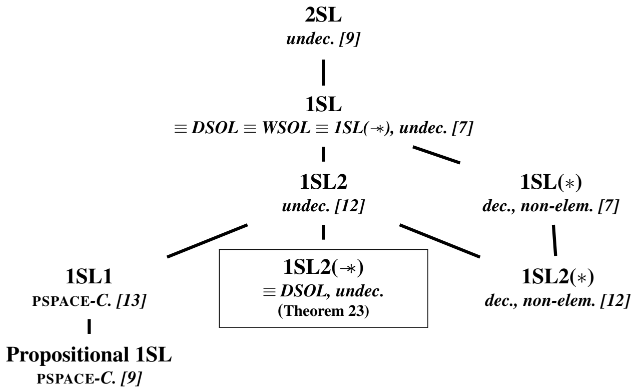
[*] Work partially supported by the EU Seventh Framework Programme (under grant PIOF-GA-2011-301166, DATAVERIF), the Air Force Office of Scientific Research (under award FA9550-09-1-0596), and the National Science Foundation (under grant 0644299).

In this paper, we compare separation logic restricted to two variables with (weak) second-order logic over concrete heaps.

Expressive power of separation logic Separation logic is used as an assertion language for Hoare-style proof systems about programs with pointers [28], and there is an ongoing quest for understanding its complexity and expressive power. Alternatively, there are a lot of activities to develop verification methods with decision procedures for fragments of practical use, see e.g. [10]. Many decision procedures have been designed for fragments of separation logics or abstract variants, from analytic methods [17, 20] to translation to theories handled by SMT solvers [27], passing via graph-based algorithms [19].

Theoretical issues for separation logic stem from the design of expressive fragments with relatively low complexity (see e.g. [10]) to the extension of known decidability results, see e.g. [2, 4, 22]. Indeed, it is known since [9] that first-order separation logic with two record fields (called herein 2SL) is undecidable (with a proof that does not require separating connectives and uses Trakhtenbrot's Theorem). This is sharpened in [7] by showing that first-order separation logic with a unique record field (called herein 1SL) is also undecidable, as a consequence of the expressive equivalence between 1SL and weak second-order logic. More recently, 1SL restricted to two variables (1SL2) is shown undecidable too [12] but without touching the central question of expressive completeness—the purpose of the current paper. From the very beginning, the relationships between separation logic and second-order logic have been quite puzzling (see e.g. an interesting answer with infinite arbitrary structures in [23]). In this paper, we go one step further by showing that two variables suffice to get expressive completeness. As a consequence, 1SL2($-\!\!*$) (that is, 1SL2 without separating conjunction) is undecidable too, which should not be confused with undecidability results from [8, 24] obtained in an alternative setting with propositional variables and no first-order quantification. It is fair to recall that separating implication has been less well-studied than separating conjunction in the literature, but its use for program verification is far more recognized nowadays; see e.g. [25, Section 1] for a recent, insightful analysis (see also [20, Section 8]).

Our contribution In this paper, we sharpen the main result in [7], namely we show that first-order separation logic with one record field, two quantified variables, and no separating conjunction is as expressive as weak second-order logic on heaps; in short, 1SL2($-\!\!*$) $\equiv$ WSOL. Even though conjectured in [6, 7], it is surprising that two variables suffice, and that further we are able to drop the separating conjunction, thus obtaining expressive completeness and undecidability with only two variables and the magic wand operator. In doing so, we improve previous undecidability results about separation logic [7, 9, 12]. Because we forbid ourselves the use of many syntactic resources, this underlines even further the power of the magic wand. By way of comparison with [18, 21], we show undecidability of a two-variable logic with second-order features, and our main undecidability result cannot be derived from [18, 21] since in 1SL models, we deal with a single functional binary relation. We believe that we have identified the core of separation logic as far as undecidability and expressive completeness are concerned, since, for instance, first-order separation logic with one record field, one quantified variable and an unbounded number of program variables, has recently been shown decidable and PSPACE-complete [13]. Below, we illustrate how the main result of the paper (Theorem 23) compares with known results from the literature (Section 2 contains definitions of the different logics).



In our proof, most of the difficulties are concentrated on the use of only two variables: we recycle variables as done for modal logics [15], but this is insufficient, especially when separating conjunction is also banished. So, as far as the proof for expressive completeness goes, we borrow some of the first principles from [7], but very quickly we are faced with serious problems when we need to identify in some heap at least $k > 0$ heap patterns (a typical example is to specify that at least $k > 0$ locations point to a given location).

Indeed, the standard way to identify such patterns is to use an unbounded number of variables or the separating conjunction. So, in the presence of only two variables and by using only the magic wand operator, instead of chopping the heap in k disjoint subheaps, we add $\mathcal{O}(k)$ new patterns so that the newly combined heap satisfies structural properties that witness the presence of the k patterns in the original heap. This high-level description has to be instantiated as many times as we have to identify different types of patterns, but this new point of view allows us to go far beyond what is known today (see e.g., the proof of Lemma 5). At times, it is not strictly necessary to introduce a radically new method, but instead we can be more thrifty in the way formulae are defined to express desirable properties; of course, this may come with more complex proofs and, above all, more ingenuity to design such formulae. It should be noted that the paper is structured in such a way that we provide more and more complex building blocks to establish our main results. Another contribution rests on the fact that we considerably simplify some of the technical insights borrowed from [7] and therefore the current paper proposes a self-contained proof of the equivalence between 1SL2($-\!\!*$) and weak second-order logic that in many ways is much simpler than what has been done so far, even though our results are stronger. Extensions with program variables or with heaps having $k > 1$ record fields are presented in Section 5.5.

2. Preliminaries

2.1 First-order separation logic with one selector (1SL)

A *heap* $\mathfrak{h}$ is a partial function $\mathfrak{h} : \mathbb{N} \rightharpoonup \mathbb{N}$ with finite domain. We write dom($\mathfrak{h}$) to denote its *domain* and ran($\mathfrak{h}$) to denote its *range*. Two heaps $\mathfrak{h}_1, \mathfrak{h}_2$ are said to be *disjoint*, if their domains are disjoint; when this holds, we write $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ to denote the disjoint union. *Locations* are elements of $\mathbb{N}$ and are denoted by $\mathfrak{l}$, possibly decorated with exponents or subscripts. We write $\mathfrak{l}_1 \to \mathfrak{l}_2 \to \cdots \to \mathfrak{l}_m$ to mean that for every $i \in [1, m-1]$, $\mathfrak{h}(\mathfrak{l}_i) = \mathfrak{l}_{i+1}$. In that case $\{\mathfrak{l}_1, \ldots, \mathfrak{l}_{m-1}\} \subseteq \mathrm{dom}(\mathfrak{h})$. We write $\widetilde{\sharp}\mathfrak{l}$ to denote the cardinal of the set $\{\mathfrak{l}' \in \mathbb{N} : \mathfrak{h}(\mathfrak{l}') = \mathfrak{l}\}$ made of *predecessors* of $\mathfrak{l}$ (heap $\mathfrak{h}$ is implicit in the expression $\widetilde{\sharp}\mathfrak{l}$) and $\widetilde{\sharp}\mathfrak{l}^{\star}$ to denote the cardinal of $(\{\mathfrak{l}' \in \mathbb{N} : \mathfrak{h}(\mathfrak{l}') = \mathfrak{l}\} \setminus \{\mathfrak{l}\})$. So, $\widetilde{\sharp}\mathfrak{l} = \widetilde{\sharp}\mathfrak{l}^{\star}$ iff (either $\mathfrak{l} \notin \mathrm{dom}(\mathfrak{h})$ or $\mathfrak{h}(\mathfrak{l}) \neq \mathfrak{l}$).

Let $\mathtt{FVAR} = \{\mathtt{u}_1, \mathtt{u}_2, \ldots\}$ be a countably infinite set of variables. Formulae of 1SL are defined by the grammar $\phi ::= \mathtt{u}_i = \mathtt{u}_j \mid \mathtt{u}_i \hookrightarrow \mathtt{u}_j \mid \phi \wedge \phi \mid \neg\phi \mid \phi * \phi \mid \phi -\!\!* \phi \mid \exists\, \mathtt{u}_i\, \phi$. The connective $*$ is called the *separating conjunction* and $-\!\!*$ is called the *separating implication* (also known as the *magic wand*). We make use of standard notations for the derived connectives.

A *valuation* is a map $\mathfrak{f} : \mathtt{FVAR} \to \mathbb{N}$. The satisfaction relation $\models$ is parameterized by valuations and is defined as follows (Boolean clauses are omitted):

- $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{u}_i = \mathtt{u}_j$ iff $\mathfrak{f}(\mathtt{u}_i) = \mathfrak{f}(\mathtt{u}_j)$.
- $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{u}_i \hookrightarrow \mathtt{u}_j$ iff $\mathfrak{f}(\mathtt{u}_i) \in \mathrm{dom}(\mathfrak{h})$ and $\mathfrak{h}(\mathfrak{f}(\mathtt{u}_i)) = \mathfrak{f}(\mathtt{u}_j)$.
- $\mathfrak{h} \models_{\mathfrak{f}} \phi_1 * \phi_2$ iff there exist $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are disjoint, $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{h}_1 \models_{\mathfrak{f}} \phi_1$ and $\mathfrak{h}_2 \models_{\mathfrak{f}} \phi_2$.
- $\mathfrak{h} \models_{\mathfrak{f}} \phi_1 -\!\!* \phi_2$ iff for all $\mathfrak{h}'$, if $\mathfrak{h}$ and $\mathfrak{h}'$ are disjoint, and $\mathfrak{h}' \models_{\mathfrak{f}} \phi_1$ then $\mathfrak{h} \uplus \mathfrak{h}' \models_{\mathfrak{f}} \phi_2$.
- $\mathfrak{h} \models_{\mathfrak{f}} \exists\, \mathtt{u}_i\, \phi$ iff there is $\mathfrak{l} \in \mathbb{N}$ such that $\mathfrak{h} \models_{\mathfrak{f}[\mathtt{u}_i \mapsto \mathfrak{l}]} \phi$.

We also introduce so-called *septraction* operator $\overset{-}{-\!\!*}$: $\phi \overset{-}{-\!\!*} \psi$ is defined as $\neg(\phi -\!\!* \neg\psi)$. So, $\mathfrak{h} \models_{\mathfrak{f}} \phi \overset{-}{-\!\!*} \psi$ iff there is $\mathfrak{h}'$ disjoint from $\mathfrak{h}$ such that $\mathfrak{h}' \models_{\mathfrak{f}} \phi$ and $\mathfrak{h} \uplus \mathfrak{h}' \models_{\mathfrak{f}} \psi$. Septraction states the existence of a disjoint heap satisfying a formula and for which its addition to the original heap satisfies another formula.

For $i \geq 1$, 1SLi denotes the fragment of 1SL restricted to i variables and 1SLi($-\!\!*$) to denote its restriction when separating conjunction is disallowed. Let $\mathfrak{L}$ be a logic among 1SL, 1SLi, 1SLi($-\!\!*$). The *satisfiability problem* for $\mathfrak{L}$ takes as input a sentence ϕ from $\mathfrak{L}$ and asks whether there is $\mathfrak{h}$ such that $\mathfrak{h} \models \phi$ (regardless of valuation, as ϕ has no free variables).

Theorem 1. [7, 12] The satisfiability problem for 1SL is undecidable, even if restricted to 1SL2.

2.2 Weak second-order logic (WSOL)

We consider a family $\mathtt{SVAR} = (\mathtt{SVAR}_i)_{i \geq 1}$ of second-order variables, denoted by $\mathtt{P}, \mathtt{Q}, \mathtt{R}, \ldots$ and interpreted as finite relations over $\mathbb{N}$. A second-order valuation $\mathfrak{f}$ is an interpretation of the second-order variables such that for every $\mathtt{P} \in \mathtt{SVAR}_i$, $\mathfrak{f}(\mathtt{P})$ is a finite subset of $\mathbb{N}^i$. Formulae ϕ of WSOL are defined by $\phi ::= \mathtt{u}_i = \mathtt{u}_j \mid \mathtt{u}_i \hookrightarrow \mathtt{u}_j \mid \phi \wedge \phi \mid \neg\phi \mid \exists\, \mathtt{u}_i\, \phi \mid \exists\, \mathtt{P}\, \phi \mid \mathtt{P}(\mathtt{u}_1, \ldots, \mathtt{u}_n)$, where the $\mathtt{P}$ are second-order variables with $\mathtt{P} \in \mathtt{SVAR}_n$ for some $n \geq 1$. We write DSOL (dyadic second-order logic) to denote the restriction of WSOL to second-order variables in $\mathtt{SVAR}_2$. Like 1SL, models for WSOL are finite heaps and quantifications are done over all the locations. Satisfaction relation $\models$ is defined as follows ($\mathfrak{f}$ is a hybrid valuation providing interpretation for any variable):

- $\mathfrak{h} \models_{\mathfrak{f}} \exists\, \mathtt{P}\, \phi$ iff there is a *finite* relation $R \subseteq \mathbb{N}^n$ such that $\mathfrak{h} \models_{\mathfrak{f}[\mathtt{P} \mapsto R]} \phi$ where $\mathtt{P} \in \mathtt{SVAR}_n$.
- $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{P}(\mathtt{u}_1, \ldots, \mathtt{u}_n)$ iff $(\mathfrak{f}(\mathtt{u}_1), \ldots, \mathfrak{f}(\mathtt{u}_n)) \in \mathfrak{f}(\mathtt{P})$.

