Two-Variable Separation Logic and Its Inner Circle

STEPHANE DEMRI, New York University, USA & CNRS, France
MORGAN DETERS, New York University, USA

Separation logic is a well-known assertion language for Hoare-style proof systems. We show that first-order separation logic with a unique record field restricted to two quantified variables and no program variables is undecidable. This is among the smallest fragments of separation logic known to be undecidable, and this contrasts with decidability of two-variable first-order logic. We also investigate its restriction by dropping the magic wand connective, known to be decidable with non-elementary complexity, and we show that the satisfiability problem with only two quantified variables is not yet elementary recursive. Furthermore, we establish insightful and concrete relationships between two-variable separation logic and propositional interval temporal logic (PITL), data logics, and modal logics, providing an inner circle of closely-related logics.

Categories and Subject Descriptors: F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Logics of Programs

General Terms: Theory, Verification

Additional Key Words and Phrases: Separation logic, two-variable logics, interval temporal logic, modal logic, data logic, decidability, complexity

ACM Reference Format:
Stéphane Demri, Morgan Deters, January 21st, 2015 [paper accepted], November 14th, 2014 [revised submission]. Initial submission: September 13th, 2013. Two-Variable Separation Logic and Its Inner Circle *ACM Trans. Comput. Logic* V, N, Article A (January YYYY), 37 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Bounding the number of variables. The satisfiability problem for the two-variable fragment of first-order logic (FO2) is decidable [Scott 1962; Mortimer 1975], and it is NEXPTIME-complete [Lewis 1980; Grädel et al. 1997a], even though the extension with three variables is known to be undecidable [Kahr et al. 1962]. Bounding the number of variables to find decidable fragments is usually motivated by identifying maximal fragments that are decidable or to determine a rough boundary between decidability and undecidability. It is often observed that the critical zone for the decidability border lies between two and three variables for first-order dialects; for instance, the two-variable fragment of first-order intuitionistic logic (with constant domain) has been shown undecidable in [Gabbay and Shehtman 1993], whereas FO2 extended with weak forms of recursion such as transitive closure operators is also undecidable [Grädel et al. 1999]. Similarly, monodic two-variable first-order linear-time temporal logic with equality is undecidable [Degtyarev et al. 2002]. By contrast, decidability results of FO2 interpreted over linear structures can be found in [Otto 2001]; FO2 on data words is de-

Work partially supported by the EU Seventh Framework Programme under grant agreement No. 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).

DOI:http://dx.doi.org/10.1145/0000000.0000000

cidable, too, and somewhat equivalent to the reachability problem for Petri nets [Bojańczyk et al. 2011]. There are also other decidable extensions of FO2, see [Pacholski et al. 1997; Grädel et al. 1997b; Kieroński et al. 2012; Szwast and Tendera 2013; Charatonik et al. 2014]. At the propositional level, bounding the number of propositional variables also makes sense, to restrict syntactic resources and to study its impact on the complexity of decision problems; see examples for modal and temporal logics in [Halpern 1995; Demri and Schnoebelen 2002]. Herein, we follow a similar path, and we consider first-order separation logic [Reynolds 2002].

Decidability and complexity issues for separation logics. Before going any further, let us recall briefly that the models (heaps) in separation logics can be understood as partial functions with profile $\mathfrak{h} : \mathbb{N} \rightharpoonup \mathbb{N}^k$ for some $k \geq 1$ and with finite domain (k corresponds to the number of record fields). The set of natural numbers behaves here as an abstraction for the sets of locations and values, respectively. Separating connectives are those in the logical formalism that operate on models by possibly updating them by addition or removal of elements in the domain of the heap (this is done in a controlled way, of course). Program variables are understood as first-order variables with a rigid interpretation (in contrast to quantified variables).

First-order separation logic is used as an assertion language for Hoare-style proof systems about programs with pointers [Reynolds 2002]. There is an ongoing quest to determine expressive fragments with relatively low complexity (see e.g. [Cook et al. 2011]) and to extend known decidability results (see e.g. [Iosif et al. 2013; Brotherston et al. 2014]). As far as decidability is concerned, first-order separation logic with two record fields (2SL) is shown undecidable in [Calcagno et al. 2001] (see also [Yang 2001, Section 8.1] or the undecidability results from [Brotherston and Kanovich 2014; Larchey-Wendling and Galmiche 2013], which are obtained in an alternative setting with propositional variables and no first-order quantification) and the proof does not require any use of separating connectives.

So, the result in [Calcagno et al. 2001] has been strengthened in [Brochenin et al. 2012] by showing that first-order separation logic with a unique record field (1SL) is also undecidable, which is obtained by showing that 1SL is equivalent to weak second-order logic (weakness refers to quantifications over finite sets only). Recently, in [Demri and Deters 2014], 1SL restricted to two quantified variables has also been shown to be equivalent to weak second-order logic, providing another undecidability proof but one more complex than what is presented below, since one needs to go through an encoding of weak second-order logic. If separating implication (also known as the "magic wand" operator) is dropped, decidability is regained but with non-elementary complexity as a consequence of [Rabin 1969; Stockmeyer 1974]. A natural continuation of this undecidability result, quite surprising considering that there is a single record field, would be to see how the undecidability is sensitive to the number of quantified variables, following the trend of works related to two-variable fragments of first-order dialects. Whereas 1SL restricted to one variable and augmented with program variables admits a decidable satisfiability problem and is the subject of another paper [Demri et al. 2014], we study herein the two-variable first-order separation logic without program variables; the atomic formulae are made of the equality predicate symbol and the points-to predicate only (no other predicate symbols are allowed), which is a considerably downgraded version of the full logic. Separating connectives include separating conjunction and separating implication.

Our contribution. The paper presents two main results and provides interesting relationships between separation logic [Reynolds 2002], interval temporal logic [Moszkowski 2004], data logics [Bojańczyk et al. 2011] and modal logics, see e.g. [Blackburn et al. 2001]. In that way, we improve our understanding about the expressive power of separation logic fragments.

(1) We show that two-variable first-order separation logic with a unique record field (1SL2) has an undecidable satisfiability problem by reduction from the halting problem for Minsky machines (Section 4). It is certainly minimal in terms of the number of quantified variables, considering that the case with one variable is decidable [Demri et al. 2014]. This concludes the classification of fragments of 1SL with respect to the number of variables. For first-order separation logic, undecidability already strikes with two variables (by contrast to the NEXPTIME-completeness of FO2), which could be explained by the fact that the magic wand can be used to simulate quantification of locations (see the proof in Section 4). In our proof, all the difficulties are concentrated on the use of only two variables: of course, we can take advantage of the recycling of variables as done for modal logics [Gabbay 1981], but this is not sufficient since we need to compare the neighborhood of locations that are not direct predecessors or successors. This is where the separating operators for separation logic are helpful, for instance to operate surgically on selections of the heap (see Section 4.4).
Additionally, we provide evidence that an undecidable logic on data words from [Bojańczyk et al. 2011] can be reduced to two-variable first-order separation logic (Section 4.6). Logics on data words have already been used to get undecidability results for separation logic with data fields, see e.g. [Bansal et al. 2009]. Herein, we use a simple version of first-order separation logic without program variables, without data fields, and apart from equality, there is only one binary relation, and it is functional and finite. This marks a substantial difference with existing work, and it highlights just how few our syntactic resources are while still getting undecidability and complexity lower bounds.
(2) Two-variable first-order separation logic with a unique record field and without the magic wand (1SL2($*$)) is known to be decidable (as a consequence of [Brochenin et al. 2012, Corollary 3.3]), but we establish non-elementary complexity by a reduction from Moszkowski's Propositional Interval Temporal Logic PITL (see e.g. [Moszkowski 2004]), despite the restriction on the number of variables. So, we are able to make an interesting bridge between interval temporal logics and separation logics (see Section 3). This is not completely surprising since there is a clear similarity between the "chop" operator in interval temporal logics and the separating conjunction in separation logic, but we are able to conclude an original result about a fragment of two-variable first-order separation logic. To our knowledge, this is the first time that the similarity has been turned into a concrete, interesting result. The possibility to relate separation logic and interval temporal logic has been already envisaged by Tony Hoare, see e.g. [Zhou Chaochen 2008].[1]
Finally, we refine this result by proposing a new simple modal logic MLH interpreted on heaps that can be translated into two-variable separation logic with a unique record field via a standard translation schema (see Section 2.4 and Section 3.4). For instance, we are able to show that MLH restricted to only the separating conjunction (i.e. without the magic wand operator) has non-elementary complexity. So, we are also happy to provide a modal logic closely related to two-variable separation logic, strengthening relationships between modal logics and two-variable fragments (see also [Lutz and Sattler 2002] in a more classical setting).

[1]We thank Ben Moszkowski for pointing us to this work.

Figure 1 diagrams the contributions of this paper and puts them in context with previously-known results. Logic definitions can be found in Sections 2, 3.1, and 4.6.

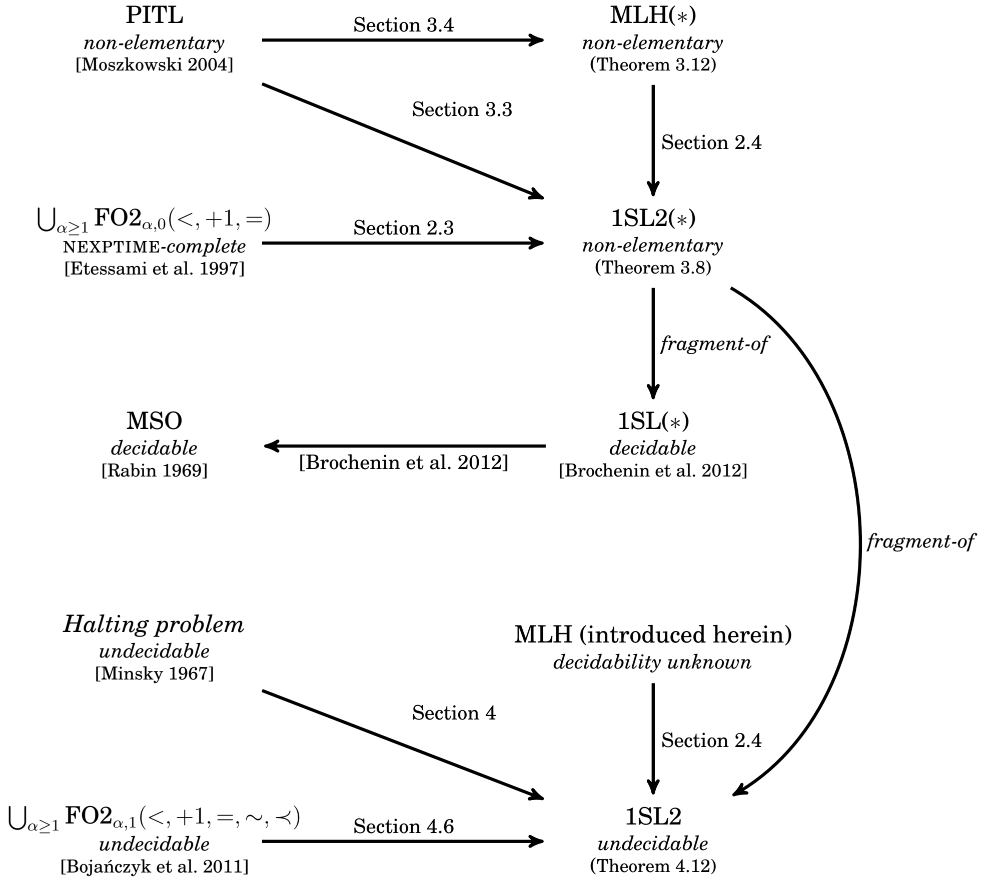

Fig. 1. A depiction of the contributions of this paper, in context, with both novel and previously-known results marked. Complexity and decidability results are shown for the satisfiability problem in each respective logic. Each arrow represents a known satisfiability-preserving translation of formulae. For example, any PITL formula ϕ can be translated into an MLH($*$) formula ψ such that ϕ is satisfiable in PITL if and only if ψ is satisfiable in MLH($*$); we abuse this notation to mean "encoding" in the case of our encoding of the halting problem for Minsky machines in 1SL2. 1SL2($*$) is a syntactic fragment of 1SL($*$), so certainly such a "translation" is possible.

Structure of the paper. Section 2 is primarily dedicated to the presentation of the separation logic 1SL, but it also explains how data words can be encoded in 1SL2($*$). This encoding will be useful many times in succeeding sections. Moreover, we introduce a new modal logic on heaps (MLH) that is a fragment of 1SL2($*$). Section 3 recalls the basics of propositional interval temporal logic PITL and we present an elementary satisfiability-preserving reduction from PITL to 1SL2($*$), leading to a non-elementary lower bound for 1SL2($*$). This result is strengthened by showing non-elementarity of MLH restricted to only the separating conjunction $*$. Section 4 contains a master reduction from the halting problem for Minsky machines into the satisfiability problem for 1SL2. To do so, several technical problems need to be solved and we dedicate one subsection to each of them. We also explain how undecidability results for first-order data logics can be used to obtain a similar result for 1SL2, which provides an alternative to the master reduction. Finally, Section 5 contains concluding remarks.

2. PRELIMINARIES

2.1. First-order separation logic with one record field (1SL)

A *heap* $\mathfrak{h}$ is a partial function $\mathfrak{h} : \mathbb{N} \rightharpoonup \mathbb{N}$ with finite domain. We write $\mathrm{dom}(\mathfrak{h})$ to denote its *domain* and $\mathrm{ran}(\mathfrak{h})$ to denote its *range*. Two heaps $\mathfrak{h}_1, \mathfrak{h}_2$ are said to be *disjoint*, denoted $\mathfrak{h}_1 \perp \mathfrak{h}_2$, if their domains are disjoint; when this holds, we write $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ to denote the heap obtained from $\mathfrak{h}_1$ and $\mathfrak{h}_2$ by taking their disjoint union. We use $\mathfrak{l}_i$, with $i, \mathfrak{l}_i \in \mathbb{N}$ and $i \geq 0$, to represent *locations*. We write $\mathfrak{l}_1 \rightarrow \mathfrak{l}_2 \rightarrow \cdots \rightarrow \mathfrak{l}_m$ (or, equivalently, $\mathfrak{l}_m \leftarrow \mathfrak{l}_{m-1} \leftarrow \cdots \leftarrow \mathfrak{l}_1$) 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 $\{\mathfrak{l}' : \mathfrak{h}(\mathfrak{l}') = \mathfrak{l}\}$ made of *predecessors* of $\mathfrak{l}$ (heap $\mathfrak{h}$ is implicit in the expression $\widetilde{\sharp}\mathfrak{l}$). A location $\mathfrak{l}$ is an *ancestor* of a location $\mathfrak{l}'$ iff there exists $i \geq 0$ such that $\mathfrak{h}^i(\mathfrak{l}) = \mathfrak{l}'$ where $\mathfrak{h}^i(\mathfrak{l})$ is shorthand for $\mathfrak{h}(\mathfrak{h}(\ldots(\mathfrak{h}(\mathfrak{l})\ldots)))$ (i applications of $\mathfrak{h}$ to $\mathfrak{l}$).

Usually in models for separation logic(s), memory states have a heap and a store for interpreting program variables, see e.g. [Reynolds 2002]. Our work concentrates on hardness results, and we are able to obtain these results without using such program variables. For this reason, we do not introduce them. Observe also that heaps will be understood as first-order structures of the form $(\mathbb{N}, \mathfrak{R})$ where $\mathfrak{R}$ is a finite and functional binary relation. Indeed, $\mathfrak{R} = \{(\mathfrak{l}, \mathfrak{h}(\mathfrak{l})) : \mathfrak{l} \in \mathrm{dom}(\mathfrak{h})\}$ for a heap $\mathfrak{h}$. A new modal logic with such frames is presented in Section 2.4. The locations $\mathfrak{l}$ and $\mathfrak{l}'$ are in the same *connected component* whenever $(\mathfrak{l}, \mathfrak{l}') \in (\mathfrak{R} \cup \mathfrak{R}^{-1})^*$. Usually, connected components are understood as non-singleton components. A finite functional graph $(\mathbb{N}, \mathfrak{R})$ can be made of several maximal connected subgraphs so that each connected subgraph is made of a cycle, possibly with trees attached to it. Figure 2 presents a heap, i.e. a finite functional graph on $\mathbb{N}$ with two maximal connected subgraphs.

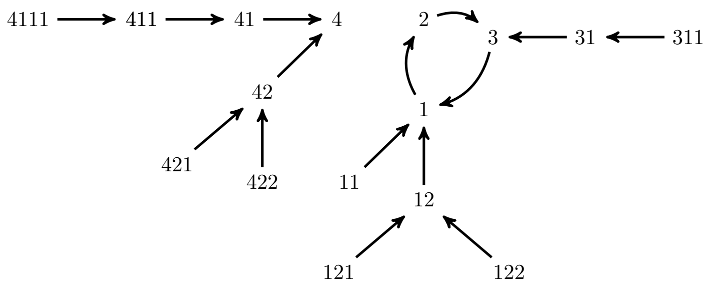

Fig. 2. A heap/finite functional graph with two maximal connected subgraphs.

Let $\mathtt{Var} = \{\mathtt{u}_1, \mathtt{u}_2, \ldots\}$ be a countably infinite set of quantified variables. Formulae of 1SL are defined by the abstract grammar below:

$$\phi \quad ::= \quad \mathtt{u}_i = \mathtt{u}_j \ \mid \ \mathtt{u}_i \hookrightarrow \mathtt{u}_j \ \mid \ \mathtt{emp} \ \mid \ \phi \wedge \phi \mid \ \neg\phi \mid \ \phi * \phi \mid \ \phi \mathrel{-\!\!*} \phi \ \mid \ \exists\, \mathtt{u}_i\ \phi$$

The connective $*$ is called the *separating conjunction* and the connective $\mathrel{-\!\!*}$ is called the *separating implication* (also known as the *magic wand*). We will make use of standard notations for the derived connectives.

An *assignment* is a map f of the form $\mathtt{Var} \rightarrow \mathbb{N}$. The satisfaction relation $\models$ is parameterized by assignments and defined as follows:

— $\mathfrak{h} \models_f \texttt{emp} \overset{\text{def}}{\Leftrightarrow} \text{dom}(\mathfrak{h}) = \emptyset$.
— $\mathfrak{h} \models_f \mathtt{u}_i = \mathtt{u}_j \overset{\text{def}}{\Leftrightarrow} f(\mathtt{u}_i) = f(\mathtt{u}_j)$.
— $\mathfrak{h} \models_f \mathtt{u}_i \hookrightarrow \mathtt{u}_j \overset{\text{def}}{\Leftrightarrow} f(\mathtt{u}_i) \in \text{dom}(\mathfrak{h})$ and $\mathfrak{h}(f(\mathtt{u}_i)) = f(\mathtt{u}_j)$.
— $\mathfrak{h} \models_f \phi_1 \wedge \phi_2 \overset{\text{def}}{\Leftrightarrow} \mathfrak{h} \models_f \phi_1$ and $\mathfrak{h} \models_f \phi_2$.
— $\mathfrak{h} \models_f \neg\phi \overset{\text{def}}{\Leftrightarrow} \mathfrak{h} \not\models_f \phi$.
— $\mathfrak{h} \models_f \phi_1 * \phi_2 \overset{\text{def}}{\Leftrightarrow}$ there exist $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h}_1 \perp \mathfrak{h}_2$, $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{h}_1 \models_f \phi_1$ and $\mathfrak{h}_2 \models_f \phi_2$.
— $\mathfrak{h} \models_f \phi_1 \mathrel{-\!\!*} \phi_2 \overset{\text{def}}{\Leftrightarrow}$ for all $\mathfrak{h}'$, if $\mathfrak{h} \perp \mathfrak{h}'$ and $\mathfrak{h}' \models_f \phi_1$ then $\mathfrak{h} \uplus \mathfrak{h}' \models_f \phi_2$.
— $\mathfrak{h} \models_f \exists\, \mathtt{u}_i\ \phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l} \in \mathbb{N}$ such that $\mathfrak{h} \models_{f[\mathtt{u}_i \mapsto \mathfrak{l}]} \phi$ ($f[\mathtt{u}_i \mapsto \mathfrak{l}]$ refers to a map equal to f except that $\mathtt{u}_i$ takes the value $\mathfrak{l}$).