The satisfiability problem for WSOL takes as input a sentence ϕ in WSOL and asks whether there is $\mathfrak{h}$ such that $\mathfrak{h} \models \phi$. By Trakhtenbrot's Theorem (see e.g. [3, 29]), the satisfiability problem for DSOL (and therefore for WSOL) is undecidable since finite satisfiability for first-order logic with a unique binary relation symbol is undecidable. Note that a monadic second-order variable can be simulated by a binary second-order variable from $\mathtt{SVAR}_2$, and this can be used to relativize a formula from DSOL in order to check finite satisfiability.

Theorem 2. [7] 1SL, WSOL and DSOL have the same expressive power.

Consequently, in order to show that 1SL2($-\!\!*$) is as expressive as WSOL (our main result), it is sufficient to prove that every sentence from DSOL has an equivalent sentence in 1SL2($-\!\!*$). It is worth noting that Theorem 2 extends to $k > 1$ record fields [7], which actually requires a simpler proof. A similar adaptation is possible from our main result (see Section 5.5).

3. Expressing Properties in 1SL2($-\!\!*$)

In the following, let $\mathtt{u}$ and $\overline{\mathtt{u}}$ be the variables $\mathtt{u}_1$ and $\mathtt{u}_2$, in either order. Throughout this paper, we build formulae with the quantified variables $\mathtt{u}$ and $\overline{\mathtt{u}}$. Note that any formula $\phi(\mathtt{u})$ with free variable $\mathtt{u}$ can be turned into an equivalent formula with free variable $\overline{\mathtt{u}}$ by permuting the two variables.

In 1SL2($-\!\!*$) it is not difficult to state that the domain of the heap is empty or that it is a singleton—for example, we can write $\exists\ \mathtt{u}\ ((\exists\ \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}}) \wedge \forall\ \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \neq \mathtt{u} \Rightarrow \neg\,(\exists\ \mathtt{u}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u})))$. However, to express that the domain contains exactly two locations, we run out of variables if we attempt to use a similar technique. Below, we propose a new and natural method to compare a number of predecessors against a constant or to express a reachability property. First, though, we define simple, standard formulae.

- $\mathtt{u}$ has a successor: $\mathtt{alloc}(\mathtt{u}) \stackrel{\text{def}}{=} \exists\ \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$.
- $\mathtt{u}$ is an *isolated location* (it is not in $\mathrm{dom}(\mathfrak{h}) \cup \mathrm{ran}(\mathfrak{h})$): $\mathtt{isoloc}(\mathtt{u}) \stackrel{\text{def}}{=} \neg\mathtt{alloc}(\mathtt{u}) \wedge \neg\exists\ \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u}$.
- $\mathrm{dom}(\mathfrak{h})$ has exactly one location: $(\mathtt{size} = 1) \stackrel{\text{def}}{=} \exists\ \mathtt{u}\ \big(\mathtt{alloc}(\mathtt{u}) \wedge \forall\ \overline{\mathtt{u}}\ (\mathtt{u} \neq \overline{\mathtt{u}} \Rightarrow \neg\mathtt{alloc}(\overline{\mathtt{u}}))\big)$.
- $\mathtt{u}$ has at least one predecessor: $\sharp\mathtt{u} > 0 \stackrel{\text{def}}{=} \exists\ \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u}$. Naturally, we can also write: $\sharp\mathtt{u} = 0 \stackrel{\text{def}}{=} \neg(\sharp\mathtt{u} > 0)$.

Let us now proceed with more complicated constructions.

3.1 Counting z-predecessors

In this paper, we will make heavy use of counting predecessors of locations. Without an adequate number of variables to refer to locations of interest, we instead "remember" locations by installing a large (and unique) number of predecessors to a location in the heap. At another point in the formula, we can identify this location and operate on it, having not used any of the logic's limited syntactic resources.

As one may imagine, counting and comparing numbers of predecessors is difficult to do in this logic. We first build up a method of counting a certain type of predecessor, then use it to formulate more involved constructions, and finally introduce a way to count the full number of predecessors of a location—that is, to compare its number of predecessors with some $k \geq 0$. After this is achieved, nearly the entirety of Section 4 extends this to comparing numbers of predecessors between two locations (rather than just a comparison to some k).

We first introduce the notion of *z-predecessors*, which are predecessors of a location that themselves have no predecessors ('z' is for 'zero'). We can then write:

- $\mathtt{u}$'s predecessors are all z-predecessors: $\mathtt{allzpred}(\mathtt{u}) \stackrel{\text{def}}{=} \forall\ \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u} \Rightarrow \sharp\overline{\mathtt{u}} = 0$.
- $\mathtt{u}$ has zero z-predecessors: $\sharp_z\mathtt{u} = 0 \stackrel{\text{def}}{=} \forall\ \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u} \Rightarrow \sharp\overline{\mathtt{u}} > 0$.
- $\mathtt{u}$ has at most $k > 0$ z-predecessors: $\sharp_z\mathtt{u} \leq k \stackrel{\text{def}}{=} (\mathtt{size} = 1) \mathbin{\overset{\neg}{*}} \sharp_z\mathtt{u} \leq k-1$ with $\sharp_z\mathtt{u} \leq 0$ defined as $\sharp_z\mathtt{u} = 0$.

Lemma 3. Let $\mathfrak{h}$ be a heap, $\mathfrak{f}$ be a valuation, and $k \in \mathbb{N}$. $\mathfrak{h} \models_{\mathfrak{f}} \sharp_z\mathtt{u} \leq k$ iff $\mathrm{card}(\{\mathfrak{l} \in \mathbb{N} : \widetilde{\sharp\mathfrak{l}} = 0,\ \mathfrak{h}(\mathfrak{l}) = \mathfrak{f}(\mathtt{u})\}) \leq k$.

Variants $\sharp_z\mathtt{u} \bowtie k$ with $\bowtie \in \{=, <, >, \geq\}$ are easily defined.

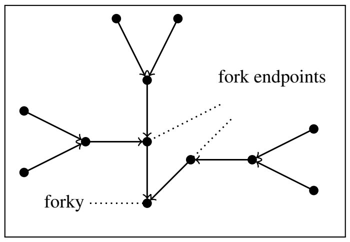
3.2 A matter of forks and knives

To address the problem of referring to memory locations despite not having the variables to do so, we introduce the notion of *forks*. Forks are simple, recognizable shapes that we add to the heap with the magic wand. Forks are also a critical building block for comparing predecessors in Section 4. A *fork* in $\mathfrak{h}$ is a sequence of distinct locations $\mathfrak{l}, \mathfrak{l}_0, \mathfrak{l}_1, \mathfrak{l}_2$ such that $\mathfrak{h}(\mathfrak{l}_0) = \mathfrak{l}$, $\widetilde{\sharp\mathfrak{l}_0} = 2$, $\mathfrak{h}(\mathfrak{l}_1) = \mathfrak{h}(\mathfrak{l}_2) = \mathfrak{l}_0$ and $\widetilde{\sharp\mathfrak{l}_1} = \widetilde{\sharp\mathfrak{l}_2} = 0$. The *endpoint* of the fork is $\mathfrak{l}$. The fork is *isolated* iff $\widetilde{\sharp\mathfrak{l}} = 1$ and $\mathfrak{l} \notin \mathrm{dom}(\mathfrak{h})$. Similarly, a *knife* in $\mathfrak{h}$ is a sequence of distinct locations $\mathfrak{l}, \mathfrak{l}_0, \mathfrak{l}_1$ such that $\mathfrak{h}(\mathfrak{l}_0) = \mathfrak{l}$, $\widetilde{\sharp\mathfrak{l}_0} = 1$, $\mathfrak{h}(\mathfrak{l}_1) = \mathfrak{l}_0$ and $\widetilde{\sharp\mathfrak{l}_1} = 0$. The *endpoint* of the knife is $\mathfrak{l}$. The knife is *isolated* iff $\widetilde{\sharp\mathfrak{l}} = 1$ and $\mathfrak{l} \notin \mathrm{dom}(\mathfrak{h})$.

To identify fork and knife endpoints in a heap, we define:

$$\mathtt{forkendpt}(\mathtt{u}) \stackrel{\text{def}}{=} \exists\ \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \sharp_z\overline{\mathtt{u}} = 2 \wedge \mathtt{allzpred}(\overline{\mathtt{u}}))$$
$$\mathtt{knifeendpt}(\mathtt{u}) \stackrel{\text{def}}{=} \exists\ \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \sharp_z\overline{\mathtt{u}} = 1 \wedge \mathtt{allzpred}(\overline{\mathtt{u}}))$$

Now, let $\mathtt{forky}(\mathtt{u})$ be a formula stating that *all* predecessors of $\mathfrak{f}(\mathtt{u})$, possibly except $\mathfrak{f}(\mathtt{u})$, are endpoints of forks: $\mathtt{forky}(\mathtt{u}) \stackrel{\text{def}}{=} \forall\ \overline{\mathtt{u}}\ ((\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \overline{\mathtt{u}} \neq \mathtt{u}) \Rightarrow \mathtt{forkendpt}(\overline{\mathtt{u}}))$. Three forks, two endpoints, and a forky location are depicted at left. Next, let $\mathtt{antiforky}(\mathtt{u})$ be a formula stating that *no* predecessor of $\mathfrak{f}(\mathtt{u})$ is the endpoint of a fork, and let $\mathtt{antiknifty}(\mathtt{u})$ be a formula stating that *no* predecessor of $\mathfrak{f}(\mathtt{u})$ is the endpoint of a knife. We define these as



$\mathtt{antiforky}(\mathtt{u}) \stackrel{\text{def}}{=} \forall\ \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \Rightarrow \neg\mathtt{forkendpt}(\overline{\mathtt{u}}))$ and $\mathtt{antiknifty}(\mathtt{u}) \stackrel{\text{def}}{=} \forall\ \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \Rightarrow \neg\mathtt{knifeendpt}(\overline{\mathtt{u}}))$. Note the asymmetry between $\mathtt{forky}(\mathtt{u})$ and $\mathtt{antiforky}(\mathtt{u})$ since $\mathfrak{f}(\mathtt{u})$ does not have to be the endpoint of a fork in $\mathtt{forky}(\mathtt{u})$ (which would be then impossible to realize for all the predecessors of $\mathfrak{f}(\mathtt{u})$ if $\mathfrak{f}(\mathtt{u})$ were a self-loop). It is also easy to enforce that the heap is made of a single fork, which will be useful afterwards.

Lemma 4. There is a formula $\mathtt{1fork}$ in 1SL2($-\!\!*$) such that for all $\mathfrak{h}$, we have $\mathfrak{h} \models \mathtt{1fork}$ iff $\mathfrak{h}$ is only made of a single, isolated fork.

Now let us build formulae that constrain the number of predecessors (not just z-predecessors): with $k \in \mathbb{Z}$, we define $\sharp\mathtt{u} \leq k \stackrel{\text{def}}{=} \perp$ if $k < 0$, $\sharp\mathtt{u} \leq 0 \stackrel{\text{def}}{=} \neg\exists\ \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u}$, and for $k > 0$,

$$\sharp\mathtt{u} \leq k \stackrel{\text{def}}{=}$$
$$(\mathtt{u} \hookrightarrow \mathtt{u} \wedge \overset{\star}{\sharp}\mathtt{u} \leq k-1) \vee (\neg(\mathtt{u} \hookrightarrow \mathtt{u}) \wedge \overset{\star}{\sharp}\mathtt{u} \leq k)$$

where $\overset{\star}{\sharp}\mathtt{u} \leq 0 \stackrel{\text{def}}{=} \neg\exists\ \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \overline{\mathtt{u}} \neq \mathtt{u})$ and $\overset{\star}{\sharp}\mathtt{u} \leq k \stackrel{\text{def}}{=} (\sharp\mathtt{u} = 0) \mathbin{\overset{\neg}{*}} (\mathtt{antiforky}(\mathtt{u}) \wedge (\underbrace{\mathtt{1fork} \mathbin{\overset{\neg}{*}} \cdots \mathbin{\overset{\neg}{*}} \mathtt{1fork}}_{k \text{ times}} \mathbin{\overset{\neg}{*}} \mathtt{forky}(\mathtt{u}))$ for $k > 0$. In a nutshell, $\mathfrak{f}(\mathtt{u})$ has at most $k > 0$ predecessors if one can make $\mathfrak{f}(\mathtt{u})$ antiforky without changing its predecessor count (always possible) and then adding k forks to make $\mathfrak{f}(\mathtt{u})$ forky (we distinguish the case when $\mathfrak{f}(\mathtt{u})$ is a self-loop).

Lemma 5. Let $k \in \mathbb{N}$, $\mathfrak{h}$ be a heap and $\mathfrak{f}$ be a valuation. (I) $\mathfrak{h} \models_{\mathfrak{f}} \overset{\star}{\sharp}\mathtt{u} \leq k$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} \leq k$; (II) $\mathfrak{h} \models_{\mathfrak{f}} \sharp\mathtt{u} \leq k$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} \leq k$.

Thus we can express in 1SL2($-\!\!*$) the following properties:

- $\mathfrak{f}(\mathtt{u})$ has at least k predecessors: $\sharp\mathtt{u} \geq k \overset{\text{def}}{=} \neg(\sharp\mathtt{u} \leq k-1)$.
- $\mathfrak{f}(\mathtt{u})$ has exactly k predecessors: $\sharp\mathtt{u} = k \overset{\text{def}}{=} \sharp\mathtt{u} \leq k \wedge \sharp\mathtt{u} \geq k$.

Lonely memory cells Now, we can express a useful property on the heap itself. The formula below states that $\mathfrak{f}(\mathtt{u})$ is an isolated cell: $\mathtt{isocell}(\mathtt{u}) \overset{\text{def}}{=} \mathtt{alloc}(\mathtt{u}) \wedge \sharp\mathtt{u} = 0 \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \sharp\overline{\mathtt{u}} = 1 \wedge \neg\mathtt{alloc}(\overline{\mathtt{u}}))$. A heap $\mathfrak{h}$ is *segmented* whenever $\mathrm{dom}(\mathfrak{h}) \cap \mathrm{ran}(\mathfrak{h}) = \emptyset$ and no location has strictly more than one predecessor. Otherwise said, all the memory cells are isolated. Being segmented can be naturally expressed in 1SL2($-\!\!*$) too:

$$\mathtt{seg} \overset{\text{def}}{=} \forall\, \mathtt{u}\, \forall\, \overline{\mathtt{u}}\ \big(\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \Rightarrow \big(\sharp\overline{\mathtt{u}} = 1 \wedge \sharp\mathtt{u} = 0\big)\big)$$

3.3 Expressing reachability

In 1SL, reachability can be expressed, and [12] gives a technique for doing so with the two-variable restriction, itself a variant of material from [7, 11]. Without the separating conjunction, we can still specify reachability from $\mathfrak{f}(\mathtt{u})$ to $\mathfrak{f}(\overline{\mathtt{u}})$, but we need a new technique: we put a fork on $\mathfrak{f}(\mathtt{u})$, propagate this fork forward in the heap, and finally check whether $\mathfrak{f}(\overline{\mathtt{u}})$ is the endpoint of some fork. This is analogous to the way reachability is handled with a monadic second-order predicate, but here, a finite set of locations is identified by propagation of forks. We define $\mathtt{propagate}(\mathtt{u})$ as the formula characterizing the property that $\mathfrak{f}(\mathtt{u})$ is the endpoint of some fork, and that the property of being the endpoint of some fork is propagated along memory cells. The reachability predicate is written $\mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}})$:

$$\begin{aligned}&\mathtt{propagate}(\mathtt{u}) \overset{\text{def}}{=} \mathtt{forkendpt}(\mathtt{u}) \wedge \\ &\quad \forall\, \mathtt{u}\, \forall\, \overline{\mathtt{u}}\ \big((\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \mathtt{forkendpt}(\mathtt{u})) \Rightarrow \mathtt{forkendpt}(\overline{\mathtt{u}})\big) \\ &\mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}}) \overset{\text{def}}{=} \top -\!\!*\, (\mathtt{propagate}(\mathtt{u}) \Rightarrow \mathtt{forkendpt}(\overline{\mathtt{u}}))\end{aligned}$$

The purpose of "$\top -\!\!*$" is to destroy forks in the original heap (if any) and to augment it with new forks such that $\mathtt{propagate}(\mathtt{u})$ is satisfied. Of course, a disjoint subheap could change the very reachability property we are testing. But some such subheaps do not, and the universal quantification of the magic wand ensures that we are on the safe side: if for *all* combined heaps satisfying $\mathtt{propagate}(\mathtt{u})$, $\mathfrak{f}(\overline{\mathtt{u}})$ is the endpoint of a fork, then we can conclude that $\mathfrak{f}(\mathtt{u})$ reaches $\mathfrak{f}(\overline{\mathtt{u}})$ in the original heap.

Lemma 6. Given a heap $\mathfrak{h}$ and valuation $\mathfrak{f}$, we have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}})$ iff $\mathfrak{h}^k(\mathfrak{f}(\mathtt{u})) = \mathfrak{f}(\overline{\mathtt{u}})$ for some $k \geq 0$.

It is notable that 1SL2($-\!\!*$) has no need for built-in reachability predicates, in contrast to formalisms from e.g. [21]. In the rest of the paper, we generalize, in a sense, what was done here in an *ad hoc* manner for reachability, so that any second-order property can be represented in 1SL2($-\!\!*$).

4. Comparing Numbers of Predecessors

The main goal of this section is to define in 1SL2($-\!\!*$) a formula expressing that $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})} + k'$, where $k, k' \in \mathbb{N}$, for any heap $\mathfrak{h}$ and valuation $\mathfrak{f}$. Without the restriction on the number of variables, we know that such properties can be expressed in 1SL($-\!\!*$) [7]. Note that arithmetical constraints on list lengths can be found in [4] but this is primitive in the logical formalism. By contrast, we show that $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})} + k'$ can be expressed in 1SL2($-\!\!*$) itself. Since a property $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})} + k'$ requires a formula with $\mathcal{O}(k + k')$ variables in 1SL($-\!\!*$) according to [7], herein we need to circumvent this issue by proposing an alternative way to express the key properties that are helpful to state that $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})} + k'$. Below, we still use first principles from [7] to construct a formula in 1SL2($-\!\!*$)—mainly, how to build a fork from a knife and an isolated memory cell—but we will need to bypass the serious problem of having only two variables at our disposal, without permitting ourselves any use of the separating conjunction.

4.1 Principles and difficulties with 1SL2($-\!\!*$)

Let $\mathfrak{h}$ be a heap and $\mathfrak{f}$ be a valuation for which we wish to check if $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} + k'$ holds (it is easy to conclude if $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})} + k'$ holds, see the proof of Theorem 10). Below, we explain which extensions of $\mathfrak{h}$ must be performed to achieve this. We mainly describe first principles from [7], but the reader should be warned that in several places, we propose a simplified alternative, apart from the fact that all the formulae need to be part of the restricted fragment 1SL2($-\!\!*$).

4.1.1 Preparing the heap

The first step consists in preparing the heap by destroying any forks and knives at $\mathfrak{f}(\mathtt{u})$ and $\mathfrak{f}(\overline{\mathtt{u}})$, and ensuring there are no isolated memory cells—these properties will be necessary in later steps—while maintaining the number of predecessors at $\mathfrak{f}(\mathtt{u})$ and $\mathfrak{f}(\overline{\mathtt{u}})$. To do this, we augment $\mathfrak{h}$ with a heap $\mathfrak{h}_p$ so that (a) $\mathfrak{h}_p$ is disjoint from $\mathfrak{h}$; (b) $\mathfrak{f}(\mathtt{u})$ has the same number of predecessors in $\mathfrak{h}$ and in $\mathfrak{h} \uplus \mathfrak{h}_p$, say $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} = m \geq 0$; (c) $\mathfrak{f}(\overline{\mathtt{u}})$ has the same number of predecessors in $\mathfrak{h}$ and in $\mathfrak{h} \uplus \mathfrak{h}_p$, say $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} = \overline{m} \geq 0$; (d) In $\mathfrak{h} \uplus \mathfrak{h}_p$, $\mathfrak{f}(\mathtt{u})$ has no predecessor that is an endpoint of a fork or knife; (e) In $\mathfrak{h} \uplus \mathfrak{h}_p$, $\mathfrak{f}(\overline{\mathtt{u}})$ has no predecessor that is an endpoint of a fork or knife; (f) $\mathfrak{h} \uplus \mathfrak{h}_p$ has no isolated memory cell. This step is always possible, since to destroy the structure of an isolated memory cell, a fork or a knife, it is sufficient to add a memory cell at the adequate position. We take advantage of the formulae $\mathtt{antiforky}(\cdot)$ and $\mathtt{antiknify}(\cdot)$ to establish properties (d) and (e); the others are straightforward.

4.1.2 Addition of a segmented heap

This step consists in checking whether, for each segmented heap $\mathfrak{h}_s$ satisfying certain properties, the condition ($\mathcal{P}$) defined below holds true. The heap $\mathfrak{h}_s$ must be segmented and also must satisfy the following: (a) $\mathfrak{h}_s$ is disjoint from $\mathfrak{h} \uplus \mathfrak{h}_p$; (b) $\mathfrak{f}(\mathtt{u})$ and $\mathfrak{f}(\overline{\mathtt{u}})$ have no predecessors in $\mathfrak{h}_s$; (c) In $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s$, neither $\mathfrak{f}(\mathtt{u})$ nor $\mathfrak{f}(\overline{\mathtt{u}})$ has a predecessor that is an endpoint of a fork or knife (they are *antiforky* and *antiknify*). Let n be the number of isolated memory cells in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s$. Note that for each $q \geq 0$, it is possible to build $\mathfrak{h}_s^q$ so that $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s^q$ has exactly q isolated memory cells and $\mathfrak{h}_s^q$ satisfies the above conditions.

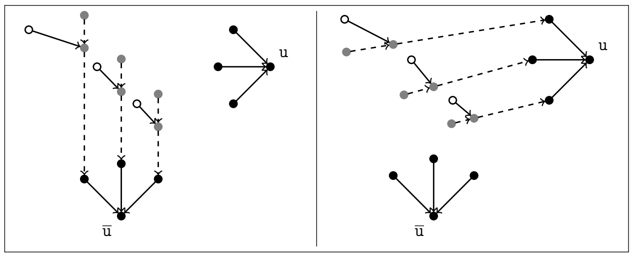


Figure 1: The construction of forks used for comparison.

In order to construct forks in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s$ whose endpoints are predecessors of $\mathfrak{f}(\mathtt{u})$ or $\mathfrak{f}(\overline{\mathtt{u}})$, either we can augment the heap with a fork, or we can augment the heap with a knife so that its combination with an isolated memory cell in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s$ leads to a fork whose endpoint is a predecessor of $\mathfrak{f}(\mathtt{u})$ or $\mathfrak{f}(\overline{\mathtt{u}})$. See Figure 1, which depicts two locations, each with three predecessors. Imagine we are performing the comparison $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star}$ (presently equivalent to $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}$). First, a segmented heap is added (allocated locations are marked with a white circle in the figure). The left half of the figure shows the addition of a collection of knives through these segments to make $\mathfrak{f}(\overline{\mathtt{u}})$ forky; with the same segments, it must then be possible to add a collection of knives to make $\mathfrak{f}(\mathtt{u})$ forky. Indeed it is; if this is true for *all* such segmented heaps, then it must be that $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star}$. The segments from the segmented heap should really be seen as "potential forks." When k or k' is nonzero, we consider additional forks on one or both sides to compensate for the offset.