Remark 2.1. Given a bijection $\sigma : \mathbb{N} \to \mathbb{N}$, we write $\mathfrak{h}' = \mathfrak{h} \circ \sigma$ to denote the heap whose graph is $\{(\sigma(\mathfrak{l}), \sigma(\mathfrak{h}(\mathfrak{l}))) : \mathfrak{l} \in \text{dom}(\mathfrak{h})\}$. Similarly, we write $f' = f \circ \sigma$ to denote the assignment such that $f'(\mathtt{u}_i) = \sigma(f(\mathtt{u}_i))$. Note that for all formulae ϕ, we have $\mathfrak{h} \models_f \phi$ iff $\mathfrak{h}' \models_{f'} \phi$. That is why heaps equipped with assignments are understood modulo bijections (i.e. concrete heaps are canonical elements of equivalence classes).

For a fixed $i \geq 0$, we write 1SLi to denote the fragment of 1SL restricted to i quantified variables and 1SL$i(*)$ to denote its restriction when the magic wand operator is disallowed.

Let $\mathcal{L}$ be a logic among 1SL, 1SLi, 1SL$i(*)$. The *satisfiability problem* for $\mathcal{L}$ takes as input a sentence ϕ from $\mathcal{L}$ and asks whether there is a heap $\mathfrak{h}$ such that $\mathfrak{h} \models \phi$ (regardless of assignment, as ϕ has no free variables).

THEOREM 2.2. *[Rabin 1969; Brochenin et al. 2012] The satisfiability problem for 1SL is undecidable, and the satisfiability problem for 1SL($*$) is decidable with non-elementary complexity.*

Decidability of 1SL($*$) is shown by (easy) reduction into weak monadic second-order logic of one unary (total) function with equality (referred to as MSO in Figure 1) that is shown decidable in [Börger et al. 1997, Corollary 7.2.11] by using [Rabin 1969].

2.2. Toolkit of formulae

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 switching the two variables.

Below, we define (standard) formulae and explain which properties they express.

— The domain $\text{dom}(\mathfrak{h})$ has exactly one location:

$$\texttt{size} = 1 \overset{\text{def}}{=} \neg\texttt{emp} \wedge \neg(\neg\texttt{emp} * \neg\texttt{emp})$$

— The domain $\text{dom}(\mathfrak{h})$ has exactly two locations:

$$\texttt{size} = 2 \overset{\text{def}}{=} (\neg\texttt{emp} * \neg\texttt{emp}) \wedge \neg(\neg\texttt{emp} * \neg\texttt{emp} * \neg\texttt{emp})$$

— $\mathtt{u}$ has a successor: $\texttt{alloc}(\mathtt{u}) \overset{\text{def}}{=} \exists\, \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$
— $\mathtt{u}$ has at least k predecessors: $\sharp\mathtt{u} \geq k \overset{\text{def}}{=} \overbrace{(\exists\, \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u}) * \cdots * (\exists\, \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u})}^{k \text{ times}}$
— $\mathtt{u}$ has at most k predecessors: $\sharp\mathtt{u} \leq k \overset{\text{def}}{=} \neg\,(\sharp\mathtt{u} \geq k+1)$
— $\mathtt{u}$ has exactly k predecessors: $\sharp\mathtt{u} = k \overset{\text{def}}{=} (\sharp\mathtt{u} \geq k) \wedge \neg(\sharp\mathtt{u} \geq k+1)$
— For all $\sim \in \{\leq, \geq, =\}$ and $i \geq 0$, we define the following formulae:

$$\begin{aligned}
\sharp\mathtt{u}^0 \sim k &\stackrel{\text{def}}{=} \sharp\mathtt{u} \sim k\\
\sharp\mathtt{u}^{i+1} \sim k &\stackrel{\text{def}}{=} \exists\, \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \sharp\overline{\mathtt{u}}^i \sim k\\
\sharp\mathtt{u}^{-i-1} \sim k &\stackrel{\text{def}}{=} \exists\, \overline{\mathtt{u}}\ \overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \sharp\overline{\mathtt{u}}^{-i} \sim k
\end{aligned}$$

For instance, $\sharp\mathtt{u}^6 \geq 5$ states that there is a (necessarily unique) location at distance 6 from $\mathtt{u}$ and its number of predecessors is greater than or equal to 5. The formula $\sharp\mathtt{u}^{-5} \leq 2$ states that there is a (not necessarily unique) location at distance -5 from $\mathtt{u}$ and its number of predecessors is not strictly greater than 2. For instance, $\sharp\mathtt{u}^1 \geq 1$ is logically equivalent to $\mathtt{alloc}(\mathtt{u})$.

— There is a non-empty path from $\mathtt{u}$ to $\overline{\mathtt{u}}$ and nothing else except loops that exclude $\overline{\mathtt{u}}$:

$$\begin{aligned}
\mathtt{ls}'(\mathtt{u},\overline{\mathtt{u}}) \stackrel{\text{def}}{=}\ & \sharp\mathtt{u} = 0 \wedge \mathtt{alloc}(\mathtt{u}) \wedge \neg\mathtt{alloc}(\overline{\mathtt{u}}) \wedge\\
& \forall\, \overline{\mathtt{u}}\ \left((\mathtt{alloc}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = 0) \Longrightarrow \overline{\mathtt{u}} = \mathtt{u}\right) \wedge\\
& \forall\, \mathtt{u}\ \left[(\sharp\mathtt{u} \neq 0 \wedge \mathtt{u} \neq \overline{\mathtt{u}}) \Longrightarrow (\sharp\mathtt{u} = 1 \wedge \mathtt{alloc}(\mathtt{u}))\right]
\end{aligned}$$

— There is a (possibly empty) path from $\mathtt{u}$ to $\overline{\mathtt{u}}$:

$$\mathtt{ls}(\mathtt{u},\overline{\mathtt{u}}) \stackrel{\text{def}}{=} \mathtt{u} = \overline{\mathtt{u}} \vee \left[\top * \mathtt{ls}'(\mathtt{u},\overline{\mathtt{u}})\right]$$

One can show that $\mathfrak{h} \models_f \mathtt{ls}(\mathtt{u},\overline{\mathtt{u}})$ iff there is $i \in \mathbb{N}$ such that $\mathfrak{h}^i(f(\mathtt{u})) = f(\overline{\mathtt{u}})$. The proof for this property can be found in [Brochenin et al. 2012, Lemma 2.4] (a similar property has been established for graph logics in [Dawar et al. 2007]).

— There is at most a single connected component (and nothing else):

$$\mathtt{1comp} \stackrel{\text{def}}{=} \neg\mathtt{emp} \wedge \exists\, \mathtt{u}\ \forall\, \overline{\mathtt{u}}\ \mathtt{alloc}(\overline{\mathtt{u}}) \Rightarrow \mathtt{ls}(\overline{\mathtt{u}},\mathtt{u})$$

— There are exactly two components: $\mathtt{2comps} \stackrel{\text{def}}{=} \mathtt{1comp} * \mathtt{1comp}$

Remark 2.3. The heap is a finite tree with at least two nodes can be expressed by the formula below:

$$\neg\mathtt{emp} \wedge \exists\, \mathtt{u}\ \neg\mathtt{alloc}(\mathtt{u}) \wedge (\forall\, \overline{\mathtt{u}}\ \mathtt{alloc}(\overline{\mathtt{u}}) \Rightarrow \mathtt{ls}(\overline{\mathtt{u}},\mathtt{u}))$$

Complexity results about two-variable fragments of first-order logic over finite trees can be found in [Benaim et al. 2013] but we cannot really take advantage of them since we do not use predicate symbols apart from equality and the points-to relation. By contrast, we do admit separating connectives.

Remark 2.4. Observe that all formulae in our toolkit above are in the 1SL2($*$) fragment. This is not by accident; we will study this fragment extensively in Section 3 and will need to employ several of the above formulae.

2.3. Encoding data words with multiple attributes

In this section, we present a simple encoding of data words with multiple attributes into heaps that will be useful in the rest of the paper. Finite data words [Bouyer 2002] are ubiquitous structures that include timed words, runs of Minsky machines, and runs of concurrent programs with an unbounded number of processes. These are finite words in which every position carries a label from a finite alphabet and a finite tuple of data values from some infinite alphabet. A wealth of specification formalisms for data words (and slight variants) has been introduced stemming from automata to adequate logical languages such as first-order logic [Bojańczyk et al. 2011; Schwentick and Zeume 2012] and temporal logics [Figueira 2010; Decker et al. 2014].

A *data word* of dimension β is a finite non-empty sequence in $([1, \alpha] \times \mathbb{N}^{\beta})^+$ for some $\alpha \geq 1$ and $\beta \geq 0$. The set $[1, \alpha]$ is understood as a finite alphabet of cardinal α whereas $\mathbb{N}$ is the infinite data domain. Data words of dimension zero are simply finite words over a finite alphabet whereas data words of dimension one correspond to data words in the sense introduced in [Bouyer 2002]. Finite runs of Minsky machines (with two counters) can be viewed as data words of dimension two over the alphabet $[1, \alpha]$ assuming that the Minsky machine has α distinct instructions (see also Section 4.1).

Let $\mathfrak{dw} = (\mathsf{a}^1, \mathfrak{d}_1^1, \ldots, \mathfrak{d}_\beta^1) \cdots (\mathsf{a}^L, \mathfrak{d}_1^L, \ldots, \mathfrak{d}_\beta^L)$ be a data word in $([1, \alpha] \times \mathbb{N}^{\beta})^+$, i.e. $\mathfrak{dw}$ is of dimension β and its underlying alphabet has cardinal $\alpha \geq 1$. The data word $\mathfrak{dw}$ shall be encoded by the heap $\mathfrak{h}_{\mathfrak{dw}}$ containing a path of the form below:

$$\mathfrak{l}_0^1 \rightarrow \mathfrak{l}_1^1 \rightarrow \cdots \rightarrow \mathfrak{l}_\beta^1 \rightarrow \cdots \rightarrow \mathfrak{l}_0^L \rightarrow \mathfrak{l}_1^L \rightarrow \cdots \rightarrow \mathfrak{l}_\beta^L$$

where

— for every $i \in [1, L]$, $\mathfrak{l}_0^i$ has $\mathsf{a}^i + 2$ predecessors,
— for all $i \in [1, L]$ and all $j \in [1, \beta]$, $\mathfrak{l}_j^i$ has $\mathfrak{d}_j^i + \alpha + 3$ predecessors,
— every location in the domain of the heap is either on that path or points to a location on that path.

Such a path from $\mathfrak{l}_0^1$ to $\mathfrak{l}_\beta^L$ is called the *main path*, and $\mathfrak{h}_{\mathfrak{dw}}^{(\beta+1)L-1}(\mathfrak{l}_0^1) = \mathfrak{l}_\beta^L$. Other simple encodings are possible (for instance without shifting the values from the finite alphabet or from the infinite domain) but the current one is well-suited for all the developments made in this paper. In particular, the encoding allows us to know easily whether a location encodes a letter from the finite alphabet or an element from the infinite domain. Note also that $\mathfrak{h}_{\mathfrak{dw}}$ is not uniquely specified, and we understand it modulo isomorphism as discussed in Remark 2.1.

Figure 3 presents the encoding of the data word $\mathfrak{dw}_0 = (2,1)(1,2)(2,2)$ of dimension 1 with $\alpha = 2$ with its representation of the heap $\mathfrak{h}_{\mathfrak{dw}}$ in which the predecessors of the locations on the main path are provided schematically. We use this type of schema in Section 4 to illustrate a few constructions.

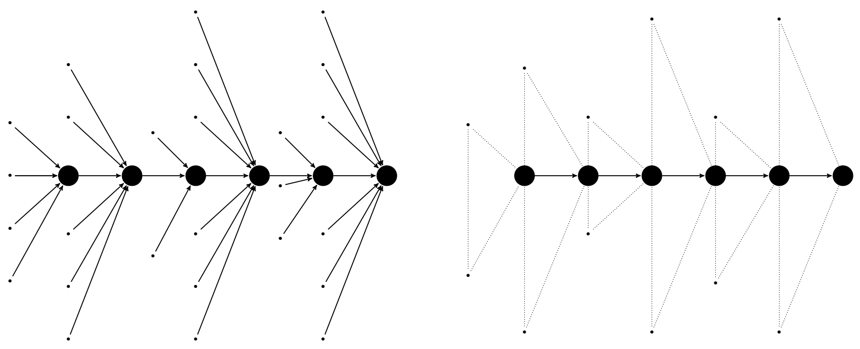

Fig. 3. The heap for data word $\mathfrak{dw}_0 = (2,1)(1,2)(2,2)$ of dimension 1 and with $\alpha = 2$ (focus on the main path) with its schematic representation.

The heap $\mathfrak{h}_{\mathfrak{dw}}$ looks like a fishbone. Let us make this precise. A heap $\mathfrak{h}$ is a *fishbone* $\overset{\text{def}}{\Leftrightarrow}$

(fb1). $\mathrm{dom}(\mathfrak{h}) \neq \emptyset$,
(fb2). there is a location reachable from all the locations of $\mathrm{dom}(\mathfrak{h})$ that is not in $\mathrm{dom}(\mathfrak{h})$, and
(fb3). there are no distinct locations $\mathfrak{l}_1, \mathfrak{l}_2, \mathfrak{l}_3, \mathfrak{l}_4, \mathfrak{l}_5$ such that $\mathfrak{l}_1 \rightarrow \mathfrak{l}_2 \rightarrow \mathfrak{l}_3 \leftarrow \mathfrak{l}_4 \leftarrow \mathfrak{l}_5$ in the heap $\mathfrak{h}$.

When $\mathfrak{h}$ is a fishbone, it has a tree-like structure (when looking at the edges backward), equipped with a root (the unique location from (fb2)), but additionally, one can recognize the locations on the main path as those locations with at least one predecessor. The existence of such a main path is guaranteed by (fb3). The first location on the main path satisfies the formula

$$\mathtt{first(u)} \stackrel{\text{def}}{=} (\sharp \mathtt{u} \geq 1) \land \neg(\sharp \mathtt{u}^{-1} \geq 1)$$

and the last location on the main path satisfies precisely the formula

$$\mathtt{last(u)} \stackrel{\text{def}}{=} (\sharp \mathtt{u} \geq 1) \land \neg \mathtt{alloc(u)}$$

Let ϕ_{fb} be the formula below:

$$\overbrace{\neg\mathtt{emp}}^{\text{(fb1)}} \land \overbrace{(\exists\, \mathtt{u}\ \neg\mathtt{alloc(u)} \land (\forall\, \overline{\mathtt{u}}\ \mathtt{alloc}(\overline{\mathtt{u}}) \Rightarrow \mathtt{ls}(\overline{\mathtt{u}},\mathtt{u})))}^{\text{(fb2)}} \land \overbrace{\neg(\exists\, \mathtt{u}\ (\sharp \mathtt{u}^{-2} \geq 0) * (\sharp \mathtt{u}^{-2} \geq 0))}^{\text{(fb3)}}$$

LEMMA 2.5. *Let $\mathfrak{h}$ be a heap. $\mathfrak{h} \models \phi_{\mathrm{fb}}$ iff $\mathfrak{h}$ is a fishbone.*

The proof for Lemma 2.5 is by an easy verification.

Now, let us refine the notion of a fishbone heap so that it takes into account constraints on numbers of predecessors. An (α, β)-*fishbone* is a fishbone heap such that

(C1). the first location on the main path has a number of predecessors in $[3, \alpha + 2]$,
(C2). on the main path, a location with a number of predecessors in $[3, \alpha + 2]$, is followed by β locations with at least $\alpha + 3$ predecessors, and
(C3). the number of locations on the main path is a multiple of $\beta + 1$.

It is easy to check that the formulae ϕ_{C1}, ϕ_{C2} and ϕ_{C3} in 1SL2($*$) defined below are able to express the conditions (C1), (C2) and (C3), respectively. This assumes that the heap is already known to be a fishbone, which is equivalent to the satisfaction of ϕ_{fb} (by Lemma 2.5).

$$\phi_{(C1)} \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ \mathtt{first(u)} \land (3 \leq \sharp \mathtt{u} \leq \alpha + 2)$$
$$\phi_{(C2)} \stackrel{\text{def}}{=} \forall\, \mathtt{u}\ (3 \leq \sharp \mathtt{u} \leq \alpha + 2) \Rightarrow \bigwedge_{i \in [1,\beta]} \sharp \mathtt{u}^{+i} \geq \alpha + 3$$
$$\phi_{(C3)} \stackrel{\text{def}}{=} \forall\, \mathtt{u}\ (3 \leq \sharp \mathtt{u} \leq \alpha + 2) \Rightarrow ((\neg \sharp \mathtt{u}^{+(\beta+1)} \geq 0) \lor (3 \leq \sharp \mathtt{u}^{+(\beta+1)} \leq \alpha + 2))$$

We write $\mathbf{dw}(\alpha, \beta)$ to denote the formula $\phi_{\mathrm{fb}} \land \phi_{(C1)} \land \phi_{(C2)} \land \phi_{(C3)}$. It specifies the shape of the encoding of data words in $([1, \alpha] \times \mathbb{N}^{\beta})^{+}$ as stated below.

LEMMA 2.6. *Let $\mathfrak{h}$ be a heap. We have $\mathfrak{h} \models \mathbf{dw}(\alpha, \beta)$ iff $\mathfrak{h}$ is an (α, β)-fishbone.*

Again, the proof is by an easy verification by using Lemma 2.5 and the correspondence between condition (Ci) and the formula $\phi_{(Ci)}$.

Given a data word $\mathfrak{dw} = (\mathtt{a}^1, \mathfrak{d}^1_1, \ldots, \mathfrak{d}^1_{\beta}) \cdots (\mathtt{a}^L, \mathfrak{d}^L_1, \ldots, \mathfrak{d}^L_{\beta})$, we can associate a (α, β)-fishbone $\mathfrak{h}_{\mathfrak{dw}}$ with $(1 + \beta) \times L$ locations on the main path, say

$$\mathfrak{l}^1_0 \rightarrow \mathfrak{l}^1_1 \rightarrow \cdots \rightarrow \mathfrak{l}^1_{\beta} \rightarrow \cdots \rightarrow \mathfrak{l}^L_0 \rightarrow \mathfrak{l}^L_1 \rightarrow \cdots \rightarrow \mathfrak{l}^L_{\beta}$$

such that

— for every $i \in [1, L]$, $\widetilde{\sharp \mathfrak{l}^i_0} = \mathtt{a}^i + 2$,
— for all $i \in [1, L]$ and all $j \in [1, \beta]$, $\widetilde{\sharp \mathfrak{l}^i_j} = \mathfrak{d}^i_j + \alpha + 3$.