This illustrates the principle behind the definition of the following property ($\mathcal{P}$): If there is a heap $\mathfrak{h}_{[k]}$ disjoint from $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s$ made of isolated knives and k isolated forks so that $\mathfrak{f}(\overline{\mathtt{u}})$ is forky in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s \uplus \mathfrak{h}_{[k]}$, then there is a heap $\mathfrak{h}_{[k']}$ disjoint from $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s$ made of isolated knives and k' isolated forks so that $\mathfrak{f}(\mathtt{u})$ is forky in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s \uplus \mathfrak{h}_{[k']}$. Note that by the conditions satisfied by $\mathfrak{h}_{[k]}$ or by $\mathfrak{h}_{[k']}$, the number of predecessors of $\mathfrak{f}(\overline{\mathtt{u}})$ [resp. $\mathfrak{f}(\mathtt{u})$] in $\mathfrak{h}_{[k]}$ [resp. $\mathfrak{h}_{[k']}$] is necessarily zero (there is no need to specify explicitly that we do not add predecessors). The number of forks in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s \uplus \mathfrak{h}_{[k]}$ whose endpoints are predecessors of $\mathfrak{f}(\overline{\mathtt{u}})$ is bounded by $n + k$ and similarly, the number of forks in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s \uplus \mathfrak{h}_{[k']}$ whose endpoints are predecessors of $\mathfrak{f}(\mathtt{u})$ is bounded by $n + k'$. Note also that the number of predecessors of $\mathfrak{f}(\mathtt{u})$ is the same in $\mathfrak{h}$ and in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s \uplus \mathfrak{h}_{[k']}$ and the number of predecessors of $\mathfrak{f}(\overline{\mathtt{u}})$ is the same in $\mathfrak{h}$ and in $\mathfrak{h} \uplus \mathfrak{h}_p \uplus \mathfrak{h}_s \uplus \mathfrak{h}_{[k]}$. So, $n + k \geq \overline{m}$ implies $n + k' \geq m$, i.e. $n \geq \overline{m} - k$ implies $n \geq m - k'$.

4.1.3 Checking the resulting heap

By checking step 2 for all $n \geq 0$, we get that for all $n \geq 0$, we have $n \geq \overline{m} - k$ implies $n \geq m - k'$, which entails that $\overline{m} - k \geq m - k'$, i.e. $m + k \leq \overline{m} + k'$, whenever $m - k' \geq 0$ and $\overline{m} - k \geq 0$. Universal quantification over n is simulated in a formula by using separating implication. When $m < k'$ or $\overline{m} < k$, we make a dedicated case analysis (see the proof of Theorem 10). Below, we present the technical developments.

4.2 Building blocks

Apart from forks, we introduce the notions of *collections* and *large forks* (instrumental in the proof of Lemma 7 below). A *large fork* is a sequence of distinct locations $\mathfrak{l}_1, \cdots, \mathfrak{l}_5$ such that $\mathfrak{l}_1$, $\mathfrak{l}_2$, and $\mathfrak{l}_3$ have no predecessors, $\mathfrak{h}(\mathfrak{l}_1) = \mathfrak{h}(\mathfrak{l}_2) = \mathfrak{h}(\mathfrak{l}_3) = \mathfrak{l}_4$, $\widetilde{\sharp\mathfrak{l}_4} = 3$ and $\mathfrak{h}(\mathfrak{l}_4) = \mathfrak{l}_5$. Location $\mathfrak{l}_5$ is called the *endpoint* of the large fork, and as with forks and knives, the large fork is called *isolated* iff $\widetilde{\sharp\mathfrak{l}_5} = 1$ and $\mathfrak{l}_5 \notin \mathrm{dom}(\mathfrak{h})$. A heap $\mathfrak{h}$ is a *collection of knives and forks* $\overset{\text{def}}{\Leftrightarrow}$ there is no location in $\mathrm{dom}(\mathfrak{h})$ that does not belong to an isolated knife or to an isolated fork. Similarly, a heap $\mathfrak{h}$ is a *collection of knives and large forks* $\overset{\text{def}}{\Leftrightarrow}$ there is no location in $\mathrm{dom}(\mathfrak{h})$ that does not belong to an isolated knife or isolated large fork.

Lemma 7. There are formulae $\mathtt{ksfs}$, $\mathtt{kslfs}$ and $\mathtt{ksfs}_k$ ($k \geq 0$) in 1SL2($-\!\!*$) such that for every heap $\mathfrak{h}$, (1) $\mathfrak{h} \models \mathtt{ksfs}$ iff $\mathfrak{h}$ is a collection of knives and forks, (2) $\mathfrak{h} \models \mathtt{kslfs}$ iff $\mathfrak{h}$ is a collection of knives and large forks, (3) $\mathfrak{h} \models \mathtt{ksfs}_k$ iff $\mathfrak{h}$ is a collection of knives and forks with exactly k forks.

Here is the way we can use the formulae from Lemma 7.

Lemma 8. Let $k \geq 0$, $\mathfrak{h}$ be a heap and $\mathfrak{f}$ be a valuation such that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{antiforky}(\mathtt{u}) \wedge \mathtt{antiknify}(\mathtt{u})$, $\mathfrak{h}$ has n isolated memory cells and $m = \widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star}$. We have $\mathfrak{h} \models_{\mathfrak{f}} (\mathtt{ksfs}_k \overset{\neg}{-\!\!*} \mathtt{forky}(\mathtt{u}))$ iff $n \geq m - k$.

The proof of Lemma 8 uses principles similar to what has been done in [7, 12] except that all formulae belong to 1SL2($-\!\!*$) and we propose a simplified version of the lemma and proof (note also our use of $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star}$). Let $\mathtt{anti}$ be $\mathtt{antiforky}(\mathtt{u}) \wedge \mathtt{antiknify}(\mathtt{u}) \wedge \mathtt{antiforky}(\overline{\mathtt{u}}) \wedge \mathtt{antiknify}(\overline{\mathtt{u}})$ and let $\mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, k, k')$ be

$$\big[(\mathtt{seg} \wedge \sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0) -\!\!*$$

$$\big(\mathtt{anti} \Rightarrow \big(\big[\mathtt{ksfs}_k \overset{\neg}{-\!\!*} \mathtt{forky}(\overline{\mathtt{u}})\big] \Rightarrow \big[\mathtt{ksfs}_{k'} \overset{\neg}{-\!\!*} \mathtt{forky}(\mathtt{u})\big]\big)\big)\big]$$

Proposition 9. Let $k, k' \geq 0$, $\mathfrak{f}$ and $\mathfrak{h}$ such that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{anti} \wedge \neg\exists\ \mathtt{u}\ \mathtt{isocell}(\mathtt{u})$, $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} - k' \geq 0$ and $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} - k \geq 0$. We have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, k, k')$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} + k'$.

Here is the main result in this section using $\mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, k, k')$.

Theorem 10. For $k, k' \geq 0$, there is ϕ in 1SL2($-\!\!*$) (of linear size in $k + k'$) such that for all $\mathfrak{h}$, $\mathfrak{f}$, ($\mathfrak{h} \models_{\mathfrak{f}} \phi$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})} + k'$).

We denote such a formula ϕ as $\sharp\mathtt{u}{+}k \leq \sharp\overline{\mathtt{u}}{+}k'$ and, as usual, easily extend to $<, \geq, >, =$. The structure of the proof of Theorem 10 is similar to the structure of the proof of [7, Theorem 5.5], except that now all formulae are in 1SL2($-\!\!*$) instead of being defined in the less-constrained 1SL($-\!\!*$). Moreover, our case analysis is quite different and we also provide several simplifications, making our new proof even more valuable.

Proof. Below, we show that for $k, k' \geq 0$, there is a formula $\phi_{k,k'}$ in 1SL2($-\!\!*$) (of linear size in $k + k'$) such that for every heap $\mathfrak{h}$ and valuation $\mathfrak{f}$, we have $\mathfrak{h} \models_{\mathfrak{f}} \phi_{k,k'}$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} + k'$. Once we have such a $\phi_{k,k'}$, we easily define ϕ as:

$$(\mathtt{u} \hookrightarrow \mathtt{u} \wedge \overline{\mathtt{u}} \hookrightarrow \overline{\mathtt{u}} \wedge \phi_{k,k'}) \vee (\neg(\mathtt{u} \hookrightarrow \mathtt{u}) \wedge \overline{\mathtt{u}} \hookrightarrow \overline{\mathtt{u}} \wedge \phi_{k,k'+1}) \vee$$
$$(\mathtt{u} \hookrightarrow \mathtt{u} \wedge \neg(\overline{\mathtt{u}} \hookrightarrow \overline{\mathtt{u}}) \wedge \phi_{k+1,k'}) \vee (\neg(\mathtt{u} \hookrightarrow \mathtt{u}) \wedge \neg(\overline{\mathtt{u}} \hookrightarrow \overline{\mathtt{u}}) \wedge \phi_{k,k'})$$

So it only remains to define the formulae $\phi_{k,k'}$ with $k, k' \geq 0$. By Proposition 9, we have the following property for any $\mathfrak{h}, \mathfrak{f}$:

($\star$) When $\mathfrak{h}$ satisfies $\mathtt{anti} \wedge \neg\exists\ \mathtt{u}\ \mathtt{isocell}(\mathtt{u})$, $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} - k' \geq 0$ and $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} - k \geq 0$, we have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, k, k')$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} + k'$.

Even though the original heap $\mathfrak{h}$ does not satisfy the formula $\mathtt{anti} \wedge \neg\exists\ \mathtt{u}\ \mathtt{isocell}(\mathtt{u})$, it can be safely extended to satisfy this property without modifying the number of predecessors of $\mathfrak{f}(\mathtt{u})$ and $\mathfrak{f}(\overline{\mathtt{u}})$.

Whenever $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} - k' \geq 0$ and $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} - k \geq 0$, we have the following equivalences:

(1) $\mathfrak{h} \models_{\mathfrak{f}} (\sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0) \overset{\neg}{-\!\!*} (\mathtt{anti} \wedge (\neg\exists\ \mathtt{u}\ \mathtt{isocell}(\mathtt{u})) \wedge \mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, k, k'))$.
(2) There is $\mathfrak{h}'$ disjoint from $\mathfrak{h}$ such that $\mathfrak{h}' \models_{\mathfrak{f}} (\sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0)$, $\mathfrak{h} \uplus \mathfrak{h}' \models_{\mathfrak{f}} \mathtt{anti} \wedge \neg\exists\ \mathtt{u}\ \mathtt{isocell}(\mathtt{u})$ and $\mathfrak{h} \uplus \mathfrak{h}' \models_{\mathfrak{f}} \mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, k, k')$.
(3) There is $\mathfrak{h}'$ disjoint from $\mathfrak{h}$ such that $\mathfrak{h}' \models_{\mathfrak{f}} (\sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0)$ and $\mathfrak{h} \uplus \mathfrak{h}' \models_{\mathfrak{f}} \mathtt{anti} \wedge \neg\exists\ \mathtt{u}\ \mathtt{isocell}(\mathtt{u})$ and $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} + k'$ (in $\mathfrak{h} \uplus \mathfrak{h}'$) by ($\star$).
(4) $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} + k'$ in $\mathfrak{h}$.

Observe that $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star}$ and $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star}$ in $\mathfrak{h}$ are equal to their values in $\mathfrak{h} \uplus \mathfrak{h}'$ since $\mathfrak{h}' \models_{\mathfrak{f}} (\sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0)$. Moreover, (4) implies (3) since it is always possible to extend a model satisfying $\mathtt{anti} \wedge \neg\exists\ \mathtt{u}\ \mathtt{isocell}(\mathtt{u})$ while preserving $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star}$ and $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star}$.
W.l.o.g., we can assume that $k \times k' = 0$.
Case $k = 0$ *and* $k' \geq 0$. So $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} - k \geq 0$ and we need to make a case analysis depending on the satisfaction of $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} \geq k'$. Note

that if $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} < k'$, then obviously $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} + k'$. So, we write $\phi_{k,k'}$ to denote the formula below:

$$(\overset{\star}{\sharp}\mathtt{u} < k') \vee ((\overset{\star}{\sharp}\mathtt{u} \geq k') \wedge ((\sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0) \overset{\neg}{-\!\!\ast}$$

$$(\mathtt{anti} \wedge (\neg\exists\, \mathtt{u}\, \mathtt{isocell}(\mathtt{u})) \wedge \mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, 0, k'))))$$

Formulae of the form $\overset{\star}{\sharp}\mathtt{u} \geq k'$ (and variants) can be defined thanks to Lemma 5.
Case $k' = 0$ and $k \geq 0$. So $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} - k' \geq 0$ and we need to make a case analysis depending on the satisfaction of $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} \geq k$. Note that if $\widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star} < k$, then $\widetilde{\sharp\mathfrak{f}(\mathtt{u})}^{\star} + k \leq \widetilde{\sharp\mathfrak{f}(\overline{\mathtt{u}})}^{\star}$ cannot hold. So, we write $\phi_{k,k'}$ to denote the formula below:

$$(\overset{\star}{\sharp}\overline{\mathtt{u}} \geq k) \wedge ((\sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0) \overset{\neg}{-\!\!\ast}$$

$$(\mathtt{anti} \wedge (\neg\exists\, \mathtt{u}\, \mathtt{isocell}(\mathtt{u})) \wedge \mathtt{comp}(\mathtt{u}, \overline{\mathtt{u}}, k, 0))) \quad \square$$

We can now find locations in a heap with a maximal number of predecessors, and we conclude this section with a definition useful in later constructions. Let $\mathtt{maxdeg}(\mathtt{u}) \overset{\text{def}}{=} \neg\exists\, \overline{\mathtt{u}}\ \sharp\overline{\mathtt{u}} > \sharp\mathtt{u}$. For all $\mathfrak{h}$ and $\mathfrak{f}$, we have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{maxdeg}(\mathtt{u})$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} = max(\{\widetilde{\sharp\mathfrak{l}} : \mathfrak{l} \in \mathbb{N}\})$.