The heap $\mathfrak{h}_{\mathfrak{dw}}$ is unique modulo isomorphism. This natural encoding generalizes the encoding of finite words by heaps in [Brochenin et al. 2012, Section 3] (see also [Bansal et al. 2009]) while providing a much more concise representation. Note also that the encoding by itself is of no use since it is essential to be able to operate on it with the logical language at hand.

Conversely, given a (α, β)-fishbone $\mathfrak{h}$ with $(1 + \beta) \times L$ locations on the main path, say

$$\mathfrak{l}^1_0 \to \mathfrak{l}^1_1 \to \cdots \to \mathfrak{l}^1_\beta \to \cdots \to \mathfrak{l}^L_0 \to \mathfrak{l}^L_1 \to \cdots \to \mathfrak{l}^L_\beta$$

we associate a (unique) data word $\mathfrak{dw}_{\mathfrak{h}} = (\mathtt{a}^1, \mathfrak{d}^1_1, \ldots, \mathfrak{d}^1_\beta) \cdots (\mathtt{a}^L, \mathfrak{d}^L_1, \ldots, \mathfrak{d}^L_\beta)$ such that for every $i \in [1, L]$, $\mathtt{a}^i \stackrel{\text{def}}{=} \widetilde{\sharp \mathfrak{l}^i_0} - 2$ and for all $i \in [1, L]$ and all $j \in [1, \beta]$, $\mathfrak{d}^i_j \stackrel{\text{def}}{=} \widetilde{\sharp \mathfrak{l}^i_j} - \alpha - 3$.

LEMMA 2.7. *There is a one-to-one map between data words in* $([1, \alpha] \times \mathbb{N}^\beta)^+$ *and* (α, β)*-fishbone heaps (modulo isomorphism).*

The proof is then by an easy verification.

So, we have seen that finite words can be encoded in 1SL2($*$), which allows us to establish that 1SL2($*$) is NEXPTIME-hard since first-order logic restricted to two quantified variables on finite words (written $\mathrm{FO2}_{\alpha,0}(<, +1, =)$ herein) is NEXPTIME-complete [Etessami et al. 1997]. Indeed, consider a sentence ϕ in that fragment of first-order logic. Let us define $t(\phi)$ such that ϕ is satisfiable iff $\mathbf{dw}(\alpha, 0) \wedge t(\phi)$ is satisfiable in 1SL2($*$).

We define the logarithmic-space translation t as follows ($i, j \in \{1, 2\}$).

— t is homomorphic for Boolean connectives,
— $t(\mathtt{u}_i = \mathtt{u}_j) \stackrel{\text{def}}{=} \mathtt{u}_i = \mathtt{u}_j$,
— $t(\mathtt{a}(\mathtt{u}_i)) \stackrel{\text{def}}{=} (\sharp \mathtt{u}_i = \mathtt{a} + 2)$,
— $t(\mathtt{u}_i = 1 + (\mathtt{u}_j)) \stackrel{\text{def}}{=} \mathtt{u}_j \hookrightarrow \mathtt{u}_i$,
— $t(\mathtt{u}_i < \mathtt{u}_j) \stackrel{\text{def}}{=} \mathtt{ls}(\mathtt{u}_i, \mathtt{u}_j) \wedge \mathtt{u}_i \neq \mathtt{u}_j$,
— $t(\exists\ \mathtt{u}_i\ \phi) \stackrel{\text{def}}{=} \exists\ \mathtt{u}_i\ (\sharp \mathtt{u}_i \geq 1) \wedge t(\phi)$.

Note that $\mathrm{FO2}_{\alpha,0}(<, +1, =)$ and 1SL2($*$) share the same number of quantified variables and $\mathtt{ls}(\mathtt{u}_i, \mathtt{u}_j)$ can be expressed in 1SL2($*$) (see Section 2.2). We do not provide the correctness proof herein since we can do much better than NEXPTIME-hardness by making a strong connection with Moszkowski's Interval Temporal Logic ITL (with the locality condition), see Section 3. However, we shall use a similar type of reduction in Section 4.6 with $\beta > 0$.

2.4. A modal logic for heaps

Let us conclude this section about logics for heaps, by introducing a new modal logic. We introduce a modal logic that is closely related to 1SL2. Modal Logic for Heaps (MLH) is a multimodal logic in which models are exactly heap graphs and it does not contain propositional variables (as 1SL does not contain unary predicate symbols). In a sense, it is similar to Hennessy-Milner logic HML [Hennessy and Milner 1980] in which the only atomic formulae are truth constants. However, the language contains modal operators and separating connectives, which is a feature shared with the logics defined in [Courtault and Galmiche 2013]. We define below the formulae of the modal logic MLH.

$$\phi \quad ::= \quad \bot \mid \neg\,\phi \mid \phi \wedge \phi \mid \Diamond\,\phi \mid \Diamond^{-1}\,\phi \mid \langle\neq\rangle\,\phi \mid \langle\star\rangle\,\phi \mid \phi \ast \phi \mid \phi \mathrel{-\!\!\ast} \phi$$

Note that there are no quantified variables involved in formulae, which is a feature shared with most known propositional modal logics, see e.g. [Blackburn et al. 2001]. We write MLH($\ast$) to denote the fragment of MLH without the magic wand operator $\mathrel{-\!\!\ast}$.

A *model for MLH* $\mathfrak{M}$ is a pair $(\mathbb{N}, \mathfrak{R})$ such that $\mathfrak{R}$ is a binary relation on $\mathbb{N}$ that is finite and functional. Otherwise said, the models for MLH are heap graphs. Models for MLH could be defined as heaps but we prefer to stick to the most usual presentation for modal logics with frames. The satisfaction relation $\models$ is defined below and it provides a standard semantics for the modal operators and separating connectives (we omit the clauses for Boolean connectives):

— never $\mathfrak{M}, \mathfrak{l} \models \bot$,
— $\mathfrak{M}, \mathfrak{l} \models \Diamond\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{R}$ and $\mathfrak{M}, \mathfrak{l}' \models \phi$,
— $\mathfrak{M}, \mathfrak{l} \models \Diamond^{-1}\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}', \mathfrak{l}) \in \mathfrak{R}$ and $\mathfrak{M}, \mathfrak{l}' \models \phi$,
— $\mathfrak{M}, \mathfrak{l} \models \langle\star\rangle\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{R}^*$ and $\mathfrak{M}, \mathfrak{l}' \models \phi$ where $\mathfrak{R}^*$ is the reflexive and transitive closure of $\mathfrak{R}$,
— $\mathfrak{M}, \mathfrak{l} \models \langle\neq\rangle\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}' \neq \mathfrak{l}$ such that $\mathfrak{M}, \mathfrak{l}' \models \phi$,
— $\mathfrak{M}, \mathfrak{l} \models \phi_1 \ast \phi_2 \overset{\text{def}}{\Leftrightarrow} (\mathbb{N}, \mathfrak{R}_1), \mathfrak{l} \models \phi_1$ and $(\mathbb{N}, \mathfrak{R}_2), \mathfrak{l} \models \phi_2$ for some partition $\{\mathfrak{R}_1, \mathfrak{R}_2\}$ of $\mathfrak{R}$,
— $\mathfrak{M}, \mathfrak{l} \models \phi_1 \mathrel{-\!\!\ast} \phi_2 \overset{\text{def}}{\Leftrightarrow}$ for all models $\mathfrak{M}' = (\mathbb{N}, \mathfrak{R}')$ such that $\mathfrak{R} \cap \mathfrak{R}' = \emptyset$ and $\mathfrak{R} \cup \mathfrak{R}'$ is functional, $\mathfrak{M}', \mathfrak{l} \models \phi_1$ implies $(\mathbb{N}, \mathfrak{R} \cup \mathfrak{R}'), \mathfrak{l} \models \phi_2$.

We use the following standard abbreviations:

— $\langle U\rangle\phi \overset{\text{def}}{=} \phi \vee \langle\neq\rangle\ \phi$, $[U]\phi \overset{\text{def}}{=} \neg\langle U\rangle\neg\phi$,
— $[\neq]\phi \overset{\text{def}}{=} \neg\langle\neq\rangle\neg\phi$,
— $\Diamond^{-1}_{\geq k}\top \overset{\text{def}}{=} \Diamond^{-1}\top \ast \cdots \ast \Diamond^{-1}\top$ ($k \geq 1$ times), $\Diamond^{-1}_{\leq k-1}\top \overset{\text{def}}{=} \neg\Diamond^{-1}_{\geq k}\top$,
— $\Diamond^{-1}_{[k_1,k_2]}\top \overset{\text{def}}{=} \Diamond^{-1}_{\geq k_1}\top \wedge \Diamond^{-1}_{\leq k_2}\top$, $\Diamond^{-1}_{=k}\top \overset{\text{def}}{=} \Diamond^{-1}_{\geq k}\top \wedge \Diamond^{-1}_{\leq k}\top$.

A formula ϕ is satisfiable whenever there is a model $\mathfrak{M}$ and a location $\mathfrak{l}$ such that $\mathfrak{M}, \mathfrak{l} \models \phi$. The satisfiability problem for MLH is therefore defined as any such problem for modal logics. Note that MLH has forward and backward modalities as in Prior's tense logic (see e.g. [Prior 1967]), the inequality modal operator (see e.g. [de Rijke 1992]) and the transitive closure operator as in PDL (see e.g. [Harel et al. 2000]). The most non-standard feature of MLH is certainly the presence of the separating connectives. It is possible to design a relational translation from MLH formulae into 1SL2 formulae by recycling variables (only $\mathtt{u}_1$ and $\mathtt{u}_2$ are used, so $i \in \{1, 2\}$):

— t is homomorphic for the connectives $\neg$, $\wedge$, $\ast$ and $\mathrel{-\!\!\ast}$,
— $t(\bot, \mathtt{u}_i) \overset{\text{def}}{=} \bot$,
— $t(\Diamond\ \phi, \mathtt{u}_i) \overset{\text{def}}{=} \exists\ \mathtt{u}_{3-i}\ (\mathtt{u}_i \hookrightarrow \mathtt{u}_{3-i}) \wedge t(\phi, \mathtt{u}_{3-i})$,
— $t(\Diamond^{-1}\ \phi, \mathtt{u}_i) \overset{\text{def}}{=} \exists\ \mathtt{u}_{3-i}\ (\mathtt{u}_{3-i} \hookrightarrow \mathtt{u}_i) \wedge t(\phi, \mathtt{u}_{3-i})$,
— $t(\langle\neq\rangle\ \phi, \mathtt{u}_i) \overset{\text{def}}{=} \exists\ \mathtt{u}_{3-i}\ (\mathtt{u}_i \neq \mathtt{u}_{3-i}) \wedge t(\phi, \mathtt{u}_{3-i})$,
— $t(\langle\star\rangle\ \phi, \mathtt{u}_i) \overset{\text{def}}{=} \exists\ \mathtt{u}_{3-i}\ \mathtt{ls}(\mathtt{u}_i, \mathtt{u}_{3-i}) \wedge t(\phi, \mathtt{u}_{3-i})$.

PROPOSITION 2.8. *A formula ϕ in MLH is satisfiable iff $\exists\ \mathtt{u}_1\ t(\phi, \mathtt{u}_1)$ is satisfiable in 1SL2. Moreover, if ϕ is in MLH($\ast$), then $\exists\ \mathtt{u}_1\ t(\phi, \mathtt{u}_1)$ is in 1SL2($\ast$).*

PROOF. (sketch) The proof is obtained as an obvious adaptation of the proof for the relational translation from modal logic K into FO2, see e.g. [Morgan 1976; van Benthem 1976; Blackburn et al. 2001]. Indeed, the models $(\mathbb{N}, \mathfrak{R})$ for MLH are heap graphs and therefore formulae in 1SL2 can be equivalently interpreted on MLH models; for instance, we get $(\mathbb{N}, \mathfrak{R}) \models_f \mathtt{u}_1 \hookrightarrow \mathtt{u}_2$ iff $(f(\mathtt{u}_1), f(\mathtt{u}_2)) \in \mathfrak{R}$. Similarly, $(\mathbb{N}, \mathfrak{R}) \models_f \phi_1 \mathrel{-\!\!*} \phi_2$ iff for all MLH models $(\mathbb{N}, \mathfrak{R}')$ such that $(\mathbb{N}, \mathfrak{R} \cup \mathfrak{R}')$ is an MLH model too and $(\mathbb{N}, \mathfrak{R}') \models_f \phi_1$, we have $(\mathbb{N}, \mathfrak{R} \cup \mathfrak{R}') \models_f \phi_2$.

Note that $\mathtt{u}_j$ is the only free variable in $t(\phi, \mathtt{u}_j)$. The standard translation t is semantically faithful in the following sense: for all MLH models $(\mathbb{N}, \mathfrak{R})$, $\mathfrak{l} \in \mathbb{N}$ and formulae ϕ in MLH, we have $(\mathbb{N}, \mathfrak{R}), \mathfrak{l} \models \phi$ iff $(\mathbb{N}, \mathfrak{R}) \models_{[\mathtt{u}_1 \mapsto \mathfrak{l}]} t(\phi, \mathtt{u}_1)$. This is sufficient to establish Proposition 2.8.

We show that for all $i \in \{1, 2\}$, for all formulae ψ in MLH, for all MLH models $\mathfrak{M} = (\mathbb{N}, \mathfrak{R})$ and for $\mathfrak{l} \in \mathbb{N}$, we have $\mathfrak{M}, \mathfrak{l} \models \psi$ iff $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} t(\psi, \mathtt{u}_i)$. The proof is by structural induction. The base case for $\perp$ and the cases in the induction step for the Boolean connectives are straightforward. By way of example, let us provide the cases in the induction step for $\psi = \langle \star \rangle\ \psi'$ and for $\psi = \psi_1 * \psi_2$. The proof for the other cases is similar and quite standard.
Case $\psi = \langle \star \rangle\ \psi'$. The following are equivalent:

— $\mathfrak{M}, \mathfrak{l} \models \psi$,
— $\mathfrak{M}, \mathfrak{l}' \models \psi'$ for some $\mathfrak{l}' \in \mathfrak{R}^*(\mathfrak{l})$ (by definition of $\models$),
— $\mathfrak{M} \models_{[\mathtt{u}_{3-i} \mapsto \mathfrak{l}']} t(\psi', \mathtt{x}_{3-i})$ for some $\mathfrak{l}' \in \mathbb{N}$ such that $\mathfrak{l}' \in \mathfrak{R}^*(\mathfrak{l})$ (by the induction hypothesis),
— $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} \exists\ \mathtt{u}_{3-i}\ \mathtt{ls}(\mathtt{u}_i, \mathtt{u}_{3-i})\ \wedge\ t(\psi', \mathtt{x}_{3-i})$ (by definition of $\models$ in 1SL2 and by the fact that `ls` is the reachability predicate),
— $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} t(\psi, \mathtt{u}_i)$ (by definition of t).

Case $\psi = \psi_1 * \psi_2$. The following are equivalent:

— $\mathfrak{M}, \mathfrak{l} \models \psi$,
— $(\mathbb{N}, \mathfrak{R}_1), \mathfrak{l} \models \psi_1$ and $(\mathbb{N}, \mathfrak{R}_2), \mathfrak{l} \models \psi_2$ for some partition $\{\mathfrak{R}_1, \mathfrak{R}_2\}$ of $\mathfrak{R}$, (by definition of $\models$ in MLH),
— $(\mathbb{N}, \mathfrak{R}_1) \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} t(\psi_1, \mathtt{u}_i)$ and $(\mathbb{N}, \mathfrak{R}_2) \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} t(\psi_2, \mathtt{u}_i)$ for some partition $\{\mathfrak{R}_1, \mathfrak{R}_2\}$ of $\mathfrak{R}$, (by the induction hypothesis),
— $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} t(\psi_1, \mathtt{u}_i) * t(\psi_2, \mathtt{u}_i)$ (by definition of the satisfaction relation in 1SL2)
— $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} t(\psi, \mathtt{u}_i)$ (by definition of t).

□

Modal logic MLH can be viewed as a fragment of 1SL2. Any formula $\psi_1 * \psi_2$ [resp. $\psi_1 \mathrel{-\!\!*} \psi_2$] in $t(\phi, \mathtt{u}_1)$ has at most one free variable. A similar restriction can be found in monodic fragments for first-order temporal logics, see e.g. [Degtyarev et al. 2002].

Since MLH($*$) can be translated into 1SL2($*$) and 1SL($*$) is decidable [Brochenin et al. 2012, Corollary 3.3], we get decidability of MLH($*$) as a corollary.

COROLLARY 2.9. *The satisfiability problem for MLH($*$) is decidable.*

Note that to be more uniform, we could have added to the modal language the converse operators $\langle \neq \rangle^{-1}$ and $\langle \star \rangle^{-1}$. However, since the inequality relation is symmetric, $\langle \neq \rangle^{-1}\phi$ is logically equivalent to $\langle \neq \rangle\phi$. The above translation can be obviously extended with the modal operator $\langle \star \rangle^{-1}$ and therefore decidability holds also for this extension. However, we have introduced MLH mainly to establish non-elementarity of MLH($*$) (shown below), refining the result for 1SL2($*$). We did not include $\langle \star \rangle^{-1}$ because the proof of non-elementarity result does not require it. By contrast, we do not know whether the satisfiability problem for MLH is decidable. As far as we know, the characterization of the computational complexity of MLH without separating connectives is open too. This corresponds to a fragment of deterministic PDL with (restricted) graded modalities and inequality modality.

3. WHEN INTERVAL TEMPORAL LOGIC MEETS SEPARATION LOGIC

Interval-based temporal logics admit time intervals as first-class objects (instead of time points), and an early and classical study for reasoning about intervals can be found in [Allen 1983]. One of the most prominent interval-based logics is Propositional Interval Temporal Logic (PITL), introduced by Ben Moszkowski in [Moszkowski 1983] for the verification of hardware components. It contains the so-called 'chop' operation that consists of chopping an interval into two subintervals. This is of course reminiscent of separating conjunction in separation logic, and in this section we make a formal statement about this correspondence, in addition to deriving new complexity results. Before doing so, it is worth noting that even though most standard point-based temporal logics used in computer science are decidable (CTL, CTL*, ECTL*, etc.), undecidability is much more common in the realm of interval-based temporal logics. Below, we consider PITL in which propositional variables are interpreted under the *locality condition* and for which decidability is guaranteed but computational complexity is very high. This will allow us to derive similar bounds for 1SL2($*$).

Below, we recall the main definitions about PITL under the locality condition and we explain why formulae from PITL can be faithfully translated into formulae in 1SL2($*$), leading to insights about both formalisms and new complexity results. A similar analysis is presented for MLH($*$) making new bridges between modal logic, interval-based temporal logics and separation logic.