5. Expressive Completeness for 1SL2($-\!\!\ast$)

In order to express sentences in DSOL by sentences in 1SL2($-\!\!\ast$), a hybrid valuation is encoded in the heap by building a disjoint *valuation heap* that takes care of pairs of locations (for interpretation of second-order variables) and that takes care of locations (for interpretation of first-order variables). In principle, this makes sense since every heap has a finite domain and therefore there is always an infinite set of locations that is not in its domain. This leaves enough room to encode a finite amount of information such as the interpretation of second-order variables when they are interpreted by finite sets. We can easily add to the original heap with the magic wand; this permits us to create and update the valuation heap. However, we then must always be able to distinguish between the original heap and the valuation heap.

The main idea to build such a valuation heap rests on the fact that a pair of locations $(\mathfrak{l}, \mathfrak{l}')$ belongs to the interpretation of a second-order variable $\mathtt{P}_i$ whenever $\mathfrak{l}$ and $\mathfrak{l}'$ can be identified by special patterns involving $\mathfrak{l}$ and $\mathfrak{l}'$ that uniquely characterize the interpretation by $\mathtt{P}_i$. Similarly, a location $\mathfrak{l}$ is the interpretation of a first-order variable whenever $\mathfrak{l}$ can be identified in the valuation heap thanks to some dedicated pattern around $\mathfrak{l}$.

Before explaining further the general principles, let us first provide more information about the above-mentioned patterns. An *entry of degree $d \geq 2$* is a sequence of distinct locations $\mathfrak{l}_1, \ldots, \mathfrak{l}_d, \mathfrak{l}_{\mathfrak{ind}}, \mathfrak{l}$ such that $\mathfrak{h}(\mathfrak{l}_1) = \cdots = \mathfrak{h}(\mathfrak{l}_d) = \mathfrak{l}_{\mathfrak{ind}}$, $\widetilde{\sharp\mathfrak{l}_{\mathfrak{ind}}} = d$, $\widetilde{\sharp\mathfrak{l}_1} = \cdots = \widetilde{\sharp\mathfrak{l}_d} = 0$ and $\mathfrak{h}(\mathfrak{l}_{\mathfrak{ind}}) = \mathfrak{l}$. The location $\mathfrak{l}$ is called the *element*, $\mathfrak{l}_{\mathfrak{ind}}$ the *index* and the locations $\mathfrak{l}_1, \ldots, \mathfrak{l}_d$, the *pins*. Entries generalize the notions of forks and large forks from Section 3 and are called *markers* in [7]. See an entry of degree 4 in the middle of Figure 2. So, the pair of locations $(\mathfrak{l}, \mathfrak{l}')$ is identified as part of the interpretation of $\mathtt{P}_i$ when $\mathfrak{l}$ and $\mathfrak{l}'$ are elements of entries with very large degree. The above-mentioned special patterns are therefore entries, but we require that the degree of the respective entries for $\mathfrak{l}$ and $\mathfrak{l}'$ satisfy arithmetical constraints, which is possible thanks to Theorem 10, and which allows us to relate $\mathfrak{l}$ with $\mathfrak{l}'$.

Then, the principle of the translation consists in building the valuation heap on demand (typically when a quantification appears) and to find special patterns involving entries with large degree whenever an atomic formula needs to be evaluated.

Apart from our essential restriction to 1SL2($-\!\!\ast$) and therefore the need for encoding also first-order valuations, these principles have been introduced in [7] to translate DSOL formulae into 1SL($-\!\!\ast$) formulae. However, because we are restricted to two first-order variables and because we also require that the separating conjunction is banished, we present below a different way to apply these principles so that we can show that 1SL2($-\!\!\ast$) is expressively equivalent to DSOL.

This high-level description of the formula translation and of the encoding of some hybrid valuation in the heap hides many of the details, which can be found below. However, before explaining how we apply these principles within 1SL2($-\!\!\ast$), let us emphasize the most obvious and difficult problems to be solved: (I) we must be able to distinguish the pairs of locations from distinct second-order variables, (II) we also need to encode first-order valuations, and (III) the main problem is certainly to access the original heap properly without interference from the valuation heap.

5.1 Left and right parentheses

We introduce variants of entries that are used as delimiters. A *left j-parenthesis of degree $d \geq 3$ with $j \geq 0$* is a sequence of distinct locations $\mathfrak{l}'_{j+1}, \ldots, \mathfrak{l}'_1, \mathfrak{l}_1, \ldots, \mathfrak{l}_d, \mathfrak{l}_{\mathfrak{ind}}$ such that (u) $\mathfrak{h}(\mathfrak{l}_1) = \cdots = \mathfrak{h}(\mathfrak{l}_d) = \mathfrak{l}_{\mathfrak{ind}}$; $\widetilde{\sharp\mathfrak{l}_{\mathfrak{ind}}} = d$; $\widetilde{\sharp\mathfrak{l}'_{j+1}} = \widetilde{\sharp\mathfrak{l}_3} = \widetilde{\sharp\mathfrak{l}_4} = \cdots = \widetilde{\sharp\mathfrak{l}_d} = 0$, (v) $\mathfrak{l}_{\mathfrak{ind}} \notin \mathrm{dom}(\mathfrak{h})$; $\mathfrak{l}'_{j+1} \rightarrow \mathfrak{l}'_j \rightarrow \mathfrak{l}'_{j-1} \rightarrow \cdots \rightarrow \mathfrak{l}'_1 \rightarrow \mathfrak{l}_1$; $\widetilde{\sharp\mathfrak{l}'_j} = \widetilde{\sharp\mathfrak{l}'_{j-1}} = \cdots = \widetilde{\sharp\mathfrak{l}'_1} = \widetilde{\sharp\mathfrak{l}_1} = 1$, and (w) $\widetilde{\sharp\mathfrak{l}_2} = 0$. The location $\mathfrak{l}_{\mathfrak{ind}}$ is called the *index*. The heap at the left of Figure 2 presents a left j-parenthesis of degree 3. A *right j-parenthesis of degree $d \geq 3$ with $j \geq 0$* is a sequence of distinct locations $\mathfrak{l}'_{j+1}, \ldots, \mathfrak{l}'_1, \mathfrak{l}''_{j+1}, \ldots, \mathfrak{l}''_1, \mathfrak{l}_1, \ldots, \mathfrak{l}_d, \mathfrak{l}_{\mathfrak{ind}}$ such that (u), (v), $\widetilde{\sharp\mathfrak{l}''_{j+1}} = 0$, $\widetilde{\sharp\mathfrak{l}''_j} = \widetilde{\sharp\mathfrak{l}''_{j-1}} = \cdots = \widetilde{\sharp\mathfrak{l}''_1} = \widetilde{\sharp\mathfrak{l}_2} = 1$, and $\mathfrak{l}''_{j+1} \rightarrow \mathfrak{l}''_j \rightarrow \mathfrak{l}''_{j-1} \rightarrow \cdots \rightarrow \mathfrak{l}''_1 \rightarrow \mathfrak{l}_2$. The location $\mathfrak{l}_{\mathfrak{ind}}$ is also called the *index*. The heap at the right of Figure 2 presents a right j-parenthesis of degree 5. A j-parenthesis can be understood as an entry, except that the index location is not allocated, and containing one or two paths of length $j+1$, depending whether it is a left or a right parenthesis.

Lemma 11. For all $j \geq 0$, there is a formula $\mathtt{lp}_j(\mathtt{u})$ [resp. $\mathtt{rp}_j(\mathtt{u})$] in 1SL2($-\!\!\ast$) such that for all heaps $\mathfrak{h}$ and valuations $\mathfrak{f}$, we have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{lp}_j(\mathtt{u})$ [resp. $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{rp}_j(\mathtt{u})$] iff $\mathfrak{f}(\mathtt{u})$ is the index of some left [resp. right] j-parenthesis in $\mathfrak{h}$.

In several places, we need to identify the indices from entries as well as their pins. Let $\mathtt{eindex}(\mathtt{u})$ be $(\sharp_z\mathtt{u} \geq 2) \wedge \mathtt{allzpred}(\mathtt{u}) \wedge \exists\, \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$ that characterizes indices from entries. Let $\mathtt{epin}(\mathtt{u})$ characterize pins from entries: $\mathtt{epin}(\mathtt{u}) \overset{\text{def}}{=} \exists\, \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \mathtt{eindex}(\overline{\mathtt{u}})$. Similarly, we need to characterize the locations from parentheses. We already know how to identify their indices (Lemma 11). It remains to identify the other locations via the formula $\mathtt{onp}_i(\mathtt{u})$ to characterize the locations on some i-parenthesis: roughly speaking, such locations are exactly those that can reach the index of some i-parenthesis in less than $i+2$ steps. Let $\mathtt{onp}_i(\mathtt{u}) \overset{\text{def}}{=} \bigvee_{j=0}^{i+2} \mathtt{dist}_i(j, \mathtt{u})$ with $\mathtt{dist}_i(0, \mathtt{u}) \overset{\text{def}}{=} \mathtt{lp}_i(\mathtt{u}) \vee \mathtt{rp}_i(\mathtt{u})$ and $\mathtt{dist}_i(j+1, \mathtt{u}) \overset{\text{def}}{=} \exists\, \overline{\mathtt{u}}\ (\mathtt{u} \hookrightarrow \overline{\mathtt{u}}) \wedge \mathtt{dist}_i(j, \overline{\mathtt{u}})$ for all $j \geq 0$.

Lemma 12. Let $\mathfrak{h}$ be a heap, $\mathfrak{f}$ be a valuation and $i \geq 0$. Then, $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{onp}_i(\mathtt{u})$ iff $\mathfrak{f}(\mathtt{u})$ is on some i-parenthesis in $\mathfrak{h}$.

5.2 The role of parentheses

First, we introduce the interval $[1, K]$ ($K \in \mathbb{N} \setminus \{0\}$) assuming that for each $j \in [1, K]$, either $\mathtt{P}_j$ or $\mathtt{u}_j$ occurs in the DSOL formula to be translated (but not both of them). So, the developments below are relative to a finite set of first-order and second-order variables and this is concretized by $[1, K]$ (always possible since a formula has a finite number of variables).

Let us come back to parentheses and assume that X is a subset of $[0, K]$. In an X-well-formed heap $\mathfrak{h}$ (see Definition 18 below), the parentheses play the following role. For each $j \in X$, we have the index location $\mathfrak{lp}_j$ from a distinguished left j-parenthesis and the index location $\mathfrak{rp}_j$ from a distinguished right j-parenthesis. Moreover, let $d^l_j = \widetilde{\sharp \mathfrak{lp}_j}$ and $d^r_j = \widetilde{\sharp \mathfrak{rp}_j}$ (in $\mathfrak{h}$). When $j \in X$ is related to a first-order variable, we require that $d^r_j = d^l_j + 2$ and there is an entry of degree $d^l_j + 1$ such that its element is understood as the interpretation of the variable $\mathtt{u}_j$ (see Figure 2 with $d^r_j = 5$ and $d^l_j = 3$). That explains why the parentheses are viewed as delimiters. Similarly, let $\{(\mathfrak{l}_1, \mathfrak{l}'_1), \ldots, (\mathfrak{l}_\beta, \mathfrak{l}'_\beta)\}$ be a finite set of pairs of locations, understood as the interpretation of a second-order variable $\mathtt{P}_j$ with $j \in X$. In $\mathfrak{h}$, there are 2β entries whose respective degrees are exactly $\{d^l_j + 3(i-1)+1, d^l_j + 3(i-1)+2 : i \in [1, \beta]\}$ with $d^r_j = d^l_j + 3\beta + 1$. A pair of entries of respective degrees $d^l_j + 3(i-1)+1$ and $d^l_j + 3(i-1)+2$ have exactly as elements $\mathfrak{l}_i$ and $\mathfrak{l}'_i$ respectively, which allows to encode the pair $(\mathfrak{l}_i, \mathfrak{l}'_i)$. All this underlying encoding makes sense only if the left and right parentheses as well as the entries whose degrees are related to their degrees are uniquely determined (see Condition (1) in Definition 14, below). For this reason, we introduce a left 0-parenthesis and a right 0-parenthesis with $d^r_0 = d^l_0 + 1$ (0 is not a variable index), the degree d^l_0 is strictly greater than the degree of any location in the original heap, all degrees d^l_j with $j \neq 0$ are strictly greater than d^l_0 and finally, the above-mentioned entries and parentheses are the only ones with their respective degrees. This guarantees that any entry from a pair of entries with successive degrees serving for the interpretation of a second-order variable, cannot serve twice for another pair or for another variable. Below, we provide the technical developments. A heap $\mathfrak{h}$ is *made of entries and parentheses only* $\overset{\text{def}}{\Leftrightarrow}$ every location in dom($\mathfrak{h}$) belongs either to a left i-parenthesis for some $i \geq 0$, to a right i-parenthesis for some $i \geq 0$, or to an entry. Given a heap $\mathfrak{h}$ made of entries and parentheses only, we write $ispect(\mathfrak{h})$ to denote $\{\widetilde{\sharp \mathfrak{l}} : \mathfrak{l}$ is the index of some entry or parenthesis in $\mathfrak{h}\}$ and call this the *index spectrum* of $\mathfrak{h}$.

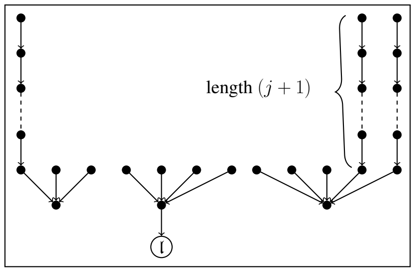


Figure 2: Encoding $[\mathtt{u}_j \mapsto \mathfrak{l}]$.

Let $\mathfrak{h}_B$ be a heap such that $\alpha = max(\{\widetilde{\sharp \mathfrak{l}} : \mathfrak{l} \in \mathbb{N}\})$. We have seen in Section 4 that it is possible to characterize the locations that witness this maximal value α thanks to $\mathtt{maxdeg(u)}$. A valuation heap $\mathfrak{h}_V$ for $\mathfrak{h}_B$ is made of entries and parentheses only whose degrees are greater than $max(3, \alpha + 1)$. The heap $\mathfrak{h}_V$ satisfies the following simple conditions (more constraints will follow): $min(ispect(\mathfrak{h}_V))$ is greater than $max(3, \alpha+1)$ and it is witnessed by the degree of some left 0-parenthesis; each degree in $ispect(\mathfrak{h}_V)$ is witnessed by exactly one entry or parenthesis. Formula $\mathtt{imin(u)}$ below is satisfied in $\mathfrak{h} = \mathfrak{h}_B \uplus \mathfrak{h}_V$ by a location $\mathfrak{l}$ witnessing the minimal value in $ispect(\mathfrak{h}_V)$:

$$\mathtt{imin(u)} \overset{\text{def}}{=} \mathtt{lp}_0\mathtt{(u)} \wedge (\forall\, \overline{\mathtt{u}}\; ((\overline{\mathtt{u}} \neq \mathtt{u}) \wedge \mathtt{lp}_0(\overline{\mathtt{u}})) \Rightarrow \sharp\overline{\mathtt{u}} < \sharp\mathtt{u})$$

Note that thanks to Section 4, we know that it is possible to compare numbers of predecessors as expressed above. So, $\mathtt{imin(u)}$ holds when $\mathfrak{f}(\mathtt{u})$ is the unique location that is the index of some left 0-parenthesis with greatest degree.

Lemma 13. Let $\mathfrak{f}$ be a valuation and $\mathfrak{h}$ be a heap. We have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{imin(u)}$ iff $\mathfrak{f}(\mathtt{u})$ is an index of some left 0-parenthesis and there is no other location $\mathfrak{l} \neq \mathfrak{f}(\mathtt{u})$ such that $\widetilde{\sharp \mathfrak{l}} \geq \widetilde{\sharp \mathfrak{f}(\mathtt{u})}$ and $\mathfrak{l}$ is the index of some left 0-parenthesis.

The proof is by an easy verification by using Lemma 11. Once a heap $\mathfrak{h}$ satisfies $\exists\ \mathtt{u}\ \mathtt{imin(u)}$, the unique location $\mathfrak{l}_0$ such that $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}_0]} \mathtt{imin(u)}$ (say with $\widetilde{\sharp \mathfrak{l}_0} = d_0$) plays the role of a delimiter between the original heap and the part of the heap than encodes the hybrid valuation. We have seen that an index spectrum is defined for heaps made of entries and parentheses only. This is fine, but below we adapt the definition to heaps $\mathfrak{h}$ satisfying $\exists\ \mathtt{u}\ \mathtt{imin(u)}$. Let us define the set $spect(\mathfrak{h})$ as

$$\{\widetilde{\sharp \mathfrak{l}} : \mathfrak{l} \text{ is an index of some entry or parenthesis in } \mathfrak{h}\} \cap [d_0, +\infty[$$

The set $spect(\mathfrak{h})$ is called the *spectrum* of $\mathfrak{h}$. This illustrates how $\mathfrak{l}_0$ and $\widetilde{\sharp \mathfrak{l}_0} = d_0$ play the role of separator between the original heap and the valuation heap.

The subheap encoding the valuation is made of parentheses and entries and we shall need to identify the indices of such patterns. The formula $\mathtt{lindex(u)}$ defined below does the job:

$$(\exists\, \overline{\mathtt{u}}\ \mathtt{imin}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} \leq \sharp\mathtt{u}) \wedge$$

$$\Big(\Big(\bigvee_{i \in [0,K]} (\mathtt{lp}_i\mathtt{(u)} \vee \mathtt{rp}_i\mathtt{(u)})\Big) \vee \mathtt{eindex(u)}\Big)$$

($\mathtt{u}$ is interpreted as a *large index*). Entries and parentheses with large indices are also called large entries and parentheses, respectively. It is easy to define a large index that is also the index of a left [resp. right] parenthesis. Let $\mathtt{llp}_i\mathtt{(u)} \overset{\text{def}}{=} \mathtt{lindex(u)} \wedge \mathtt{lp}_i\mathtt{(u)}$ and $\mathtt{lrp}_i\mathtt{(u)} \overset{\text{def}}{=} \mathtt{lindex(u)} \wedge \mathtt{rp}_i\mathtt{(u)}$ (see Lemma 11). The large index with a maximal degree can be also characterized as follows:

$$\mathtt{maxlindex(u)} \overset{\text{def}}{=} (\forall\, \overline{\mathtt{u}}\ \mathtt{lindex}(\overline{\mathtt{u}}) \Rightarrow (\sharp\overline{\mathtt{u}} \leq \sharp\mathtt{u})) \wedge \mathtt{lindex(u)}$$

Below, we state how the parentheses are organized.

Definition 14. Let $X = \{i_0, \ldots, i_s\} \subseteq [0, K]$ with $0 = i_0 < i_1 < \cdots < i_s$. A heap $\mathfrak{h}$ is X*-almost-well-formed* $\overset{\text{def}}{\Leftrightarrow}$

1. For every $j \in [0, s]$, there is a unique location $\mathfrak{l}^l_j$ [resp. $\mathfrak{l}^r_j$] such that $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}^l_j]} \mathtt{llp}_{i_j}\mathtt{(u)}$ [resp. $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}^r_j]} \mathtt{lrp}_{i_j}\mathtt{(u)}$].
2. For every $j \in [0, s]$, $\widetilde{\sharp \mathfrak{l}^l_j} < \widetilde{\sharp \mathfrak{l}^r_j}$, and $\widetilde{\sharp \mathfrak{l}^r_0} = \widetilde{\sharp \mathfrak{l}^l_0} + 1$.
3. For every $j \in [1, s]$, we have $\widetilde{\sharp \mathfrak{l}^l_j} = \widetilde{\sharp \mathfrak{l}^r_{j-1}} + 1$.
4. $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}^r_s]} \mathtt{maxlindex(u)}$.
5. For every $j \in [1, s]$, if i_j is the index of a first-order variable, then $\widetilde{\sharp \mathfrak{l}^l_j} = \widetilde{\sharp \mathfrak{l}^r_j} - 2$ (see Figure 2).
6. For every $j \in ([1, K] \setminus X)$, there is no location $\mathfrak{l}$ such that $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}]} \mathtt{llp}_j\mathtt{(u)} \vee \mathtt{lrp}_j\mathtt{(u)}$.

The degrees are organized as follows and they all belong to the spectrum of $\mathfrak{h}$ (below we let $d^l_j = \widetilde{\sharp \mathfrak{l}^l_j}$ and $d^r_j = \widetilde{\sharp \mathfrak{l}^r_j}$).

$$\begin{array}{ccccccccccccccc}
\models \mathtt{imin(u)} & & & & & & & & & & & & & & \models \mathtt{maxlindex(u)} \\
d^l_0 & < & d^r_0 & < & d^l_1 & < & d^r_1 & < & d^l_2 & < & d^r_2 & < \cdots < & d^l_s & < & d^r_s \\
 & & \| & & \| & & & & \| & & & & \| & & \\
 & & d^l_0 + 1 & & d^r_0 + 1 & & & & d^r_1 + 1 & & & & d^r_{s-1} + 1 & &
\end{array}$$

Lemma 15. There is $\mathtt{awfh}_X$ in 1SL2($-\!\!*$) such that $\mathfrak{h} \models \mathtt{awfh}_X$ iff $\mathfrak{h}$ is X-almost-well-formed.

Let $\mathfrak{h}$ be an X-almost-well-formed heap for some $\{0\} \subseteq X \subseteq [0, K]$ and $i \in X$. We write $\mathtt{vind}_i\mathtt{(u)}$ to denote

$$\mathtt{lindex(u)} \wedge \mathtt{eindex(u)} \wedge$$
$$(\exists\, \overline{\mathtt{u}}\ \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} < \sharp\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{lrp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} > \sharp\mathtt{u})$$

It characterizes indices whose degree is strictly between the degree of some large left i-parenthesis and the degree of some large right i-parenthesis. We write $degrees(i, \mathfrak{h})$ to denote the set: $\{\widetilde{\sharp\mathfrak{l}} : \mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}]} \mathtt{vind}_i(\mathtt{u}),\ \mathfrak{l} \in \mathbb{N}\}$.

Lemma 16. Let $\mathfrak{h}$ be such that $\mathfrak{h} \models \exists\,\mathtt{u}\ \mathtt{imin}(\mathtt{u})$ and $i \geq 0$ be such that there are unique locations $\mathfrak{lp}$ and $\mathfrak{rp}$ with $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{lp}]} \mathtt{llp}_i(\mathtt{u})$ and $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{rp}]} \mathtt{lrp}_i(\mathtt{u})$. For every $\mathfrak{l} \in \mathbb{N}$, we have $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}]} \mathtt{vind}_i(\mathtt{u})$ iff $\mathfrak{l}$ is the index of some entry and $\widetilde{\sharp\mathfrak{lp}} < \widetilde{\sharp\mathfrak{l}} < \widetilde{\sharp\mathfrak{rp}}$.

The translation of $\mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$ is therefore: $\exists\,\mathtt{u}\ (\mathtt{on}_j(\mathtt{u}) \wedge \exists\,\overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \mathtt{vind}_i(\overline{\mathtt{u}}) \wedge \exists\,\mathtt{u}\ (\sharp\mathtt{u} = \sharp\overline{\mathtt{u}} + 1 \wedge \mathtt{vind}_i(\mathtt{u}) \wedge \exists\,\overline{\mathtt{u}}\ (\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \mathtt{on}_k(\overline{\mathtt{u}})))))$, where $\mathtt{on}_j(\mathtt{u}) \overset{\text{def}}{=} \exists\,\overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \wedge \mathtt{vind}_j(\overline{\mathtt{u}})$. Formula $\mathtt{on}_j(\mathtt{u})$ holds true when $\mathtt{u}$ is interpreted as the element of the unique entry attached to $\mathtt{u}_j$. These definitions take advantage of the fact that there are unique large left and right parentheses for each variable index. Figure 3 illustrates the constraints when $j < i < k$. From left to right, it represents explicitly a left j-parenthesis, then a right j-parenthesis, then a left i-parenthesis, a right i-parenthesis and a left k-parenthesis, followed finally by a right k-parenthesis. Other entries and parentheses are present in the figure, but they are represented by dots in order to focus on the memory cells relevant to evaluate the formula obtained by translation of $\mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$. The degrees of parentheses and entries increase from left to right.