3.1. Logic PITL

Given $\alpha \geq 1$, we consider the finite alphabet $\Sigma = [1, \alpha]$ and we write PITL$_\Sigma$ to denote propositional interval temporal logic in which the models are non-empty finite words in Σ^+. We write PITL instead of PITL$_\Sigma$ when the finite alphabet Σ is clear from the context. Formulae for PITL$_\Sigma$ are defined according to the following abstract grammar:

$$\phi \quad ::= \quad \mathtt{a} \mid \mathtt{pt} \mid \neg\phi \mid \phi \wedge \phi \mid \phi\, \mathbf{C}\, \phi$$

with $\mathsf{a} \in \Sigma$. Even though elements of Σ are natural numbers (for the sake of technical convenience), we write a to denote such an arbitrary element to emphasize that a is a letter from a finite alphabet. Roughly speaking, a holds true at word $\mathfrak{w}$ when a is the first letter of $\mathfrak{w}$. Similarly, the atomic formula $\mathtt{pt}$ holds true at a word $\mathfrak{w}$ when the word $\mathfrak{w}$ is only a single letter. The connective $\mathbf{C}$ is the *chop* operator, which chops a word.

Formally, we have a nonempty word $\mathfrak{w} \in \Sigma^+$, its length $|\mathfrak{w}|$, extractions of the ith letter $\mathfrak{w}_i$ where $1 \leq i \leq |\mathfrak{w}|$, and extractions of nonempty subwords $\mathfrak{w}_{i..j} = \mathfrak{w}_i\mathfrak{w}_{i+1}..\mathfrak{w}_j$, where $1 \leq i \leq j \leq |\mathfrak{w}|$. We define a ternary relation chops on words:

$$\mathsf{chops} \overset{\text{def}}{=} \{(\mathfrak{w}_1, \mathfrak{w}_2, \mathfrak{w}_3) \mid \exists\, \mathsf{a}, \mathfrak{w}', \mathfrak{w}'' \text{ such that } \mathfrak{w}_1 = \mathfrak{w}'\mathsf{a}\mathfrak{w}'', \mathfrak{w}_2 = \mathfrak{w}'\mathsf{a}, \mathfrak{w}_3 = \mathsf{a}\mathfrak{w}''\}$$

Observe that when a word $\mathfrak{w}_1$ is chopped into two subwords $\mathfrak{w}_2$ and $\mathfrak{w}_3$, there is an overlap between the last letter of $\mathfrak{w}_2$ and the first letter of $\mathfrak{w}_3$. For instance, $(\mathsf{abb}, \mathsf{ab}, \mathsf{bb}) \in \mathsf{chops}$ but $(\mathsf{ab}, \mathsf{a}, \mathsf{b}) \notin \mathsf{chops}$.

Let us define the satisfaction relation $\models$ for PITL$_\Sigma$ between a word $\mathfrak{w} \in \Sigma^+$ and a formula ϕ:

— $\mathfrak{w} \models \mathsf{a} \overset{\text{def}}{\Leftrightarrow} \mathfrak{w}_1 = \mathsf{a}$ (here, $\mathfrak{w}_1$ denotes the first letter of $\mathfrak{w}$).
— $\mathfrak{w} \models \mathtt{pt} \overset{\text{def}}{\Leftrightarrow} |\mathfrak{w}| = 1$.
— $\mathfrak{w} \models \neg\phi \overset{\text{def}}{\Leftrightarrow} \mathfrak{w} \not\models \phi$.
— $\mathfrak{w} \models \phi \wedge \psi \overset{\text{def}}{\Leftrightarrow} \mathfrak{w} \models \phi$ and $\mathfrak{w} \models \psi$.

— $\mathfrak{w} \models \phi \, \mathbf{C} \, \psi \overset{\text{def}}{\Leftrightarrow}$ there exist words $\mathfrak{w}_1, \mathfrak{w}_2$ such that $\mathsf{chops}(\mathfrak{w}, \mathfrak{w}_1, \mathfrak{w}_2)$, $\mathfrak{w}_1 \models \phi$ and $\mathfrak{w}_2 \models \psi$.

The satisfiability problem for PITL_Σ consists in checking whether a PITL_Σ formula admits a model satisfying it. Note that the models are *nonempty, finite* words and the satisfaction of a letter on a word depends only on its first letter (the locality condition).

Two examples. Consider the alphabet Σ with two distinct letters $\mathtt{a}$ and $\mathtt{b}$ and the PITL_Σ formula below:

$$(\mathtt{b} \, \mathbf{C} \, \mathtt{a}) \, \mathbf{C} \, \neg \mathtt{pt}$$

This formula is satisfiable; many words satisfy this formula, for example the word "bab"—the top-level chop is satisfied since $\mathtt{ba} \models \mathtt{b} \, \mathbf{C} \, \mathtt{a}$ and $\mathtt{ab} \models \neg\mathtt{pt}$. This gives insight on how to specify a lower-bound on word length, by applying sufficiently many chops and $\neg\mathtt{pt}$ to force a particular (minimum) length. Of course, $\mathtt{b} \, \mathbf{C} \, \mathtt{a}$ also enforces a minimum word length (of 2), but constrains also the word content.

Consider another example:

$$\mathtt{pt} \wedge (\mathtt{a} \, \mathbf{C} \, \mathtt{b})$$

For this formula to be satisfiable, there must exist a word $\mathfrak{w}$ for which both $\mathfrak{w} \models \mathtt{pt}$ and $\mathfrak{w} \models \mathtt{a} \, \mathbf{C} \, \mathtt{b}$. This is impossible, as the first implies $|\mathfrak{w}| = 1$, and there is no way to chop a single-letter word into subwords that satisfy both $\mathtt{a}$ and $\mathtt{b}$; the formula is unsatisfiable.

THEOREM 3.1. *(see e.g. [Moszkowski 1983; 2004]) Given $\alpha \geq 1$ and $\Sigma = [1, \alpha]$, the satisfiability problem for PITL_Σ is decidable, but with $\alpha \geq 2$ is not elementary recursive.*

3.2. Correspondence between words and heaps

From now on, we use the data word representation of Section 2.3. From Lemma 2.7, we know there is a fishbone heap $\mathfrak{h}_{\mathfrak{w}}$ corresponding to each nonempty word $\mathfrak{w} \in \Sigma^+$. Let us define a relation $\sim$ that establishes this correspondence between words and their fishbone representations, adding also a correspondence between the empty word and the empty heap:

$$\sim \overset{\text{def}}{=} \left\{ (\mathfrak{w}, \mathfrak{h}_{\mathfrak{w}}) \mid \mathfrak{w} \in \Sigma^+ \right\} \cup \{(\epsilon, \emptyset)\}$$

Here, observe that:

(1) $\sim$ is a bijection between the set of finite words in Σ^* and the set of (equivalence classes of isomorphic) $(\alpha, 0)$-fishbone heaps augmented with the empty heap;
(2) so, every word $\mathfrak{w}$ is in dom($\sim$);
(3) so, every $(\alpha, 0)$-fishbone heap is in ran($\sim$);
(4) so, if $\mathfrak{w} \sim \mathfrak{h}$, $\mathfrak{h}$ is either empty or an $(\alpha, 0)$-fishbone heap; and
(5) if $\mathfrak{w} \sim \mathfrak{h}$, then $\mathfrak{w}$ is empty iff dom($\mathfrak{h}$) is empty.

In this section, we will only employ $(\alpha, 0)$-fishbone heaps, with $\alpha = \text{card}(\Sigma)$.

The correspondence between finite words in Σ^+ and $(\alpha, 0)$-fishbone heaps satisfies a nice property as far as splitting a word into two disjoint subwords is concerned (which is a slight variant of chopping). Before making a formal statement, let us introduce the following notion.

A *clean cut* of a $(\alpha, 0)$-fishbone heap $\mathfrak{h}$ is a pair of $(\alpha, 0)$-fishbone heaps $(\mathfrak{h}_1, \mathfrak{h}_2)$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, and for some words $\mathfrak{w}_1 \sim \mathfrak{h}_1$ and $\mathfrak{w}_2 \sim \mathfrak{h}_2$, we have $\mathfrak{w}_1\mathfrak{w}_2 \sim \mathfrak{h}$. That is, a clean cut is one that neatly cleaves a heap representation of a word into two subheaps in correspondence with two subwords. Figure 4 illustrates examples of a clean cut and a non-clean cut on a fishbone heap. Informally, a non-clean cut is one that either results in one subheap (or both) no longer satisfying the $(\alpha, 0)$-fishbone conditions, or that results in subheaps that don't preserve predecessor counts and thus don't represent subwords of the original.

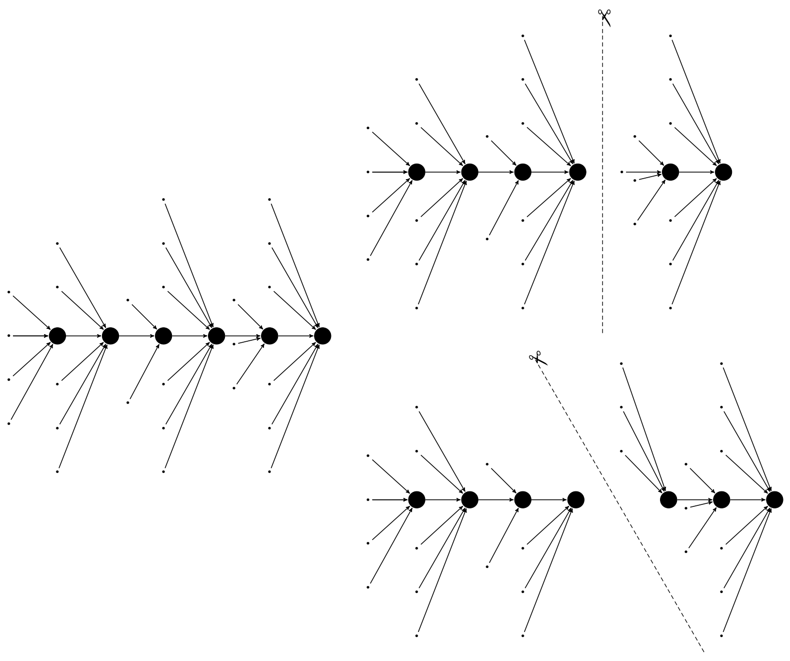

Fig. 4. A visual depiction of *clean* and *non-clean cuts*. Clockwise from left: the original $(\alpha, 0)$-fishbone heap; a clean cut of the original heap; a non-clean cut of the original heap. Note that clean cuts must result in two $(\alpha, 0)$-fishbone heaps. A non-clean cut may or may not do so; the figure depicts a non-clean cut that does result in two $(\alpha, 0)$-fishbone heaps.

LEMMA 3.2. *Let* $\mathfrak{w} \sim \mathfrak{h}$ *with* $\mathfrak{w} = \mathfrak{w}_1\mathfrak{w}_2 \in \Sigma^*$. *There exist heaps* $\mathfrak{h}_1$ *and* $\mathfrak{h}_2$ *such that* $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{w}_1 \sim \mathfrak{h}_1$, *and* $\mathfrak{w}_2 \sim \mathfrak{h}_2$.

PROOF. Suppose that $\mathfrak{w} \sim \mathfrak{h}$ and $\mathfrak{w} = \mathfrak{w}_1\mathfrak{w}_2 \in \Sigma^*$. If $\mathfrak{w}_1 = \varepsilon$ or $\mathfrak{w}_2 = \varepsilon$, then the proof is by an easy verification with $\mathfrak{h}$ equal to $\mathfrak{h}_1$ or $\mathfrak{h}_2$ respectively. In particular, if $\mathfrak{w} = \varepsilon$, then $\mathfrak{h}$ is the empty heap and therefore $\mathfrak{w}_1 = \mathfrak{w}_2 = \varepsilon$ and $\mathfrak{h}_1 = \mathfrak{h}_2 = \emptyset$, which satisfies the statement.

Otherwise suppose that $\mathfrak{w} = \mathtt{a}_1 \cdots \mathtt{a}_K \in \Sigma^+$, $\mathfrak{w}_1 = \mathtt{a}_1 \cdots \mathtt{a}_{K'} \in \Sigma^+$, $\mathfrak{w}_2 = \mathtt{a}_{K'+1} \cdots \mathtt{a}_K \in \Sigma^+$ $(K > K')$. Since $\mathfrak{w}$ is nonempty and $\mathfrak{w} \sim \mathfrak{h}$, $\mathfrak{h}$ is a fishbone heap and the main path of $\mathfrak{h}$ is of the form $\mathfrak{l}_1 \to \mathfrak{l}_2 \to \cdots \to \mathfrak{l}_K$ and for every $i \in [1, K]$, $\widetilde{\sharp\mathfrak{l}_i} = \mathtt{a}_i + 2$. Let $\mathfrak{h}_1$ be the subheap of $\mathfrak{h}$ whose domain is $\{\mathfrak{l}' \in \mathbb{N} : \mathfrak{l}' \text{ is an ancestor of } \mathfrak{l}_{K'} \text{ in } \mathfrak{h}\}$, and let $\mathfrak{h}_2$ be the unique heap such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$. It is easy to show that $\mathfrak{w}_1 \sim \mathfrak{h}_1$ and $\mathfrak{w}_2 \sim \mathfrak{h}_2$. Moreover, it is not difficult to see that $(\mathfrak{h}_1, \mathfrak{h}_2)$ is a clean cut of $\mathfrak{h}$. $\square$

Lemma 3.2 entails the following lemma, that will be useful to show the correctness of our reduction from PITL_Σ into 1SL2($*$). It is tailored to the semantics of the chop operator in PITL_Σ.

LEMMA 3.3. *For all letters* $\mathtt{a}, \mathtt{b} \in \Sigma$, *words* $\mathfrak{w} \in \Sigma^+$ *and* $\mathfrak{w}', \mathfrak{w}'' \in \Sigma^*$, *and heaps* $\mathfrak{h}$ *such that* $\mathfrak{w} \sim \mathfrak{h}$ *and* $\mathsf{chops}(\mathtt{a}\mathfrak{w}, \mathtt{a}\mathfrak{w}'\mathtt{b}, \mathtt{b}\mathfrak{w}'')$, *there exist heaps* $\mathfrak{h}_1, \mathfrak{h}_2$ *such that* $\mathfrak{w}'\mathtt{b} \sim \mathfrak{h}_1$, $\mathfrak{w}'' \sim \mathfrak{h}_2$, *and* $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$.

3.3. A reduction and its three ways to chop

In this section, we present a satisfiability-preserving translation of PITL_Σ into 1SL2($*$). This translation hinges on the insight that the chop operation is very similar to the separating conjunction in separation logic. However, the correspondence is not an exact one: the connective **C** of PITL_Σ does not cut into disjoint pieces, but rather preserves one letter on both sides, in a sense "duplicating" the letter upon which the chop operates.

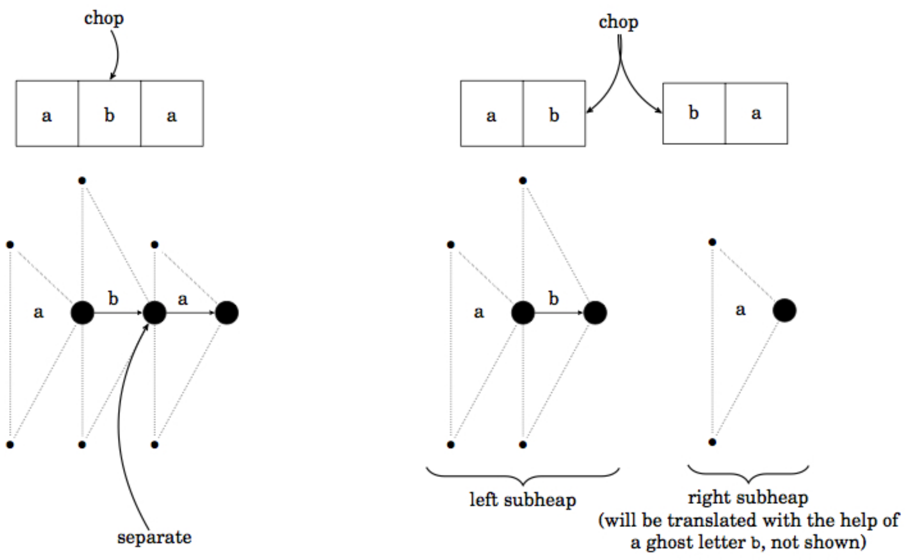
chop
a
b
a
b
a
separate
chop
a
b
b
a
a
b
a
left subheap
right subheap
(will be translated with the help of
a ghost letter b, not shown)

Fig. 5. The correspondence between PITL's chop '**C**' and separation logic's separating conjunction '$*$' (before and after).

To handle this discrepancy, our translation uses the standard separating conjunction on heaps, but internally carries a "ghost letter" (a parameter to the translation) on one side to represent this "lost" letter. In the translation, we denote this ghost letter parameter $\mathtt{a} \in \Sigma$. Figure 5 illustrates how a chop operation on words is translated into a separation on heaps. It is worth noting that we must always obtain a clean cut from the original heap.

Before presenting the formal definition of the translation, let us present a formula that allows us to perform a clean cut for which one of the subheaps contains all the ancestors of $f(\mathtt{u})$. Such a formula will be used in the translation and this is the purpose of Lemma 3.4.

LEMMA 3.4. *Given a fishbone heap $\mathfrak{h}$ and a word $\mathfrak{w}$ such that $\mathfrak{w} \sim \mathfrak{h}$, and an assignment f such that $f(\mathtt{u})$ is a location on the main path of $\mathfrak{h}$ with $\mathfrak{h} \models_f \mathtt{alloc}(\mathtt{u})$, any pair of heaps $(\mathfrak{h}_1, \mathfrak{h}_2)$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{h}_1 \models_f \mathbf{dw}(\alpha, 0) \wedge \neg\mathtt{alloc}(\mathtt{u})$, and $\mathfrak{h}_2 \models_f \mathbf{dw}(\alpha, 0) \wedge \sharp\mathtt{u} = 0$, is a clean cut of $\mathfrak{h}$.*

PROOF. Since $\mathfrak{h}_1 \models_f \mathbf{dw}(\alpha, 0)$ and $\mathfrak{h}_2 \models_f \mathbf{dw}(\alpha, 0)$, we know the heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are $(\alpha, 0)$-fishbones. Fishbones are single components, so we know that $\mathfrak{h}$ must be separated into exactly two connected components. It remains to analyze precisely how $\mathfrak{h}$ can be separated into two fishbones, and to show that it must be a clean cut.

We know $\mathfrak{l} = f(\mathtt{u})$ is on the main path of $\mathfrak{h}$, so that means $\mathfrak{h} \models_f \sharp\mathtt{u} > 0$. Since $\mathfrak{h}_2 \models_f \sharp\mathtt{u} = 0$, that means $\mathfrak{l}$ must have the same number of predecessors in $\mathfrak{h}$ as it does in $\mathfrak{h}_1$. We know $\mathfrak{h}_1 \models_f \neg\mathtt{alloc}(\mathtt{u})$, and we know $\mathfrak{l}$ has at least one predecessor in $\mathfrak{h}_1$. Therefore, $\mathfrak{l}$ is on the main path of $\mathfrak{h}_1$. We know $\mathfrak{h}_2 \models_f \sharp\mathtt{u} = 0$, so $\mathfrak{l}$ is not on the main path of $\mathfrak{h}_2$. However, it is allocated (since $\mathfrak{h} \models_f \mathtt{alloc}(\mathtt{u})$ and $\mathfrak{h}_1 \models_f \neg\mathtt{alloc}(\mathtt{u})$), so its successor (call the location $\mathfrak{l}'$) is on the main path of $\mathfrak{h}_2$. Let $f' = f[\bar{\mathtt{u}} \mapsto \mathfrak{l}']$. Now, note that $\mathfrak{h} \models_{f'} \mathtt{u} \hookrightarrow \bar{\mathtt{u}}$ and $\mathfrak{h}_1 \not\models_{f'} \mathtt{u} \hookrightarrow \bar{\mathtt{u}}$, and recall that that on a fishbone, no two predecessors of an element can both have predecessors (fb3). Therefore, $\mathfrak{l}'$ must have the same number of predecessors in $\mathfrak{h}_2$ as it did in $\mathfrak{h}$, and none of these predecessors can be on the main path.

Thus $\mathfrak{l}$ is the final location on the main path of $\mathfrak{h}_1$, and $\mathfrak{l}'$ is the first location on the main path of $\mathfrak{h}_2$. Further, $\mathfrak{l}$ has 0 predecessors in $\mathfrak{h}_2$ and the same number of predecessors in $\mathfrak{h}$ and $\mathfrak{h}_1$. $\mathfrak{l}'$ has 0 predecessors in $\mathfrak{h}_1$ and the same number of predecessors in $\mathfrak{h}$ and $\mathfrak{h}_2$.

Putting the above together, since $\mathfrak{l}$ and $\mathfrak{l}'$ are positions on the main path of $\mathfrak{h}$ such that $\mathfrak{h}(\mathfrak{l}) = \mathfrak{l}'$, and since $\mathfrak{l} \notin \mathrm{dom}(\mathfrak{h}_1)$, $\mathfrak{l} \in \mathrm{dom}(\mathfrak{h}_2)$, $\mathfrak{l} \in \mathrm{ran}(\mathfrak{h}_1)$, $\mathfrak{l} \notin \mathrm{ran}(\mathfrak{h}_2)$, we must have a clean cut. $\square$

We reduce a PITL$_\Sigma$ formula ϕ to a 1SL2($*$) formula $t(\phi)$ with the help of the main translation $t(\cdot)$. We use the auxiliary translation map $t_{\mathsf{a}}(\cdot)$ parameterized by a ghost letter a. The three disjuncts in the translation of $\phi\, \mathbf{C}\, \psi$ correspond to three types of chopping of $\mathfrak{w}$ that leads to three ways of separating the heap $\mathfrak{h}$ (assuming that $\mathfrak{w} \sim \mathfrak{h}$):

(1) When $(\mathfrak{w}, \mathsf{a}\mathfrak{w}_1\mathsf{b}, \mathsf{b}\mathfrak{w}_2) \in \mathsf{chops}$ and the ghost letter is a, the heap $\mathfrak{h}$ is separated into the heap $\mathfrak{h}_1$ with $\mathfrak{w}_1\mathsf{b} \sim \mathfrak{h}_1$ (with ghost letter a) and into the heap $\mathfrak{h}_2$ with $\mathfrak{w}_2 \sim \mathfrak{h}_2$ (with ghost letter b).
(2) When $(\mathfrak{w}, \mathfrak{w}, \mathsf{b}) \in \mathsf{chops}$ and the ghost letter is a, the heap $\mathfrak{h}$ is separated into itself (again with ghost letter a) and into the empty heap (with ghost letter b).
(3) When $(\mathfrak{w}, \mathsf{a}, \mathfrak{w}) \in \mathsf{chops}$ and the ghost letter is a, the heap $\mathfrak{h}$ is separated into the empty heap (with ghost letter a) and into itself (again with ghost letter a).

These are the three possible cases and the rest of the translation is quite straightforward.

$$t(\phi) \stackrel{\text{def}}{=} \big(\mathbf{dw}(\alpha,0) \vee \mathtt{emp}\big) \wedge \bigvee_{\mathtt{a}\in\Sigma} t_{\mathtt{a}}(\phi)$$

$$\begin{aligned}
t_{\mathtt{a}}(\mathtt{b}) &\stackrel{\text{def}}{=} \begin{cases} \top & \text{if } \mathtt{b} = \mathtt{a} \\ \bot & \text{if } \mathtt{b} \neq \mathtt{a} \end{cases} \\
t_{\mathtt{a}}(\mathtt{pt}) &\stackrel{\text{def}}{=} \mathtt{emp} \\
t_{\mathtt{a}}(\neg\phi) &\stackrel{\text{def}}{=} \neg t_{\mathtt{a}}(\phi) \\
t_{\mathtt{a}}(\phi \wedge \psi) &\stackrel{\text{def}}{=} t_{\mathtt{a}}(\phi) \wedge t_{\mathtt{a}}(\psi) \\
t_{\mathtt{a}}(\phi \,\mathbf{C}\, \psi) &\stackrel{\text{def}}{=} \mathsf{chop1}_{\mathtt{a}} \vee \mathsf{chop2}_{\mathtt{a}} \vee \mathsf{chop3}_{\mathtt{a}} \qquad \text{(with the three formulae as defined below)}
\end{aligned}$$

$$\begin{aligned}
\mathsf{chop1}_{\mathtt{a}} &\stackrel{\text{def}}{=} \bigvee_{\mathtt{b}\in\Sigma} \exists\, \mathtt{u} \Big(\sharp\mathtt{u} = \mathtt{b} + 2 \wedge \\
&\qquad\qquad \Big[\mathbf{dw}(\alpha,0) \wedge \neg\mathtt{alloc}(\mathtt{u}) \wedge t_{\mathtt{a}}(\phi) \ * \ \mathbf{dw}(\alpha,0) \wedge \sharp\mathtt{u} = 0 \wedge t_{\mathtt{b}}(\psi)\Big]\Big) \\
\mathsf{chop2}_{\mathtt{a}} &\stackrel{\text{def}}{=} \bigvee_{\mathtt{b}\in\Sigma} (\exists\, \mathtt{u}\ \mathtt{last}(\mathtt{u}) \wedge \sharp\mathtt{u} = \mathtt{b} + 2) \wedge \Big[t_{\mathtt{a}}(\phi) \ * \ \mathtt{emp} \wedge t_{\mathtt{b}}(\psi)\Big] \\
\mathsf{chop3}_{\mathtt{a}} &\stackrel{\text{def}}{=} \mathtt{emp} \wedge t_{\mathtt{a}}(\phi) \ * \ t_{\mathtt{a}}(\psi)
\end{aligned}$$

In full generality, $t_{\mathtt{a}}(\cdot)$ is also parameterized by the alphabet Σ (see the clause for formulae with outermost chop operator $\mathbf{C}$) and the formulae $\mathsf{chop1}_{\mathtt{a}}$, $\mathsf{chop2}_{\mathtt{a}}$, and $\mathsf{chop3}_{\mathtt{a}}$ are parameterized by $\phi \,\mathbf{C}\, \psi$. Clearly the translation $t(\cdot)$ can only produce 1SL2($*$) formulae, as the right-hand side of each translation step above is in 1SL2($*$). Note also that $t_{\mathtt{a}}(\phi)$ always produces a closed formula (i.e., without free occurrences of individual variables).

The correctness of the translation is stated below, making completely explicit the role of the ghost letter in the translation process.

LEMMA 3.5. *Let* $\mathtt{a} \in \Sigma$, $\mathfrak{w} \in \Sigma^*$, *and* $\mathfrak{h}$ *be a heap such that* $\mathfrak{w} \sim \mathfrak{h}$. *For every PITL*$_\Sigma$ *formula* ϕ, *we have* $\mathtt{a}\mathfrak{w} \models \phi$ *iff* $\mathfrak{h} \models t_{\mathtt{a}}(\phi)$.

Remark 3.6. Note that since $t_{\mathtt{a}}(\phi)$ has no free occurrences of individual variables, in Lemma 3.5, there is no need to specify what the assignments are.

PROOF. The proof is by structural induction.
The base cases are:

— $t_{\mathtt{a}}(\mathtt{b})$. ($\Rightarrow$) If $\mathtt{a}\mathfrak{w} \models \mathtt{b}$, then $\mathtt{a} = \mathtt{b}$. Clearly $\mathfrak{h} \models \top$, so $\mathfrak{h} \models t_{\mathtt{a}}(\mathtt{b})$.
($\Leftarrow$) $\mathfrak{h} \models t_{\mathtt{a}}(\mathtt{b})$, so it must be that $\mathtt{a} = \mathtt{b}$ (since $\mathfrak{h} \not\models \bot$). Thus $\mathtt{a}\mathfrak{w} \models \mathtt{b}$.
— $t_{\mathtt{a}}(\mathtt{pt})$. ($\Rightarrow$) If $\mathtt{a}\mathfrak{w} \models \mathtt{pt}$, then $\mathfrak{w} = \epsilon$. Thus $\mathfrak{h} = \emptyset$. Since $t_{\mathtt{a}}(\mathtt{pt}) = \mathtt{emp}$, we have $\mathfrak{h} \models t_{\mathtt{a}}(\mathtt{pt})$.
($\Leftarrow$) $\mathfrak{h} \models t_{\mathtt{a}}(\mathtt{pt})$, and since $t_{\mathtt{a}}(\mathtt{pt}) = \mathtt{emp}$, we know $\mathfrak{h} = \emptyset$. Thus $\mathfrak{w} = \epsilon$, so clearly $\mathtt{a}\mathfrak{w} \models \mathtt{pt}$.

The inductive cases are:

— $t_{\mathtt{a}}(\neg\phi)$. ($\Rightarrow$) $\mathtt{a}\mathfrak{w} \models \neg\phi$, so $\mathtt{a}\mathfrak{w} \not\models \phi$. By IH, $\mathfrak{h} \not\models t_{\mathtt{a}}(\phi)$, so $\mathfrak{h} \models \neg t_{\mathtt{a}}(\phi)$ and precisely $\neg t_{\mathtt{a}}(\phi) = t_{\mathtt{a}}(\neg\phi)$.
($\Leftarrow$) $\mathfrak{h} \models \neg t_{\mathtt{a}}(\phi)$, so $\mathfrak{h} \not\models t_{\mathtt{a}}(\phi)$. By IH, $\mathtt{a}\mathfrak{w} \not\models \phi$, so $\mathtt{a}\mathfrak{w} \models \neg\phi$.

— $t_{\mathsf{a}}(\phi \wedge \psi)$. ($\Rightarrow$) $\mathsf{a}\mathfrak{w} \models \phi \wedge \psi$, so $\mathsf{a}\mathfrak{w} \models \phi$ and $\mathsf{a}\mathfrak{w} \models \psi$. By IH, $\mathfrak{h} \models t_{\mathsf{a}}(\phi)$ and $\mathfrak{h} \models t_{\mathsf{a}}(\psi)$, so then $\mathfrak{h} \models t_{\mathsf{a}}(\phi) \wedge t_{\mathsf{a}}(\psi)$. We have $\mathfrak{h} \models t_{\mathsf{a}}(\phi \wedge \psi)$.
($\Leftarrow$) $\mathfrak{h} \models t_{\mathsf{a}}(\phi \wedge \psi)$, so $\mathfrak{h} \models t_{\mathsf{a}}(\phi) \wedge t_{\mathsf{a}}(\psi)$. Then $\mathfrak{h} \models t_{\mathsf{a}}(\phi)$ and $\mathfrak{h} \models t_{\mathsf{a}}(\psi)$. By IH, $\mathsf{a}\mathfrak{w} \models \phi$ and $\mathsf{a}\mathfrak{w} \models \psi$, so $\mathsf{a}\mathfrak{w} \models \phi \wedge \psi$.

— $t_{\mathsf{a}}(\phi\, \mathbf{C}\, \psi)$. ($\Rightarrow$) If $\mathsf{a}\mathfrak{w} \models \phi\, \mathbf{C}\, \psi$, then there are $\mathfrak{w}_1, \mathfrak{w}_2$ such that $\mathsf{chops}(\mathsf{a}\mathfrak{w}, \mathfrak{w}_1, \mathfrak{w}_2)$, $\mathfrak{w}_1 \models \phi$, and $\mathfrak{w}_2 \models \psi$. From the definition of chops, we have, further, that there are $\mathsf{b}, \mathfrak{w}', \mathfrak{w}''$ such that $\mathsf{a}\mathfrak{w} = \mathfrak{w}'\mathsf{b}\mathfrak{w}''$, $\mathfrak{w}_1 = \mathfrak{w}'\mathsf{b}$, and $\mathfrak{w}_2 = \mathsf{b}\mathfrak{w}''$.

Case 1. $\mathfrak{w}' = \epsilon$ (the chop occurs at the first position). Then $\mathsf{a}\mathfrak{w} = \mathsf{b}\mathfrak{w}''$ and $\mathfrak{w}_1 = \mathsf{b} = \mathsf{a}$, so $\mathfrak{w} = \mathfrak{w}''$. Since $\mathsf{a} \models \phi$ and $\mathsf{a}\mathfrak{w} \models \psi$, we have by IH that $\emptyset \models t_{\mathsf{a}}(\phi)$ and $\mathfrak{h} \models t_{\mathsf{a}}(\psi)$. Then $\mathfrak{h} \models \mathtt{emp} \wedge t_{\mathsf{a}}(\phi) \ * \ t_{\mathsf{a}}(\psi)$, so $\mathfrak{h} \models \mathsf{chop3}_{\mathsf{a}}$, and $\mathfrak{h} \models t_{\mathsf{a}}(\phi\, \mathbf{C}\, \psi)$.

Case 2. $\mathfrak{w}' \neq \epsilon$, $\mathfrak{w}'' = \epsilon$ (the chop occurs at the last position). $\mathsf{a}\mathfrak{w} = \mathfrak{w}_1$, so $\mathsf{a}\mathfrak{w} \models \phi$. $\mathfrak{w}_2 = \mathsf{b}$, so $\mathsf{b} \models \psi$. By IH, we then have $\mathfrak{h} \models t_{\mathsf{a}}(\phi)$ and $\emptyset \models t_{\mathsf{b}}(\psi)$. Thus $\mathfrak{h} \models t_{\mathsf{a}}(\phi) \ * \ \mathtt{emp} \wedge t_{\mathsf{b}}(\psi)$. Further, since the last letter of $\mathfrak{w}$ is b and $\mathfrak{w} \sim \mathfrak{h}$, the last location on the main path of $\mathfrak{h}$ must have $\mathsf{b} + 2$ predecessors. Therefore, $\mathfrak{h} \models \exists\, \mathtt{u}\ \mathtt{last(u)} \wedge \sharp\mathtt{u} = \mathsf{b} + 2$. So $\mathfrak{h} \models \mathsf{chop2}_{\mathsf{a}}$, and thus we have $\mathfrak{h} \models t_{\mathsf{a}}(\phi\, \mathbf{C}\, \psi)$.

Case 3. $\mathfrak{w}' \neq \epsilon$, $\mathfrak{w}'' \neq \epsilon$ (the chop occurs in the middle of the word). We know $\mathfrak{w}_1$ is nonempty and starts with a, so let $\mathfrak{w}_1'$ be a word such that $\mathfrak{w}_1 = \mathsf{a}\mathfrak{w}_1'$. From Lemma 3.3, we have heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 * \mathfrak{h}_2$, $\mathfrak{w}_1' \sim \mathfrak{h}_1$, and $\mathfrak{w}'' \sim \mathfrak{h}_2$. Observe that, since they are in $\mathrm{ran}(\sim)$, $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are $(\alpha, 0)$-fishbones. Since $\mathsf{a}\mathfrak{w}_1' \models \phi$ and $\mathsf{b}\mathfrak{w}'' \models \psi$, then by IH, we have $\mathfrak{h}_1 \models t_{\mathsf{a}}(\phi)$ and $\mathfrak{h}_2 \models t_{\mathsf{b}}(\psi)$. Now, let $\mathfrak{l}$ be the location of the chop, so that $\mathfrak{l}$ is the last position on the main path of $\mathfrak{h}_1$ and a predecessor of the first position on the main path of $\mathfrak{h}_2$. Let f be an assignment such that $f(\mathtt{u}) = \mathfrak{l}$. Since $\mathfrak{h}_1 \models_f \mathbf{dw}(\alpha, 0) \wedge \neg\mathtt{alloc(u)} \wedge t_{\mathsf{a}}(\phi)$ and $\mathfrak{h}_2 \models_f \mathbf{dw}(\alpha, 0) \wedge \sharp\mathtt{u} = 0 \wedge t_{\mathsf{b}}(\psi)$, we have $\mathfrak{h} \models_f \big[\mathbf{dw}(\alpha, 0) \wedge \neg\mathtt{alloc(u)} \wedge t_{\mathsf{a}}(\phi) \ * \ \mathbf{dw}(\alpha, 0) \wedge \sharp\mathtt{u} = 0 \wedge t_{\mathsf{b}}(\psi)\big]$. Further, since in $\mathfrak{h}$, location $\mathfrak{l}$ encodes the letter b, it has $\mathsf{b} + 2$ predecessors, so we have $\mathfrak{h} \models_f \sharp\mathtt{u} = \mathsf{b} + 2$, so $\mathfrak{h} \models \mathsf{chop1}_{\mathsf{a}}$. Then $\mathfrak{h} \models t_{\mathsf{a}}(\phi\, \mathbf{C}\, \psi)$.

($\Leftarrow$) If we have that $\mathfrak{h} \models t_{\mathsf{a}}(\phi\, \mathbf{C}\, \psi)$, then one of the disjuncts in the translation holds, depending on where the separation applies in the heap and the content at that position. We consider three cases based on which of the constraints ($\mathsf{chop1}_{\mathsf{a}}$, $\mathsf{chop2}_{\mathsf{a}}$, or $\mathsf{chop3}_{\mathsf{a}}$) applies.

Case 1. The $\mathsf{chop1}_{\mathsf{a}}$ constraint applies (neither subheap is empty). We know it holds for some $\mathsf{b} \in \Sigma$, and that the existentially quantified formulae holds for some location; w.l.o.g., assume that the case $\mathsf{b} \in \Sigma$ holds, with location $\mathfrak{l}$. Let f be an assignment such that $f(\mathtt{u}) = \mathfrak{l}$. We know that $\mathfrak{h} \models_f \sharp\mathtt{u} = \mathsf{b} + 2$, and that $\mathfrak{h}$ can be separated into two disjoint subheaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h}_1 \models_f \mathbf{dw}(\alpha, 0) \wedge \neg\mathtt{alloc(u)} \wedge t_{\mathsf{a}}(\phi)$ and $\mathfrak{h}_2 \models_f \mathbf{dw}(\alpha, 0) \wedge \sharp\mathtt{u} = 0 \wedge t_{\mathsf{b}}(\psi)$. This cut must be a *clean cut* by Lemma 3.4. By IH, then, we have words $\mathfrak{w}_1, \mathfrak{w}_2$ such that $\mathfrak{w}_1 \sim \mathfrak{h}_1$, $\mathfrak{w}_2 \sim \mathfrak{h}_2$, $\mathsf{a}\mathfrak{w}_1 \models \phi$, and $\mathsf{b}\mathfrak{w}_2 \models \psi$. We know $\mathfrak{w}_1$ ends with a letter b, since $\mathtt{u}$ is the last position on the main path of $\mathfrak{h}_1$ and has $\mathsf{b} + 2$ predecessors. So then $\mathsf{chops}(\mathsf{a}\mathfrak{w}_1\mathfrak{w}_2, \mathsf{a}\mathfrak{w}_1, \mathsf{b}\mathfrak{w}_2)$ holds. Thus $\mathsf{a}\mathfrak{w}_1\mathfrak{w}_2 \models \phi\, \mathbf{C}\, \psi$.