5.3 Taking care of valuations

Now that we have a way of identifying that part of the heap that encodes our valuation, we turn our attention to encoding the valuation itself. Below, we introduce a condition for a subheap to be "glued" to an existing valuation. We distinguish three cases. A *local* 0*-valuation* is a heap made of a left 0-parenthesis of degree d and a right 0-parenthesis of degree $d + 1$ only, for some $d \geq 3$. Let $i \in [1, K]$ be the index of some first-order variable. A *local i-valuation* is a heap made of a left i-parenthesis of degree d, an entry of degree $d+1$ and a right i-parenthesis of degree $d+2$ only, for some $d \geq 3$. Let $i \in [1, K]$ be the index of some second-order variable. A *local i-valuation* is a heap $\mathfrak{h}$ such that

1. every location $\mathfrak{l}$ in dom($\mathfrak{h}$) belongs either to a left i-parenthesis, to a right i-parenthesis, or to an entry,
2. $\mathfrak{h}$ contains a unique left [resp. right] i-parenthesis,
3. $min(ispect(\mathfrak{h}))$ is the degree of some left i-parenthesis,
4. $max(ispect(\mathfrak{h}))$ is the degree of some right i-parenthesis,
5. $ispect(\mathfrak{h})$ is of the form below for some $\alpha \geq 3$, $\beta \geq 0$, $\{\alpha\} \cup \{\alpha+3(i-1)+1, \alpha+3(i-1)+2 : i \in [1, \beta]\} \cup \{\alpha+3\beta+1\}$ (when $\beta = 0$, $ispect(\mathfrak{h})$ is equal to $\{\alpha, \alpha+1\}$),
6. there are no two distinct indices with the same degree.

Since local i-valuations are typically heaps that are added to the current heap to encode the interpretation of a variable, it is essential to be able to characterize them by 1SL2($-\!\!*$) formulae. This is the purpose of the result below.

Lemma 17. Let $i \in [0, K]$. There is $\mathtt{localval}_i(\mathtt{u})$ in 1SL2($-\!\!*$) such that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{localval}_i(\mathtt{u})$ iff $\mathfrak{h}$ is a local i-valuation and $\mathfrak{f}(\mathtt{u})$ is the index of its left i-parenthesis.

The definition for X-almost-well-formed heaps mainly takes care of parentheses. In Definition 18, constraints on the degrees of large indices are specified.

Definition 18. Let $X = \{i_0, \ldots, i_s\} \subseteq [0, K]$ with $0 = i_0 < i_1 < \cdots < i_s$. A heap $\mathfrak{h}$ is *X-well-formed* $\overset{\text{def}}{\Leftrightarrow}$

1. $\mathfrak{h}$ is X-almost-well-formed,
2. for every $j \in [1, s]$, if i_j is the index of a first-order variable, then $degrees(i_j, \mathfrak{h})$ is a singleton,
3. for every $j \in [1, s]$, if i_j is the index of a second-order variable, $degrees(i_j, \mathfrak{h})$ is the set below for some $\alpha_j \geq 3$, $\beta_j \geq 0$: $\{\alpha_j + 3(i-1)+1, \alpha_j + 3(i-1)+2 : i \in [1, \beta_j]\}$,
4. for every location $\mathfrak{l}$ such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}]} \mathtt{lindex}(\mathtt{u})$, there is no $\mathfrak{l}' \neq \mathfrak{l}$ such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}']} \mathtt{lindex}(\mathtt{u})$ and $\widetilde{\sharp\mathfrak{l}} = \widetilde{\sharp\mathfrak{l}'}$.

When $\mathfrak{h}$ is X-well-formed, we write $\mathfrak{h} = \mathfrak{h}_B \uplus \mathfrak{h}_V$ such that dom($\mathfrak{h}_V$) is made of entries and parentheses of degree $d \geq \widetilde{\sharp\mathfrak{l}_0}$ for some $\mathfrak{l}_0 \in \mathbb{N}$ such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}_0]} \mathtt{imin}(\mathtt{u})$. By Definition 18, we have $spect(\mathfrak{h}) = ispect(\mathfrak{h}_V)$ and clearly the decomposition is unique since $\mathfrak{l}_0$ is unique.

Again, well-formed heaps can be characterized by formulae in 1SL2($-\!\!*$) whose size is linear in K.

Lemma 19. Given $\{0\} \subseteq X \subseteq [0, K]$, there is a formula $\mathtt{wfh}_X$ in 1SL2($-\!\!*$) such that $\mathfrak{h} \models \mathtt{wfh}_X$ iff $\mathfrak{h}$ is X-well-formed.

Let us define a valuation from a valuation heap.

Definition 20. Let $\mathfrak{h}$ be an X-well-formed heap for some $\{0\} \subseteq X \subseteq [0, K]$. For every second-order $i \in X$, we define $\mathfrak{V}_{\mathfrak{h}}(\mathtt{P}_i) \overset{\text{def}}{=} \{(\mathfrak{h}_V(\mathfrak{l}), \mathfrak{h}_V(\mathfrak{l}')) : \widetilde{\sharp\mathfrak{l}'} = \widetilde{\sharp\mathfrak{l}} + 1,\ \widetilde{\sharp\mathfrak{l}}, \widetilde{\sharp\mathfrak{l}'} \in degrees(i, \mathfrak{h}),\ \mathfrak{l}, \mathfrak{l}'$ are index locations$\}$ and for every first-order $i \in X$, $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i) \overset{\text{def}}{=} \mathfrak{h}_V(\mathfrak{l})$ where $\mathfrak{l}$ is the unique index location such that $\widetilde{\sharp\mathfrak{l}} \in degrees(i, \mathfrak{h})$. We say that $\mathfrak{V}_{\mathfrak{h}}$ is the valuation *extracted* from $\mathfrak{h}$.

Below, we present an essential result stating how heaps can be composed when a new variable needs to be interpreted. The formulae involved to compose the X-well-formed heap $\mathfrak{h}$ and the local i-valuation heap $\mathfrak{h}'$ are directly used in the translation of quantified formulae (see Section 5.4). Lemma 21 is only used in the proof of Lemma 22 but it is stated here in order to emphasize how the heaps can be safely composed.

Lemma 21 (Composition)**.** Let $\mathfrak{f}$ be a valuation, $\mathfrak{h}$ be an X-well-formed heap with $\{0\} \subseteq X \subseteq [0, K]$, $i \in [1, K] \setminus X$ with $i > max(X)$, and $\mathfrak{h}'$ be a disjoint heap such that:

1. $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{imin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})$,
2. $\mathfrak{h}' \models_{\mathfrak{f}} \mathtt{localval}_i(\overline{\mathtt{u}})$,
3. $\mathfrak{h} \uplus \mathfrak{h}' \models_{\mathfrak{f}} \mathtt{wfh}_{X\cup\{i\}} \wedge \mathtt{imin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}})$.

Then, $spect(\mathfrak{h} \uplus \mathfrak{h}') = spect(\mathfrak{h}) \uplus ispect(\mathfrak{h}')$.

The proof of Lemma 21 is quite combinatorial and this is the place where we check that the original heap cannot be confused with the valuation heap (and the other way around). It is important to guarantee, as the proof does, that adding a new part of the valuation does not destroy what has been built so far.

5.4 A reduction from DSOL into 1SL2($-\!\!*$)

Below, we define a translation from a sentence ϕ in DSOL into a sentence in 1SL2($-\!\!*$) that uses only logspace. Without any loss of generality, we assume that (1) two occurrences of quantified variables in ϕ have distinct variable indices (e.g., $\mathtt{P}_4$ and $\mathtt{u}_4$ cannot both occur in ϕ and "$\forall\ \mathtt{u}_4$" cannot occur more than once) and (2) if $\exists\ \mathtt{u}_i\ \psi_1$ is a subformula of $\exists\ \mathtt{u}_j\ \psi_2$, then $i > j$ and this holds for any combination of first-order/second-order variables. We may rename variables so that these conditions are satisfied. We assume that the variable indices are among $[1, K]$.

The translation of the formula ϕ, written $\mathtt{T}(\phi)$, first applies a top-level translation $t_{top}(\cdot)$ which takes care of initializing the valuation heap; then, a recursive map $t(\cdot)$ is applied. So, $\mathtt{T}(\phi) \overset{\text{def}}{=} t_{top}(\phi)$ where $t_{top}(\phi)$ is defined as follows:

$$t_{top}(\phi) \overset{\text{def}}{=} \exists\,\mathtt{u}\ \mathtt{isoloc}(\mathtt{u}) \wedge (\mathtt{localval}_0(\mathtt{u}) \overset{\rightarrow}{-\!\!*} (\mathtt{wfh}_{\{0\}} \wedge$$

$$\mathtt{imin}(\mathtt{u}) \wedge (\forall\,\overline{\mathtt{u}}\ ((\mathtt{u} \neq \overline{\mathtt{u}}) \wedge \neg\mathtt{lrp}_0(\overline{\mathtt{u}})) \Rightarrow (\sharp\overline{\mathtt{u}} < \sharp\mathtt{u})) \wedge t(\{0\}, \phi)))$$

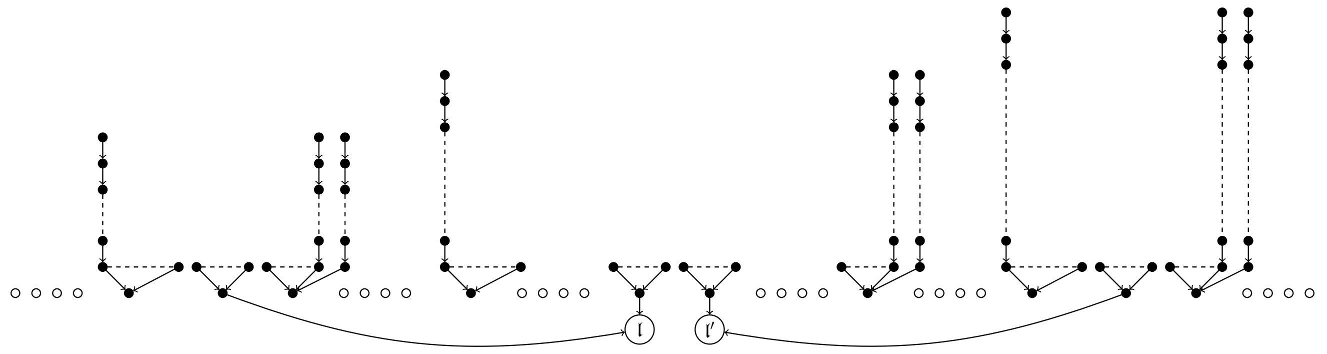


Figure 3: How the translation of $\mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$ works ($j < i < k$): $(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{V}_{\mathfrak{h}}(\mathtt{P}_i)$.

The first step of the translation consists in adding 0-parentheses so that the heap that evaluates $t(\{0\}, \phi)$ is $\{0\}$-well-formed. The recursive map $t(\cdot)$ is inductively defined as follows ($X \subseteq [0, K]$, ψ subformula of ϕ):

- $t(X, \cdot)$ is homomorphic for Boolean connectives,
- $t(X, \mathtt{u}_i = \mathtt{u}_j) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ \mathtt{on}_i(\mathtt{u}) \wedge \mathtt{on}_j(\mathtt{u})$,
- $t(X, \mathtt{u}_i \hookrightarrow \mathtt{u}_j) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ \exists\, \overline{\mathtt{u}}\ (\mathtt{on}_i(\mathtt{u}) \wedge \mathtt{on}_j(\overline{\mathtt{u}}) \wedge \mathtt{u} \hookrightarrow \overline{\mathtt{u}})$,
- $t(X, \mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ (\mathtt{on}_j(\mathtt{u}) \wedge \exists\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \mathtt{vind}_i(\overline{\mathtt{u}}) \wedge \exists\, \mathtt{u}\ (\sharp\mathtt{u} = \sharp\overline{\mathtt{u}} + 1 \wedge \mathtt{vind}_i(\mathtt{u}) \wedge \exists\, \overline{\mathtt{u}}\ (\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \mathtt{on}_k(\overline{\mathtt{u}})))))$.
- For the quantifier $\exists\, \mathtt{u}_i$, we choose $\mathfrak{l}$ and $\mathfrak{l}'$ such that $\mathfrak{l}$ is the index of the left 0-parenthesis and $\mathfrak{l}'$ is an isolated location. We construct a new heap that is a local i-valuation while enforcing that the index of the left 0-parenthesis is preserved and $\mathfrak{l}'$ becomes the index of the unique large left i-parenthesis (see Lemma 21). $t(X, \exists\, \mathtt{u}_i\ \psi) \stackrel{\text{def}}{=}$

$$\exists\, \mathtt{u}\ \exists\, \overline{\mathtt{u}}\ ((\mathtt{imin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})) \wedge (\mathtt{localval}_i(\overline{\mathtt{u}}) \mathrel{-\!\!\circledast} (\mathtt{wfh}_{X \cup \{i\}} \wedge \mathtt{imin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge t(X \cup \{i\}, \psi))))$$

- The translation with second-order variables is analogous (the formula $\mathtt{localval}_i(\overline{\mathtt{u}})$ is actually defined differently, see the proof of Lemma 17): $t(X, \exists\, \mathtt{P}_i\ \psi) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ \exists\, \overline{\mathtt{u}}\ ((\mathtt{imin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})) \wedge (\mathtt{localval}_i(\overline{\mathtt{u}}) \mathrel{-\!\!\circledast} (\mathtt{wfh}_{X \cup \{i\}} \wedge \mathtt{imin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge t(X \cup \{i\}, \psi))))$.