Case 2. The $\mathsf{chop2}_{\mathsf{a}}$ constraint applies (the right subheap is empty). We know it holds for some $\mathsf{b} \in \Sigma$, and that the existentially quantified formula holds for some location; w.l.o.g., then, assume that the case $\mathsf{b} \in \Sigma$ holds, at some location $\mathfrak{l}$. We know that heap $\mathfrak{h}$ can be separated into two disjoint subheaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h}_1 \models t_{\mathsf{a}}(\phi)$, $\mathfrak{h}_2 = \emptyset$, and $\mathfrak{h}_2 \models t_{\mathsf{b}}(\psi)$. Thus $\mathfrak{h} = \mathfrak{h}_1$, and by IH we have $\mathsf{a}\mathfrak{w} \models \phi$ and $\mathsf{b} \models \psi$. Next, let f be an assignment such that $f(\mathtt{u}) = \mathfrak{l}$. We know $\mathfrak{h} \models_f \mathtt{last(u)}$ and that $\mathfrak{h} \models_f \sharp\mathtt{u} = \mathsf{b} + 2$.

Since $\mathfrak{w} \sim \mathfrak{h}$, the last letter of $\mathfrak{w}$ is $\mathtt{b}$. So $\mathsf{chops}(\mathtt{a}\mathfrak{w}, \mathtt{a}\mathfrak{w}, \mathtt{b})$, and therefore $\mathtt{a}\mathfrak{w} \models \phi \, \mathbf{C} \, \psi$.

Case 3. The $\mathsf{chop3}_{\mathtt{a}}$ constraint applies (the left subheap is empty). We then have $\mathfrak{h}_1 = \emptyset$, $\emptyset \models t_{\mathtt{a}}(\phi)$, and $\mathfrak{h}_2 \models t_{\mathtt{a}}(\psi)$. Since $\mathfrak{h} = \mathfrak{h}_2$, by IH we have $\mathtt{a} \models \phi$ and $\mathtt{a}\mathfrak{w} \models \psi$. Then $\mathsf{chops}(\mathtt{a}\mathfrak{w}, \mathtt{a}, \mathtt{a}\mathfrak{w})$; thus $\mathtt{a}\mathfrak{w} \models \phi \, \mathbf{C} \, \psi$.

Therefore, $\mathtt{a}\mathfrak{w} \models \phi$ iff $\mathfrak{h} \models t_{\mathtt{a}}(\phi)$ □

As a result, we obtain a reduction between the satisfiability problems, as stated below.

LEMMA 3.7. *Given $\alpha \geq 1$ and $\Sigma = [1, \alpha]$, a PITL$_\Sigma$ formula ϕ is satisfiable if and only if the 1SL2($*$) formula $t(\phi)$ is satisfiable.*

PROOF. ($\Rightarrow$) Suppose that ϕ is satisfiable. This means that there exists a nonempty word $\mathfrak{w}$ such that $\mathfrak{w} \models \phi$. The word $\mathfrak{w}$ can be written in the form $\mathfrak{w} = \mathtt{a}\mathfrak{w}'$ for some letter $\mathtt{a}$. If $\mathfrak{w}' = \epsilon$, we have $\mathfrak{w}' \sim \emptyset$ and by Lemma 3.5, we have $\emptyset \models \mathtt{emp} \wedge t_{\mathtt{a}}(\phi)$. So $\emptyset \models t(\phi)$ and therefore $t(\phi)$ is satisfiable. If $\mathfrak{w}' \neq \epsilon$, then there is a $(\alpha, 0)$-fishbone heap $\mathfrak{h}'$ such that $\mathfrak{w}' \sim \mathfrak{h}'$. By Lemma 3.5, we have $\mathfrak{h}' \models \mathbf{dw}(\alpha, 0) \wedge t_{\mathtt{a}}(\phi)$. So $\mathfrak{h}' \models t(\phi)$ and therefore $t(\phi)$ is satisfiable.
($\Leftarrow$) If $t(\phi)$ is satisfiable, then there exists a heap $\mathfrak{h}$ such that $\mathfrak{h} \models (\mathbf{dw}(\alpha, 0) \vee \mathtt{emp}) \wedge \bigvee_{\mathtt{a} \in \Sigma} t_{\mathtt{a}}(\phi)$. If $\mathfrak{h} \models \mathtt{emp} \wedge t_{\mathtt{a}}(\phi)$ for some letter $\mathtt{a}$, then by Lemma 3.5, we have $\mathtt{a} \models \phi$. Otherwise, if $\mathfrak{h} \models \mathbf{dw}(\alpha, 0) \wedge t_{\mathtt{a}}(\phi)$, then $\mathfrak{h}$ is an $(\alpha, 0)$-fishbone heap by Lemma 2.6 and then there is a word $\mathfrak{w}$ such that $\mathfrak{w} \sim \mathfrak{h}$ such that by Lemma 3.5, we have $\mathtt{a}\mathfrak{w} \models \phi$. In both cases, ϕ is a satisfiable formula in PITL$_\Sigma$. □

THEOREM 3.8. *The satisfiability problem for 1SL2($*$) is decidable but not elementary recursive.*

PROOF. Satisfiability for PITL$_\Sigma$ is known to be decidable with non-elementary complexity when Σ has at least two elements, see e.g. [Moszkowski 1983; 2004], and 1SL($*$) is decidable [Brochenin et al. 2012]. From the correctness of our translation $t(\cdot)$ of PITL$_\Sigma$ to 1SL2($*$) (Lemma 3.7), we then conclude that 1SL2($*$) is decidable but not elementary recursive. Note that the map $t(\cdot)$ may require exponential time and space in the size of the input formula in the worst-case but this is still fine to establish that 1SL2($*$) is not elementary recursive, since this adds only a single exponential. □

As mentioned earlier, Theorem 3.8 refines the non-elementarity result for 1SL($*$) established in [Brochenin et al. 2012].

Remark 3.9. The reduction from PITL to 1SL2($*$) provided in this section allows us to underline the common features of both formalisms. However, non-elementarity of 1SL2($*$) can be established in a slightly different way as explained below. First, non-elementarity of PITL is due to Dexter Kozen (see e.g. [Moszkowski 2004, Appendix A.3])[2], and the proof is by reduction from the nonemptiness problem of regular expressions built over a binary alphabet with union, concatenation and complement [Stockmeyer 1974]. Nonelementarity of 1SL2($*$) can be obtained by defining a similar reduction, but this is of course less insightful to understand the relationships between interval temporal logic and separation logic. Alternatively, it is also possible to consider the variant of PITL in which the chop operator does not share a letter, since this variant is of identical expressive power and complexity. In that way, we may avoid the introduction of the ghost letter but at the cost of introducing empty models (which may occur when chopping has no sharing) and of using a less standard interval temporal

[2]We thank Ben Moszkowski for pointing us to this fact.

logic. So, the current reduction from PITL is quite an attractive option to relate the logics. Finally, as noted in [Moszkowski 2004, Appendix A], complexity results about PITL presented in [Moszkowski 1983] were obtained in collaboration with Joseph Halpern.

In Section 3.4 below, we establish an even stronger result (see Theorem 3.12). The proof uses the same principles as for the proof of Theorem 3.8 and we only need to express the properties in *modal lingua*.

3.4. A refinement with the modal fragment of 1SL2($*$)

In this section, we show that the satisfiability problem for MLH($*$) is decidable but it is not elementary recursive. Decidability is due to the fact that the standard translation leads to formulae in 1SL2($*$), see Section 2.4. In order to establish the lower bound, we express in MLH($*$) all the properties that were useful to translate PITL$_\Sigma$ formulae into 1SL2($*$). For instance, note that the empty heap is the only heap validating the formula $([U]\neg\Diamond\top)$. Similarly, a location with at least one predecessor and with no successor (for instance, last location on the main path in a fishbone heap) satisfies the formula $(\Diamond^{-1}\top \wedge \neg\Diamond\top)$.

More interestingly, the formula in 1SL2($*$) characterizing the (α,β)-fishbone heaps has a modal counterpart. Let us consider the following formulae.

— The formula $\phi^{\Box}_{\mathrm{fb}}$ defined below is designed exactly as the formula ϕ_{fb} (see Section 2.3).

$$(\langle U\rangle\Diamond\top)\wedge$$

$$\langle U\rangle((\Diamond^{-1}\top \wedge \neg\Diamond\top) \wedge [\neq]\neg(\Diamond^{-1}\top \wedge \neg\Diamond\top)) \wedge [U](\Diamond\top \Rightarrow \langle\star\rangle(\Diamond^{-1}\top \wedge \neg\Diamond\top))\wedge$$

$$(\neg\langle U\rangle(\Diamond^{-1}\Diamond^{-1}\top * \Diamond^{-1}\Diamond^{-1}\top))$$

This is a faithful translation except that we use the specification language MLH($*$).

— The formula $\phi^{\Box}_{(C1)}$ defined below is also designed exactly as the formula $\phi_{(C1)}$ (see Section 2.3).

$$\langle U\rangle((\Diamond^{-1}\top) \wedge (\neg\Diamond^{-1}\Diamond^{-1}\top) \wedge \Diamond^{-1}_{[3,\alpha+3]}\top)$$

— The formula $\phi^{\Box}_{(C2)}$ is equal to $[U](\Diamond^{-1}_{[3,\alpha+3]}\top \Rightarrow \bigwedge_{i\in[1,\beta]} \overbrace{\Diamond\cdots\Diamond}^{i \text{ times}} \Diamond^{-1}_{\geq\alpha+3}\top)$

— The formula $\phi^{\Box}_{(C3)}$ is defined below:

$$[U](\Diamond^{-1}_{[3,\alpha+3]}\top \Rightarrow (\neg \overbrace{\Diamond\cdots\Diamond}^{\beta+1 \text{ times}} \top) \vee \overbrace{\Diamond\cdots\Diamond}^{\beta+1 \text{ times}} (\Diamond^{-1}_{[3,\alpha+3]}\top))$$

We write $\mathbf{dw}^{\Box}(\alpha,\beta)$ to denote the formula $\phi^{\Box}_{\mathrm{fb}} \wedge \phi^{\Box}_{(C1)} \wedge \phi^{\Box}_{(C2)} \wedge \phi^{\Box}_{(C3)}$. It specifies the shape of the encoding of data words in $([1,\alpha]\times\mathbb{N}^{\beta})^{+}$ as stated below. Note that since $\mathbf{dw}^{\Box}(\alpha,\beta)$ is a Boolean combination of formulae whose outermost connectives are $[U]$ or $\langle U\rangle$, then $\mathbf{dw}^{\Box}(\alpha,\beta)$ holds true at some location iff $\mathbf{dw}^{\Box}(\alpha,\beta)$ holds true at any location.

LEMMA 3.10. *Let $\mathfrak{M} = (\mathbb{N},\mathfrak{R})$ be a model for MLH. $\mathfrak{M},\mathfrak{l} \models \mathbf{dw}^{\Box}(\alpha,\beta)$ for some location $\mathfrak{l}$ iff $\mathfrak{M}$ is the graph of an (α,β)-fishbone heap.*

Again, the proof is by an easy verification by using Lemma 2.5 and the correspondence between condition (Ci) and the formula $\phi^{\Box}_{(Ci)}$. In the rest of this section we are back to the case $\beta = 0$.

Given a formula ϕ in PITL$_\Sigma$ with $\Sigma = [1,\alpha]$, we define a modal formula $t^{\Box}(\phi)$ such that ϕ is satisfiable iff $t^{\Box}(\phi)$ is satisfiable. Actually, the modal formula $t^{\Box}(\phi)$ will express exactly the same properties as in the translation into 1SL2($*$). For instance, $t^{\Box}(\phi)$

is precisely the formula below:

$$(\mathbf{dw}^{\Box}(\alpha, 0) \vee ([U]\neg\Diamond\top)) \wedge (\bigvee_{\mathtt{a}\in\Sigma} t^{\Box}_{\mathtt{a}}(\phi))$$

The formula $t^{\Box}_{\mathtt{a}}(\phi)$ is defined inductively as follows.

— $t^{\Box}_{\mathtt{a}}(\mathtt{a}) \overset{\text{def}}{=} \top$ and $t^{\Box}_{\mathtt{a}}(\mathtt{b}) \overset{\text{def}}{=} \bot$ for every letter $\mathtt{b} \in \Sigma \setminus \{\mathtt{a}\}$.

— $t^{\Box}_{\mathtt{a}}(\cdot)$ is homomorphic for Boolean connectives.

— $t^{\Box}_{\mathtt{a}}(\mathtt{pt}) \overset{\text{def}}{=} ([U]\neg\Diamond\top)$.

— The formula $t^{\Box}_{\mathtt{a}}(\phi \,\mathbf{C}\, \psi)$ is defined as $\mathsf{chop1}^{\Box}_{\mathtt{a}} \vee \mathsf{chop2}^{\Box}_{\mathtt{a}} \vee \mathsf{chop3}^{\Box}_{\mathtt{a}}$ where:
  — $\mathsf{chop1}^{\Box}_{\mathtt{a}} \overset{\text{def}}{=} \bigvee_{\mathtt{b}\in\Sigma}\langle U\rangle((\Diamond^{-1}_{=\mathtt{b}+2}\top \wedge \mathbf{dw}^{\Box}(\alpha, 0) \wedge \neg\Diamond\top \wedge t^{\Box}_{\mathtt{a}}(\phi)) * (\mathbf{dw}^{\Box}(\alpha, 0) \wedge \neg\Diamond^{-1}\top \wedge t^{\Box}_{\mathtt{b}}(\psi)))$,

  — $\mathsf{chop2}^{\Box}_{\mathtt{a}} \overset{\text{def}}{=} (\bigvee_{\mathtt{b}\in\Sigma}\langle U\rangle\ ((\Diamond^{-1}\top \wedge \neg\Diamond\top)\ \wedge \Diamond^{-1}_{=\mathtt{b}+2}\top) \wedge (t^{\Box}_{\mathtt{a}}(\phi) * (t^{\Box}_{\mathtt{b}}(\psi) \wedge ([U]\neg\Diamond\top))))$,

  — $\mathsf{chop3}^{\Box}_{\mathtt{a}} \overset{\text{def}}{=} ((t^{\Box}_{\mathtt{a}}(\phi) \wedge ([U]\neg\Diamond\top)) * t^{\Box}_{\mathtt{a}}(\psi))$.

LEMMA 3.11. *Let $\alpha \geq 1$, $\Sigma = [1, \alpha]$, ϕ be a PITL$_\Sigma$ formula and $t^{\Box}(\phi)$ be its translation in MLH. We have ϕ is satisfiable iff $t^{\Box}(\phi)$ is satisfiable.*

The proof goes as in the case for the direct translation into 1SL2($*$) since the modal subformulae express exactly the same properties. Therefore, we can refine Theorem 3.8 as follows.

THEOREM 3.12. *The satisfiability problem for MLH($*$) is decidable but not elementary recursive.*

Interestingly, we do not know the decidability status for full MLH (i.e., with the magic wand operator).

4. HOW TWO VARIABLES WITH THE MAGIC WAND ENCODE RUNS

In this section, we consider the logic 1SL2 (i.e., equipped with both the separating conjunction *and* the separating implication), and we prove its undecidability. In order to show that the 1SL2 satisfiability problem is undecidable, we reduce the halting problem for Minsky machines [Minsky 1967].

4.1. Minsky machines in a nutshell

Let M be a Minsky machine with $\alpha \geq 1$ instructions, where 1 is the initial instruction and α is the halting instruction. Machine M has two counters $\mathtt{C}_1$ and $\mathtt{C}_2$ and the instructions are of the following types:

(a) I: $\mathtt{C}_j := \mathtt{C}_j + 1$; goto J.
(b) I: if $\mathtt{C}_j = 0$ then goto J_1 else ($\mathtt{C}_j := \mathtt{C}_j - 1$; goto J_2).
(c) α: halt.

($j \in [1, 2]$, $I \in [1, \alpha - 1]$, $J, J_1, J_2 \in [1, \alpha]$)

Machine M halts if there is a run of the form

$$(I_0, c^1_0, c^2_0), (I_1, c^1_1, c^2_1), \ldots, (I_L, c^1_L, c^2_L)$$

such that $(I_i, c^1_i, c^2_i) \in [1, \alpha] \times \mathbb{N}^2$ ($i \in [1, L]$), the succession of configurations respects the instructions (in the obvious way), $I_0 = 1$, $I_L = \alpha$, and $c^1_0 = c^2_0 = 0$. The halting problem consists in checking whether a machine halts and it is known to be undecidable, see e.g. [Minsky 1967]. Clearly, a halting run is a data word $\mathfrak{dw}$ of dimension 2 such that the first letter is 1 and the last letter is α.

When a Minsky machine M has $\alpha \geq 1$ instructions, any run starting from the initial instruction 1 and ending by the halting instruction α (there is a single such run since M is deterministic) is a data word of dimension two over the finite alphabet $[1, \alpha]$. The main and obvious idea to get undecidability is to show how 1SL2 can characterize heaps encoding data words of dimension two corresponding to halting runs of M.

4.2. Roadmap

Let us start by explaining how the rest of the section is structured. The next paragraph describes how initial and final conditions on the run are encoded in 1SL2; typically the run starts by instruction 1 and possibly ends by the halting instruction α. Then, the next two paragraphs deal with the description of two problems that we need to tackle. (Section 4.3 and Section 4.4 contain the technical developments.) In a nutshell, the first problem consists of being able to compare the numbers of predecessors of two locations (which corresponds to comparison of two counter values in Minsky machines), whereas the second problem is related to the fact that these two locations may be separated by distance three along the main path. Both issues stem from the fact that we have only two individual variables at hand. (It is already known how to solve these problems with an unbounded number of variables [Brochenin et al. 2012].) Section 4.5 provides the final definition of the reduction from the halting problem for Minsky machines to the satisfiability problem for 1SL2 whereas Section 4.6 presents the idea of an alternative undecidability proof by reduction from a first-order logic on data words (see Section 2.3). These proofs share essential building blocks, for instance the constructions allowing to compare numbers of predecessors. However, this provides different points of view as well.

Limit conditions. We have seen that $(\alpha, 2)$-fishbone heaps can be characterized thanks to the formula $\mathbf{dw}(\alpha, 2)$. Obviously, more constraints need to be expressed, typically those related to the first instruction and those related to the halting instruction. Let us start by specifying the limit conditions thanks to the formulae $\phi_{\rm first}$ and $\phi_{\rm last}$ below.

— The first three locations on the main path have 3, $\alpha + 3$, and $\alpha + 3$ predecessors respectively:

$$\phi_{\rm first} \overset{\text{def}}{=} \exists\, \mathtt{u}\ \mathtt{first}(\mathtt{u}) \wedge (\sharp \mathtt{u} = 3) \wedge (\sharp \mathtt{u}^{+1} = \alpha + 3) \wedge (\sharp \mathtt{u}^{+2} = \alpha + 3)$$

— The main path encoding the run ends by a configuration with the halting instruction:

$$\phi_{\rm last} \overset{\text{def}}{=} \exists\, \mathtt{u}\ ((\sharp \mathtt{u} = \alpha + 2) \wedge (\sharp \mathtt{u}^{+2} \geq 0) \wedge \neg(\sharp \mathtt{u}^{+3} \geq 0))$$

Let us call $\phi^{\star}$ the conjunction of $\mathbf{dw}(\alpha, 2) \wedge \phi_{\rm first} \wedge \phi_{\rm last}$. It specifies the shape of the encoding of the run without taking care of the constraints about counter values and instruction counter.

LEMMA 4.1. *Let $\mathfrak{h}$ be a heap. $\mathfrak{h} \models \phi^{\star}$ iff $\mathfrak{h}$ encodes a data word*

$$\mathfrak{dw} = (\mathtt{a}^1, \mathfrak{d}^1_1, \mathfrak{d}^1_2) \cdots (\mathtt{a}^L, \mathfrak{d}^L_1, \mathfrak{d}^L_2)$$

such that $\mathtt{a}^1 = 1$, $\mathtt{a}^L = \alpha$, *and* $\mathfrak{d}^1_1 = \mathfrak{d}^1_2 = 0$.

We have provided formulae for basic properties about the encoding of the runs, but this is insufficient. Indeed, three consecutive locations on the main path encode a configuration of the Minsky machine M. In order to check that two consecutive configurations correspond to a step that is valid for M, we need to compare numbers of predecessors for locations on the main path at distance three from each other. For instance, considering locations $\mathfrak{l}$ and $\mathfrak{l}'$ on the main path such that $\mathfrak{l}' = \mathfrak{h}^3(\mathfrak{l})$, we plan to build formulae to express constraints between $\widetilde{\sharp\mathfrak{l}}$ and $\widetilde{\sharp\mathfrak{l}'}$. There are two problems there.

Distance three. Firstly, with two variables, one can explore the heap but is obliged to "forget" previously visited locations (in fact, it is possible to store only a finite amount of information). Since [Gabbay 1981], it is known how to visit a graph with only two quantified variables. In the current encoding of runs, we will need to compare the numbers of predecessors of locations at distance three on the main path. Let us consider the formula below:

$$\exists\, \mathrm{u}\; [\forall\, \overline{\mathrm{u}}\; (\overline{\mathrm{u}} \hookrightarrow \mathrm{u}) \Rightarrow (\sharp\overline{\mathrm{u}} = 3 \vee \sharp\overline{\mathrm{u}} = 7)] \wedge [\exists\, \overline{\mathrm{u}}\; (\mathrm{u} \hookrightarrow \overline{\mathrm{u}}) \wedge \exists\, \mathrm{u}\; ((\overline{\mathrm{u}} \hookrightarrow \mathrm{u}) \wedge \sharp\mathrm{u} = 11)]$$

This formula states that there are two locations in the model for which there is a path of length 2 between them, the second location has exactly 11 predecessors and for the first one, every predecessor has either 3 or 7 predecessors.

In MLH (without the use of $*$ or $-\!\!*$), this property can be written as follows:

$$\langle U \rangle (\Box^{-1}(\Diamond^{-1}_{=3}\top \vee \Diamond^{-1}_{=7}\top) \wedge \Diamond\Diamond\Diamond^{-1}_{=11}\top)$$

So, it is possible to state properties between locations that are not direct successors, but note that we can only enforce properties while we are visiting the nodes: once we move forward or backward we have no more access to previously visited locations. This becomes a problem when we need to compare number of predecessors for locations at distance three. Observe that if we had proposed another encoding of the runs, at some stage we would have to deal with locations that are not direct successors and for which we need to compare some potentially unbounded amount of information (since we need anyhow to encode counter values that are potentially unbounded). Section 4.4 provides a solution to this problem by presenting a selective chopping of the heap that preserves the numbers of predecessors we wish to compare while being able to access easily those locations for which the number of predecessors are compared.

It is worth observing that for locations on the main path related to counter instructions, we are not in big trouble because it is possible with two quantified variables to visit two successive locations related to counter instructions and to check that they respect the instructions of the Minsky machine (because only a finite amount of information needs to be encoded). While moving along the model, we may "forget" where we start, but this is fine since we can remember the value of the counter instruction in the formula. By contrast, when comparing two locations corresponding to two successive values of the same counter, this does not work anymore.

Arithmetical constraints. Secondly, we have also to be able to compare numbers of predecessors between locations. For instance, given two locations $\mathfrak{l}$ and $\mathfrak{l}'$, we wish to be able to check whether $\widetilde{\sharp\mathfrak{l}} = \widetilde{\sharp\mathfrak{l}'}$ or $\widetilde{\sharp\mathfrak{l}} = \widetilde{\sharp\mathfrak{l}'} + 1$. Such a need should not come as a surprise, since in our encoding of data words, data values are represented by numbers of predecessors. Such arithmetical constraints can be expressed in 1SL (i.e. without limiting the number of quantified variables), and this has been a key step to establish 1SL's equivalence to weak second-order logic on heaps [Brochenin et al. 2012]. In Section 4.3, we provide a fine-tuned adaptation with only two quantified variables (instead of an unbounded number of variables) and with substantial simplifications. The developments in Section 4.3 are not consequences of developments from [Brochenin et al. 2012, Section 5.2] but rather refinements using similar principles. Actually, we can take advantage of the fact that we do not work on an arbitrary heap but rather on an $(\alpha, 2)$-fishbone heap. Moreover, we do not seek expressive completeness but rather we aim at expressing a sufficient set of arithmetical constraints to allow us to characterize $(\alpha, 2)$-fishbone heaps that encode halting runs of M.

4.3. Expressing arithmetical constraints

Below, we show how to express in 1SL2 the constraints $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}}$, $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}} + 1$ and $\sharp\overline{\mathrm{u}} = \sharp\mathrm{u} + 1$, which is not at all obvious. We explain why this can be done in 1SL2 by a careful recycling of variables; along the way, we also take advantage of the properties of heaps satisfying $\phi^{\star}$.

We shall use the fact that $N \leq N'$ iff for every $n \geq 0$, we have $N' \leq n$ implies $N \leq n$. Quantification over the set of natural numbers will be simulated by quantification over disjoint heaps in which n is related to the cardinal of their domains. Such quantification is performed thanks to the magic wand operator.

A *fork* in $\mathfrak{h}$ is a sequence of distinct locations $\mathfrak{l}, \mathfrak{l}_0, \mathfrak{l}_1, \mathfrak{l}_2$ such that $\mathfrak{h}(\mathfrak{l}_0) = \mathfrak{l}$, $\widetilde{\sharp\mathfrak{l}_0} = 2$, $\mathfrak{h}(\mathfrak{l}_1) = \mathfrak{l}_0$, $\mathfrak{h}(\mathfrak{l}_2) = \mathfrak{l}_0$ and $\widetilde{\sharp\mathfrak{l}_1} = \widetilde{\sharp\mathfrak{l}_2} = 0$. The *endpoint* of the fork is $\mathfrak{l}$. 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 $\mathfrak{l}$. By way of example, the heap of Figure 6 contains three knives, two forks and four endpoints (identified by '$\star$').

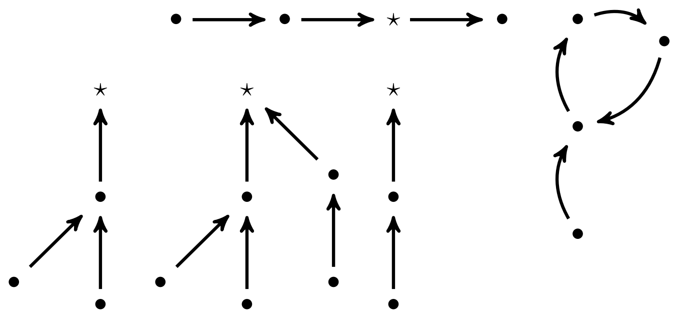

Fig. 6. A heap with three knives, two forks and four endpoints.

LEMMA 4.2. *Let $\mathfrak{h}$ be a (α,β)-fishbone heap with $\alpha \geq 1$ and $\beta \geq 0$. Then, $\mathfrak{h}$ has no knife and no fork.*

Indeed, in such heaps, any allocated location has no predecessor or at least three predecessors.

A heap $\mathfrak{h}$ is a *collection of knives* $\overset{\text{def}}{\Leftrightarrow}$ there is no location in $\mathrm{dom}(\mathfrak{h})$ that does not belong to a knife and no distinct knives share the same endpoint. 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.

LEMMA 4.3. *Let $\mathfrak{h}$ be a (α,β)-fishbone heap with $\alpha \geq 1$, $\beta \geq 0$ and $\mathfrak{h}'$ be a segmented heap disjoint from $\mathfrak{h}$. Then, $\mathfrak{h} \uplus \mathfrak{h}'$ has no fork.*

Being segmented can be naturally expressed in 1SL2:

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

The statement below is counterpart to [Brochenin et al. 2012, Lemma 5.2] with simplified properties and with simpler formulae but using only two quantified variables.

LEMMA 4.4. *There are formulae $\mathtt{forky}(\mathrm{u})$, $\mathtt{KS}$ and $\mathtt{KS1F}$ in 1SL2 such that for every heap $\mathfrak{h}$,*

(I) $\mathfrak{h} \models_f \mathtt{forky}(\mathrm{u})$ *iff all the predecessors of $f(\mathrm{u})$ are endpoints of forks,*
(II) $\mathfrak{h} \models \mathtt{KS}$ *iff $\mathfrak{h}$ is a collection of knives,*

(III) $\mathfrak{h} \models \texttt{KS1F}$ *iff there are* $\mathfrak{h}_1, \mathfrak{h}_2$ *such that* $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{h}_1$ *is a collection of knives and* $\mathfrak{h}_2$ *is made of a unique fork such that its unique endpoint is not in the range of* $\mathfrak{h}_1$.

PROOF. $\texttt{forky}(\mathtt{u})$ is equal to:

$$\forall\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \Rightarrow (\exists\, \mathtt{u}\ (\mathtt{u} \hookrightarrow \overline{\mathtt{u}}) \wedge (\sharp\mathtt{u} = 2) \wedge \neg(\sharp\mathtt{u}^{-1} \geq 1))$$

A knife is made of two consecutive memory cells that can be respectively called part 1 and part 2 as shown in $\mathfrak{l} \xrightarrow{\text{part 1}} \mathfrak{l}' \xrightarrow{\text{part 2}} \mathfrak{l}''$.

$$\texttt{KS} \stackrel{\text{def}}{=} \forall\, \mathtt{u}\ \texttt{alloc}(\mathtt{u}) \Rightarrow (\phi_{\text{part1}}(\mathtt{u}) \vee \phi_{\text{part2}}(\mathtt{u}))$$

where

$$\phi_{\text{part1}}(\mathtt{u}) \stackrel{\text{def}}{=} (\sharp\mathtt{u} = 0) \wedge (\sharp\mathtt{u}^{+1} = 1) \wedge (\sharp\mathtt{u}^{+2} = 1) \wedge \neg(\sharp\mathtt{u}^{+3} \geq 0)$$

$$\phi_{\text{part2}}(\mathtt{u}) \stackrel{\text{def}}{=} (\sharp\mathtt{u} = 1) \wedge (\sharp\mathtt{u}^{-1} = 0) \wedge (\sharp\mathtt{u}^{+1} = 1) \wedge \neg(\sharp\mathtt{u}^{+2} \geq 0)$$

$$\texttt{KS1F} \stackrel{\text{def}}{=}$$

$$\overbrace{[\exists\, \mathtt{u}\ (\sharp\mathtt{u} = 2) \wedge (\sharp\mathtt{u}^{+1} = 1) \wedge \neg(\sharp\mathtt{u}^{+2} \geq 0) \wedge \neg(\sharp\mathtt{u}^{-1} \geq 1) \wedge \neg(\exists\, \overline{\mathtt{u}}\ (\mathtt{u} \neq \overline{\mathtt{u}}) \wedge (\sharp\overline{\mathtt{u}} = 2))]}^{\text{unique fork}} \wedge$$

$$[\forall\, \mathtt{u}\ \texttt{alloc}(\mathtt{u}) \Rightarrow$$

$$\underbrace{(\phi_{\text{part1}}(\mathtt{u}) \vee \phi_{\text{part2}}(\mathtt{u})}_{\text{part with knifes}} \vee \underbrace{((\sharp\mathtt{u} = 0) \wedge (\sharp\mathtt{u}^{+1} = 2)) \vee (\sharp\mathtt{u} = 2))}_{\text{part with one fork}}]$$

$\Box$

In our proof, we use the idea of augmenting the heap with a segmented heap, then augmenting it further with knives to form forks whose endpoints are predecessors of u; this is borrowed from [Brochenin et al. 2012]. As it is, this would not be sufficient to express arithmetical constraints on fishbone heaps since only two quantified variables are allowed. This restriction is not considered in [Brochenin et al. 2012]—the formulae there use strictly more than two quantified variables. This is why we had to provide specific developments that are well-tailored to fishbone heaps while taking into account our limited amount of syntactic resources. Simplifications have also been made in order to focus on undecidability rather than on questions of expressive power.

LEMMA 4.5. *Let* $\mathfrak{h}$ *be a heap with* $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ *and* f *be an assignment such that* $\mathfrak{h}_1 \models_f \phi^\star$, $f(\mathtt{u})$ *is on the main path of the* $(\alpha, 2)$*-fishbone heap* $\mathfrak{h}$, $\mathfrak{h}_2 \models_f \texttt{seg} \wedge \sharp\mathtt{u} = 0$, $n = card(ran(\mathfrak{h}_2) \setminus dom(\mathfrak{h}_1))$ *and* m *is the number of predecessors of* $f(\mathtt{u})$ *in* $\mathfrak{h}_1$. *We have the following properties:*

(a) $\mathfrak{h} \models_f \neg(\texttt{KS} \twoheadrightarrow \neg\texttt{forky}(\mathtt{u}))$ *iff* $n \geq m$.
(b) $\mathfrak{h} \models_f \neg(\texttt{KS1F} \twoheadrightarrow \neg\texttt{forky}(\mathtt{u}))$ *iff* $n \geq m - 1$.

In Figure 7, we present three heaps obtained by combining a segmented heap $\mathfrak{h}_2$ with collections of knives (corresponding to $\mathfrak{h}_3$ in the proof of Lemma 4.5). Edges labelled by '1' are part of a fishbone heap $\mathfrak{h}_1$ (partially represented) whereas edges labelled by '2' are part of a segmented heap $\mathfrak{h}_2$ so that no edge points to $f(\mathtt{u})$ or to $f(\overline{\mathtt{u}})$. The heap on the left (corresponding to $\mathfrak{h}_1 * \mathfrak{h}_2$ in Lemma 4.5) is obtained by adding a segmented heap $\mathfrak{h}_2$ whereas the heap in the middle (say $\mathfrak{h}_1 * \mathfrak{h}_2 * \mathfrak{h}_3$) is obtained then by adding a

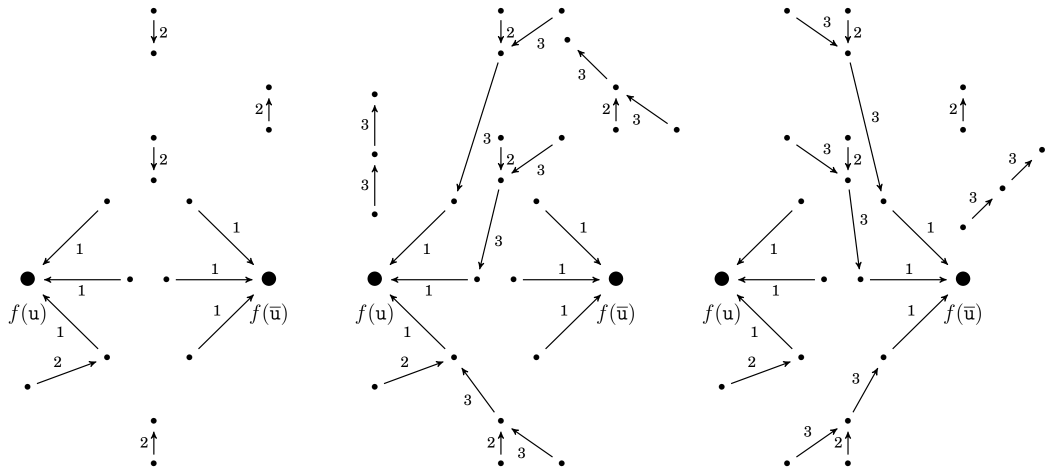

Fig. 7. A segmented heap and collections of knives.

collection of knives $\mathfrak{h}_3$ so that every predecessor of $f(\mathrm{u})$ is the endpoint of a fork. Note that not all edges of the segmented heap are used to build forks. Similarly, the heap on the right (say $\mathfrak{h}_1 * \mathfrak{h}_2 * \mathfrak{h}'_3$) is obtained then by adding a collection of knives $\mathfrak{h}'_3$ to the heap $\mathfrak{h}_1 * \mathfrak{h}_2$ on the very left so that every predecessor of $f(\bar{\mathrm{u}})$ is the endpoint of a fork.

PROOF. Let us provide the proof for (a). (The proof for (b), being analogous, is omitted.) So, let $\mathfrak{h}$ be a heap with $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ such that $\mathfrak{h}_1 \models_f \phi^\star$ and $\mathfrak{h}_2 \models_f \mathtt{seg} \wedge \sharp\mathrm{u} = 0$. Moreover, $f(\mathrm{u})$ is on the main path, which entails that $\mathfrak{h}(f(\mathrm{u})) \neq f(\mathrm{u})$ (if $\mathfrak{h}(f(\mathrm{u}))$ is defined at all) and $f(\mathrm{u})$ has at least one predecessor.

One can make the following (obvious) observations.

(O1). The heap $\mathfrak{h}_1$ has no knives and, $\mathfrak{h}_1$ and $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ have no forks. (see Lemma 4.2 and Lemma 4.3).
(O2). $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ may not satisfy $\phi^\star$ but this is fine since we only need to focus on the number of predecessors of $f(\mathrm{u})$ (i.e., on the value m). Indeed, $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ may contain knives (see the left heap in Figure 7). A knife $\mathfrak{l}_1 \to \mathfrak{l}_2 \to \mathfrak{l}_3$ in $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ is made of $\mathfrak{l}_1 \in \mathrm{dom}(\mathfrak{h}_2)$ and of $\mathfrak{l}_2 \in \mathrm{dom}(\mathfrak{h}_1)$. This observation is not really used below but, hopefully, it could be helpful to better grasp how the heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are combined.
(O3). $f(\mathrm{u})$ has the same number of predecessors in $\mathfrak{h}_1$ and in $\mathfrak{h}_1 \uplus \mathfrak{h}_2$. This is due to the fact that $\mathfrak{h}_2 \models_f \sharp\mathrm{u} = 0$.
(O4). For every $n \geq 0$, there is a disjoint heap $\mathfrak{h}'_2$ such that $\mathfrak{h}' = \mathfrak{h}_1 \uplus \mathfrak{h}'_2$, $\mathfrak{h}'_2 \models_f \mathtt{seg} \wedge \sharp\mathrm{u} = 0$ and $\mathrm{card}(\mathrm{ran}(\mathfrak{h}'_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = n$. See the left heap in Figure 7 with $\mathrm{card}(\mathrm{dom}(\mathfrak{h}_2)) = 5$ and $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = 4$ (look at edges labelled by '2'). Once more, this observation is not used below but it will be in the proof of Proposition 4.6.