Every subformula $t(X, \psi)$ has no free variable from $\mathtt{fr}(\psi) \subseteq X$ where $\mathtt{fr}(\psi)$ denotes the set of variable indices in ψ from either first-order or second-order *free* variables. As noted by one anonymous referee, a standard trick is to convert first-order variables into second-order ones so that the proof has only to deal with one type of variable. Herein, we do not quite eliminate first-order variables but we provide a uniform treatment for first-order quantifications and second-order quantifications, which essentially amounts to dealing with a single type of encoding.

Below, we state the correctness lemma that allows us to get Theorem 23.

Lemma 22. Let ϕ be a DSOL sentence of the above form, ψ be one of its subformulae and $(\mathtt{fr}(\psi) \cup \{0\}) \subseteq X \subseteq [0, K]$. Let $\mathfrak{h} = \mathfrak{h}_B \uplus \mathfrak{h}_V$ be a X-well-formed heap and $\mathfrak{V}_{\mathfrak{h}}$ be the valuation extracted from $\mathfrak{h}$. Then, $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h}}} \psi$ iff $\mathfrak{h} \models t(X, \psi)$.

Here is the main result of the paper.

Theorem 23. For every sentence ϕ in DSOL, for every heap $\mathfrak{h}$, we have $\mathfrak{h} \models \phi$ iff $\mathfrak{h} \models \mathtt{T}(\phi)$, so WSOL and 1SL2($-\!\!\circledast$) have the same expressive power.

Observe that $\mathtt{T}(\phi)$ can be computed in logspace (to do this, one must check the size of all the formulae built in the proofs). So, the restriction to two variables does not reduce the expressive power, unlike restrictions in [14], for instance.

We get the ultimate undecidability result below (no separating conjunction, two quantified variables, one record field).

Corollary 24. Satisfiability problem for 1SL2($-\!\!\circledast$) is undecidable.

The absence of program variables in 1SL2($-\!\!\circledast$) makes the proof of Corollary 24 even more difficult to design, which is perfect to obtain the sharpest undecidability result. An expressivity result with program variables is briefly presented in Section 5.5.

It is also possible to establish the following consequences.

Corollary 25. (I) Let ϕ be a sentence in 1SL. There is an equivalent sentence in 1SL2($-\!\!\circledast$) of polynomial size in the size of ϕ. (II) 1SL2($-\!\!\circledast$) is strictly more expressive than 1SL($*$).

Corollary 25(II) follows from Theorem 23 and 1SL($*$) is strictly less expressive than MSO [1, Corollary 5.3].

Our main results are Theorem 23 and Corollary 24, significantly improving previously known results (see the figure in Section 1). As far as the translation into 1SL2($-\!\!\circledast$) is concerned (see the current section but it uses in essential ways the formulae of Section 4), the lack of variables is partially compensated by the introduction of the parentheses in order to constrain sufficiently the valuation heap. More importantly, we have shown that this is a viable solution in 1SL2($-\!\!\circledast$) despite only having two variables (see the proof of Lemma 21). This was not at all clear at the outset, and of course, in view of the complexity of the final proof, this led to lengthy arguments to show correctness of the whole enterprise.

5.5 Adding an unbounded number of program variables

We consider 1SL with program variables, which is a strict extension of 1SL and therefore undecidability for 1SL2($-\!\!\circledast$) is still valid in the presence of program variables. Let $\mathtt{PVAR} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$ be a countably infinite set of *program variables*. A *memory state* is a pair $(\mathfrak{s}, \mathfrak{h})$ such that $\mathfrak{s} : \mathtt{PVAR} \to \mathbb{N}$ and $\mathfrak{h}$ is a heap. Formulae of *1SL with program variables* are built from expressions of the form $e ::= \mathtt{x} \mid \mathtt{u}$ where $\mathtt{x} \in \mathtt{PVAR}$ and $\mathtt{u} \in \mathtt{FVAR}$, and atomic formulae of the form $\pi ::= e = e' \mid e \hookrightarrow e'$. Formulae are defined by $\phi ::= \pi \mid \phi \wedge \psi \mid \neg\phi \mid \phi * \psi \mid \phi \mathrel{-\!\!\circledast} \psi \mid \exists\, \mathtt{u}\ \phi$, where $\mathtt{u} \in \mathtt{FVAR}$. A *valuation* is a map $\mathfrak{f} : \mathtt{FVAR} \to \mathbb{N}$. The satisfaction relation $\models$ is extended as follows:

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} e = e'$ iff $[e] = [e']$, with $[\mathtt{x}] \stackrel{\text{def}}{=} \mathfrak{s}(\mathtt{x})$, $[\mathtt{u}] \stackrel{\text{def}}{=} \mathfrak{f}(\mathtt{u})$. Obviously, program variables can be understood as free quantified variables interpreted rigidly.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} e \hookrightarrow e'$ iff $[e] \in \mathrm{dom}(\mathfrak{h})$ and $\mathfrak{h}([e]) = [e']$.

The satisfiability problem takes as input a sentence from 1SL with program variables, in which the only free variables are program variables. The version of DSOL with program variables is defined similarly when models are memory states.

Theorem 26. There is a translation $\mathtt{T}$ such that for every sentence ϕ in DSOL with program variables, the sentence $\mathtt{T}(\phi)$ in 1SL2($-\!\!*$) with program variables (of polynomial-size in the size of ϕ) is such that for all $(\mathfrak{s}, \mathfrak{h})$, we have $(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{T}(\phi)$.

Actually, the translation $\mathtt{T}$ is defined as a variant of the one in Section 5.4 that takes into account program variables. The variant is pretty natural since program variables do not require any encoding. For instance, the map $t(\cdot)$ is extended as follows (more details in the proof of Theorem 26): $t(X, \mathtt{u}_i = \mathtt{x}) \overset{\text{def}}{=} \exists\, \mathtt{u}\ (\mathtt{on}_i(\mathtt{u}) \wedge \mathtt{u} = \mathtt{x})$ and $t(X, \mathtt{u}_i \hookrightarrow \mathtt{x}) \overset{\text{def}}{=} \exists\, \mathtt{u}\ (\mathtt{on}_i(\mathtt{u}) \wedge \mathtt{u} \hookrightarrow \mathtt{x})$.

Extension with k record fields So far, memory cells have a unique record field but it is possible to extend our results to $k > 1$ record fields, along the lines of [7, Section 7]. Let kSL2($-\!\!*$) be the logic in which heaps are partial functions $\mathfrak{h} : \mathbb{N} \rightharpoonup \mathbb{N}^k$ with finite domain and atomic formulae include $\mathtt{u}_j \overset{i}{\hookrightarrow} \overline{\mathtt{u}}_{j'}$ ($i \in [1, k]$); kWSOL and kDSOL are defined similarly. One can show that every sentence in kDSOL has an equivalent sentence in kSL2($-\!\!*$). To do so, we need to adapt the definitions from Sections 3 and 4 so that memory cells involved in the valuation heap are relevant only with respect to $\overset{1}{\hookrightarrow}$ (and comparing numbers of predecessors is performed only with respect to $\overset{1}{\hookrightarrow}$). Details are tedious because many notions need to be redefined relatively to $\overset{1}{\hookrightarrow}$ but the encoding of valuations is based on the same general principles as for 1SL2($-\!\!*$).

6. Conclusion

We have shown that 1SL2($-\!\!*$) is as expressive as weak second-order logic on concrete heaps (Theorem 23). As a consequence, the satisfiability problem for 1SL2($-\!\!*$) is undecidable (Corollary 24) and we have identified the undecidable core of separation logic, apart from illustrating further the power of separating implication when interpreted on concrete heaps. We only use two variables, and our results also exclude separating conjunction, which is quite remarkable in view of the restricted number of variables. As far as the proofs are concerned, we used first principles from [7] but we had to provide non-trivial adaptations to fit the restricted fragment 1SL2($-\!\!*$). However, this illustrates the robustness of those principles since they could be applied by using the proof techniques developed in the present paper. Other semantical variants are possible and will be the subject of future work.

Acknowledgments

We would like to thank the anonymous referees for their suggestions and remarks to improve the quality of the paper.

References

[1] T. Antonopoulos and A. Dawar. Separating graph logic from MSO. In *FOSSACS'09*, volume 5504 of *LNCS*, pages 63–77. Springer, 2009.

[2] T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich, and J. Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *FOSSACS'14*, volume 8412 of *LNCS*, pages 411–425. Springer, 2014.

[3] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

[4] M. Bozga, R. Iosif, and S. Perarnau. Quantitative separation logic and programs with lists. *Journal of Automated Reasoning*, 45(2):131–156, 2010.

[5] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Metric propositional neighborhood logics: Expressiveness, decidability, and undecidability. In *ECAI'10*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 695–700. IOS Press.

[6] R. Brochenin. *Separation Logic: Expressiveness, Complexity, Temporal Extension.* PhD thesis, LSV, ENS Cachan, September 2013.

[7] R. Brochenin, S. Demri, and E. Lozes. On the Almighty Wand. *Information and Computation*, 211:106–137, 2012.

[8] J. Brotherston and M. Kanovich. Undecidability of propositional separation logic and its neighbours. In *LICS'10*, pages 130–139. IEEE.

[9] C. Calcagno, P. O'Hearn, and H. Yang. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS'01*, volume 2245 of *LNCS*, pages 108–119. Springer, 2001.

[10] B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR'11*, volume 6901 of *LNCS*, pages 235–249. Springer, 2011.

[11] A. Dawar, P. Gardner, and G. Ghelli. Expressiveness and complexity of graph logic. *Information and Computation*, 205(3):263–310, 2007.

[12] S. Demri and M. Deters. Two-variable separation logic and its inner circle, September 2013. Under submission.

[13] S. Demri, D. Galmiche, D. Larchey-Wendling, and D. Mery. Separation logic with one quantified variable. In *CSR'14*, volume 8476 of *LNCS*, pages 125–138. Springer, 2014.

[14] K. Etessami, M. Vardi, and T. Wilke. First-order logic with two variables and unary temporal logics. In *LICS'97*, pages 228–235. IEEE, 1997.

[15] D. Gabbay. Expressive functional completeness in tense logic. In *Aspects of Philosophical Logic*, pages 91–117. Reidel, 1981.

[16] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic - Mathematical Foundations and Computational Aspects, Vol. 1.* OUP, 1994.

[17] D. Galmiche and D. Méry. Tableaux and resource graphs for separation logic. *Journal of Logic and Computation*, 20(1):189–231, 2010.

[18] E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Arch. Mathematical Logic*, 38(4–5):313–354, 1999.

[19] C. Haase, S. Ishtiaq, J. Ouaknine, and M. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *CAV'13*, volume 8044 of *LNCS*, pages 790–795. Springer, 2013.

[20] Z. Hou, R. Clouston, R. Goré, and A. Tiu. Proof search for propositional abstract separation logics via labelled sequents. In *POPL'14*, pages 465–476. ACM, 2014.

[21] N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL'04*, volume 3210 of *LNCS*, pages 160–174. Springer, 2004.

[22] R. Iosif, A. Rogalewicz, and J. Simacek. The tree width of separation logic with recursive definitions. In *CADE'13*, volume 7898 of *LNCS*, pages 21–38. Springer, 2013.

[23] V. Kuncak and M. Rinard. On spatial conjunction as second-order logic. Technical Report MIT–CSAIL–TR–2004–067, MIT CSAIL, October 2004.

[24] D. Larchey-Wendling and D. Galmiche. The undecidability of Boolean BI through phase semantics. In *LICS'10*, pages 140–149. IEEE, 2010.

[25] W. Lee and S. Park. A proof system for separation logic with magic wand. In *POPL'14*, pages 477–490. ACM, 2014.

[26] C. Lutz, U. Sattler, and F. Wolter. Modal logic and the two-variable fragment. In *CSL'01*, volume 2142 of *LNCS*, pages 247–261. Springer.

[27] R. Piskac, T. Wies, and D. Zufferey. Automating separation logic using SMT. In *CAV'13*, volume 8044 of *LNCS*, pages 773–789. Springer.

[28] J. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.

[29] B. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *AMS Translations, Series 2*, 23:1–5, 1963.

[30] P. Weis. *Expressiveness and Succinctness of First-Order Logic on Finite Words.* PhD thesis, University of Massachussetts, 2011.