First, let us suppose that $\mathfrak{h} \models_f \neg(\mathtt{KS} \twoheadrightarrow \neg\mathtt{forky}(\mathrm{u}))$, i.e., (†) there is a heap $\mathfrak{h}_3$, disjoint from $\mathfrak{h}_1 \uplus \mathfrak{h}_2$, such that $(\mathfrak{h}_1 \uplus \mathfrak{h}_2) \uplus \mathfrak{h}_3 \models_f \mathtt{forky}(\mathrm{u})$ and $\mathfrak{h}_3 \models_f \mathtt{KS}$. Let us make additional observations.

— The only forks in $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3$ whose endpoints are predecessors of $f(\mathrm{u})$ are those obtained with $\mathfrak{l}_1 \to \mathfrak{l}_2$ such that $\mathfrak{l}_1 \in \mathrm{dom}(\mathfrak{h}_2)$ (so $\mathfrak{h}_2(\mathfrak{l}_1) = \mathfrak{l}_2$), $\mathfrak{l}_2 \notin \mathrm{dom}(\mathfrak{h}_1)$, and $\mathfrak{l}'_1 \to \mathfrak{l}_2 \to \mathfrak{l}'_3$ is a knife from $\mathfrak{h}_3$. This is due to (O1) and to the fact that all the predecessors of $f(\mathrm{u})$ in $\mathfrak{h}$ have no predecessors since $f(\mathrm{u})$ is on the main path of $\mathfrak{h}$.

— The number of forks in $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3$ whose endpoints are predecessors of $f(\mathtt{u})$ is therefore less of equal to $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1))$.
— The number of predecessors of $f(\mathtt{u})$ in $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3$ is greater or equal to the number of its predecessors in $\mathfrak{h}_1$ (by using (O3)). So, if $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3 \models_f \mathtt{forky}(\mathtt{u})$, then the number of predecessors of $f(\mathtt{u})$ in $\mathfrak{h}_1$ is smaller or equal to $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = n$, i.e. $n \geq m$.

Now, let us establish the other direction and let us suppose that $n \geq m$ and the predecessors of $f(\mathtt{u})$ are $\mathfrak{p}_1, \ldots, \mathfrak{p}_m$. Let $\mathfrak{l}_1^1, \mathfrak{l}_1^2, \ldots, \mathfrak{l}_n^1, \mathfrak{l}_n^2$ be locations such that $\{\mathfrak{l}_1^1, \ldots, \mathfrak{l}_n^1\} = \mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)$ and for every $i \in [1, n]$, we have $\mathfrak{h}_2(\mathfrak{l}_i^2) = \mathfrak{l}_i^1$. Let us build $\mathfrak{h}_3$ so that it satisfies (†), which is quite easy to realize. Let $\mathfrak{l}_1^{\mathrm{new}}, \ldots, \mathfrak{l}_m^{\mathrm{new}}$ be (new) locations that are not in $\mathrm{dom}(\mathfrak{h}_1 \uplus \mathfrak{h}_2) \cup \mathrm{ran}(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$. We define $\mathfrak{h}_3$ so that it contains exactly m knives whose endpoints are exactly all the predecessors of $f(\mathtt{u})$. For every $i \in [1, m]$, we define $\mathfrak{h}_3(\mathfrak{l}_i^{\mathrm{new}}) \stackrel{\text{def}}{=} \mathfrak{l}_i^1$ and $\mathfrak{h}_3(\mathfrak{l}_i^1) \stackrel{\text{def}}{=} \mathfrak{p}_i$ (well, that is possible because $\mathfrak{l}_i^1 \notin \mathrm{dom}(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$). It is easy to check that $\mathfrak{h}_3$ satisfies (†).
Consequently, $\mathfrak{h} \models_f \neg(\mathtt{KS} \twoheadrightarrow \neg\mathtt{forky}(\mathtt{u}))$ iff $n \geq m$. $\square$

Now, we are able to state the main proposition of this section that allows us to compare the numbers of predecessors for two locations on the main path of a fishbone heap. Let us introduce the following abbreviations: $\varphi_1(\mathtt{u}, \bar{\mathtt{u}}) \stackrel{\text{def}}{=} \mathtt{seg} \wedge \sharp\mathtt{u} = 0 \wedge \sharp\bar{\mathtt{u}} = 0$, $\varphi_2(\mathtt{u}) \stackrel{\text{def}}{=} \neg(\mathtt{KS} \twoheadrightarrow \neg\mathtt{forky}(\mathtt{u}))$ and $\varphi_3(\mathtt{u}) \stackrel{\text{def}}{=} \neg(\mathtt{KS1F} \twoheadrightarrow \neg\mathtt{forky}(\mathtt{u}))$.

PROPOSITION 4.6. *Suppose $\mathfrak{h}_1 \models_f \phi^\star$ and, $f(\mathtt{u})$ and $f(\bar{\mathtt{u}})$ are on the main path of $\mathfrak{h}_1$. We have the following equivalences:*

— $\mathfrak{h}_1 \models_f \varphi_1(\mathtt{u}, \bar{\mathtt{u}}) \twoheadrightarrow (\varphi_2(\bar{\mathtt{u}}) \Rightarrow \varphi_2(\mathtt{u}))$ *iff* $\widetilde{\sharp\mathtt{u}} \leq \widetilde{\sharp\bar{\mathtt{u}}}$.
— $\mathfrak{h}_1 \models_f \varphi_1(\mathtt{u}, \bar{\mathtt{u}}) \twoheadrightarrow (\varphi_2(\bar{\mathtt{u}}) \Rightarrow \varphi_3(\mathtt{u}))$ *iff* $\widetilde{\sharp\mathtt{u}} \leq \widetilde{\sharp\bar{\mathtt{u}}} + 1$.
— $\mathfrak{h}_1 \models_f \varphi_1(\mathtt{u}, \bar{\mathtt{u}}) \twoheadrightarrow (\varphi_3(\bar{\mathtt{u}}) \Rightarrow \varphi_2(\mathtt{u}))$ *iff* $\widetilde{\sharp\mathtt{u}} \leq \widetilde{\sharp\bar{\mathtt{u}}} - 1$.

PROOF. By way of example, let us show the second property. The other cases are proved in a similar fashion. Let $\mathfrak{h}_1$ be a heap satisfying $\phi^\star$. The statements below are equivalent.

(1) $\mathfrak{h}_1 \models_f (\varphi_1(\mathtt{u}, \bar{\mathtt{u}}) \twoheadrightarrow (\varphi_2(\bar{\mathtt{u}}) \Rightarrow \varphi_3(\mathtt{u}))))$.
(2) For every disjoint heap $\mathfrak{h}_2$ such that $\mathfrak{h}_2 \models_f \varphi_1(\mathtt{u}, \bar{\mathtt{u}})$, if $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_f \varphi_2(\bar{\mathtt{u}})$, then $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_f \models \varphi_3(\mathtt{u})$. (by definition of $\models_f$)
(3) For every $n \geq 0$, there is a disjoint heap $\mathfrak{h}_2$ with $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = n$ such that $\mathfrak{h}_2 \models_f \varphi_1(\mathtt{u}, \bar{\mathtt{u}})$ and if $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_f \varphi_2(\bar{\mathtt{u}})$, then $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_f \varphi_3(\mathtt{u})$ (see (O4) in the proof of Lemma 4.5). This is possible by using the fact that one can add a segmented heap so that the resulting heap has n isolated memory cells. Indeed, given the heap $\mathfrak{h}_1$, let us build a disjoint heap $\mathfrak{h}_2$ such that $\mathfrak{h}_2 \models_f \varphi_1(\mathtt{u}, \bar{\mathtt{u}})$ and $\mathrm{dom}(\mathfrak{h}_2) = n$ for any fixed $n \geq 0$. Since $X = \mathrm{dom}(\mathfrak{h}_1) \cup \mathrm{ran}(\mathfrak{h}_2) \cup \{f(\mathtt{u}), f(\bar{\mathtt{u}})\}$ is a finite subset of $\mathbb{N}$, there are $2n$ distinct locations $\mathfrak{l}_1^1, \mathfrak{l}_1^2, \ldots, \mathfrak{l}_n^1, \mathfrak{l}_n^2$ in $\mathbb{N} \setminus X$. We simply need to define $\mathfrak{h}_2$ such that $\mathrm{dom}(\mathfrak{h}_2) \stackrel{\text{def}}{=} \{\mathfrak{l}_1^1, \ldots, \mathfrak{l}_n^1\}$, $\mathrm{ran}(\mathfrak{h}_2) \stackrel{\text{def}}{=} \{\mathfrak{l}_1^2, \ldots, \mathfrak{l}_n^2\}$ and for all $i \in [1, n]$, we set $\mathfrak{h}_2(\mathfrak{l}_i^1) \stackrel{\text{def}}{=} \mathfrak{l}_i^2$.
(4) for every $n \geq 0$, we have $n \geq \widetilde{\sharp\bar{\mathtt{u}}}$ in $\mathfrak{h}$ implies $n \geq \widetilde{\sharp\mathtt{u}} - 1$ in $\mathfrak{h}$. (by Lemma 4.5)
(5) $\widetilde{\sharp\mathtt{u}} \leq \widetilde{\sharp\bar{\mathtt{u}}} + 1$. $\square$

4.4. Constraints between locations at distance three

The goal of this section is the following: given a formula $\varphi(\mathtt{u}, \bar{\mathtt{u}})$ equal to either $\sharp\mathtt{u} = \sharp\bar{\mathtt{u}}$ or $\sharp\mathtt{u} = \sharp\bar{\mathtt{u}} + 1$ (in particular, this means that $\varphi(\mathtt{u}, \bar{\mathtt{u}})$ only deals with numbers of predecessors and Section 4.3 explains how to define these formulae in 1SL2), we show how to define a formula in 1SL2, say $\varphi^{+3}(\mathrm{u})$, such that

$$\mathfrak{h} \models_f \varphi^{+3}(\mathrm{u}) \text{ iff } \mathfrak{h} \models_{f[\overline{\mathrm{u}} \mapsto \mathfrak{h}^3(f(\mathrm{u}))]} \varphi(\mathrm{u}, \overline{\mathrm{u}}),$$

assuming that $\mathfrak{h}^3(f(\mathrm{u}))$ is defined and $\mathfrak{h} \models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathrm{u} \geq \alpha + 3))$. When $\varphi(\mathrm{u}, \overline{\mathrm{u}})$ is equal to $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}}$ [resp. $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}} + 1$], we write $\sharp\mathrm{u} = \sharp\mathrm{u}^{+3}$ [resp. $\sharp\mathrm{u} = \sharp\mathrm{u}^{+3} + 1$] instead of $\varphi^{+3}(\mathrm{u})$. Note that if we had three quantified variables, defining $\varphi^{+3}(\mathrm{u})$ would not require much work since the formula below does the job:

$$\exists\, \mathrm{u}' \,(\mathrm{u} \hookrightarrow \mathrm{u}' \wedge \exists\, \overline{\mathrm{u}} \,(\mathrm{u}' \hookrightarrow \overline{\mathrm{u}} \wedge \exists\, \mathrm{u}' \,(\overline{\mathrm{u}} \hookrightarrow \mathrm{u}' \wedge \varphi(\mathrm{u}, \mathrm{u}'))))$$

Let us start our construction. To do so, let $\mathfrak{h}$ be a heap and f be an assignment such that $\mathfrak{h} \models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathrm{u}^{+3} \geq 0) \wedge (\sharp\mathrm{u} \geq \alpha + 3))$. In the statements below, this property is always satisfied.

The u-*3cut* of $\mathfrak{h}$ is the minimal subheap $\mathfrak{h}_{3cut}$ of $\mathfrak{h}$ (with respect to set inclusion of the domain) such that all the ancestors of $\mathfrak{l}' = \mathfrak{h}^3(f(\mathrm{u}))$ in $\mathrm{dom}(\mathfrak{h})$ are also ancestors of $\mathfrak{l}'$ in $\mathfrak{h}_{3cut}$. It looks like a clean cut from Section 3 but operated on $\mathfrak{h}^3(f(\mathrm{u}))$ and on an $(\alpha, 2)$-fishbone heap. As a consequence, $f(\mathrm{u})$ and $\mathfrak{l}'$ have the same amount of predecessors in $\mathfrak{h}$ and in the u-3cut heap. Moreover, if $f(\mathrm{u})$ has more than $\alpha + 3$ predecessors, then the u-3cut of $\mathfrak{h}$ is also a $(\alpha, 2)$-fishbone heap.

In Figure 8, the bottom left heap is the u-3cut of the heap at the top. When $\mathfrak{h} \models_f \sharp\mathrm{u}^{+4} \geq 0$, the *almost* u-*3cut* of $\mathfrak{h}$ is the minimal subheap of $\mathfrak{h}$ containing the u-3cut heap and such that $\sharp\mathrm{u}^{+4} = 1$ holds true. The almost u-3cut of $\mathfrak{h}$ contains the edge from $\mathfrak{l}'$ which is the only predecessor of the interpretation of u^{+4}. In Figure 8, the middle left heap is the almost u-3cut of the heap at the top. Below, we explain how to obtain the u-3cut of some heap, possibly via the construction of the almost u-3cut, if it exists.

Lemma 4.7 below states that all we need to define $\varphi^{+3}(\mathrm{u})$ is to be able to express a property in its u-3cut. In particular, the only location that is unallocated and on the main path is $\mathfrak{h}^3(f(\mathrm{u}))$.

LEMMA 4.7. *Let* $\mathfrak{h} \models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathrm{u}^{+3} \geq 0) \wedge (\sharp\mathrm{u} \geq \alpha + 3))$ *and* $\mathfrak{h}'$ *be its* u-*3cut heap. Then,* $\mathfrak{h} \models_{f[\overline{\mathrm{u}} \mapsto \mathfrak{h}^3(f(\mathrm{u}))]} \varphi(\mathrm{u}, \overline{\mathrm{u}})$ *iff* $\mathfrak{h}' \models_f (\exists\, \overline{\mathrm{u}}\, \neg\mathtt{alloc}(\overline{\mathrm{u}}) \wedge \sharp\overline{\mathrm{u}} \geq 1 \wedge \varphi(\mathrm{u}, \overline{\mathrm{u}}))$.

PROOF. Let $\mathfrak{l}' = \mathfrak{h}^3(f(\mathrm{u}))$ and $\mathfrak{l} = f(\mathrm{u})$. Let $\mathfrak{h}'$ be the u-3cut heap of $\mathfrak{h}$. We have (†) $\widetilde{\sharp\mathfrak{l}}$ in $\mathfrak{h}$ is equal to $\widetilde{\sharp\mathfrak{l}}$ in $\mathfrak{h}'$ and $\widetilde{\sharp\mathfrak{l}'}$ in $\mathfrak{h}$ is equal to $\widetilde{\sharp\mathfrak{l}'}$ in $\mathfrak{h}'$. Indeed, the u-3cut heap $\mathfrak{h}'$ is a subheap of $\mathfrak{h}$ such that all the ancestors of $\mathfrak{l}'$ in $\mathfrak{h}$ are also ancestors of $\mathfrak{l}'$ in $\mathfrak{h}'$ and $\mathfrak{l}$ is an ancestor of $\mathfrak{l}'$ in $\mathfrak{h}$. Note also that $\mathfrak{l}'$ is the unique location such that $\mathfrak{h}' \models_{[\overline{\mathrm{u}} \mapsto \mathfrak{l}']} \neg\mathtt{alloc}(\overline{\mathrm{u}}) \wedge \sharp\overline{\mathrm{u}} \geq 1$. So, $\mathfrak{h}' \models_f (\exists\, \overline{\mathrm{u}}\, \neg\mathtt{alloc}(\overline{\mathrm{u}}) \wedge \sharp\overline{\mathrm{u}} \geq 1 \wedge \varphi(\mathrm{u}, \overline{\mathrm{u}}))$ iff $\mathfrak{h}' \models_{f[\overline{\mathrm{u}} \mapsto \mathfrak{l}']} \varphi(\mathrm{u}, \overline{\mathrm{u}}))$ iff $\mathfrak{h} \models_{f[\overline{\mathrm{u}} \mapsto \mathfrak{h}^3(f(\mathrm{u}))]} \varphi(\mathrm{u}, \overline{\mathrm{u}})$ by (†). Note that we use the fact that $\varphi(\mathrm{u}, \overline{\mathrm{u}})$ specifies a property about the numbers of predecessors. □

When $\mathfrak{h}$ is equal to its u-3cut, i.e. when $(\sharp\mathrm{u}^{+4} \geq 0)$ does not hold, we have $\mathfrak{h} \models_{f[\overline{\mathrm{u}} \mapsto \mathfrak{h}^3(f(\mathrm{u}))]} \varphi(\mathrm{u}, \overline{\mathrm{u}})$ iff $\mathfrak{h} \models_f \phi_{\mathrm{UC}}(\mathrm{u})$ with

$$\phi_{\mathrm{UC}}(\mathrm{u}) \overset{\text{def}}{=} (\exists\, \overline{\mathrm{u}}\, \neg\mathtt{alloc}(\overline{\mathrm{u}}) \wedge \sharp\overline{\mathrm{u}} \geq 1 \wedge \varphi(\mathrm{u}, \overline{\mathrm{u}}))$$

Now, let us consider the case when $\mathfrak{h}$ is not equal to its u-3cut (probably, the most common situation) and let us show how to separate the current heap so that we can isolate the u-3cut heap.

LEMMA 4.8. *Let* $\mathfrak{h} \models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathrm{u}^{+4} \geq 0) \wedge (\sharp\mathrm{u} \geq \alpha + 3))$ *and* $\phi(\mathrm{u})$ *be an arbitrary formula. Then,* $\mathfrak{h} \models_f \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp\mathrm{u}^{+4} = 1) \wedge \neg(\sharp\mathrm{u}^{+5} \geq 0) \wedge \phi(\mathrm{u}))$ *iff the almost* u-*3cut of* $\mathfrak{h}$*, say* $\mathfrak{h}'$*, satisfies:* $\mathfrak{h}' \models_f \phi(\mathrm{u})$.

The formula 1comp was introduced in Section 2.2, and it states that the heap is made of a unique connected component. The way $\mathfrak{h}$ has to be divided to satisfy the formula is illustrated by the two heaps in the middle of Figure 8.

PROOF. Let $\mathfrak{h}$ be heap such that $\mathfrak{h} \models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp u^{+4} \geq 0) \wedge (\sharp u \geq \alpha + 3)$. Let $\mathfrak{h}'$ be the almost u-3cut heap of $\mathfrak{h}$ and $\mathfrak{h}''$ be the heap such that $\mathfrak{h} = \mathfrak{h}' \uplus \mathfrak{h}''$. By construction of $\mathfrak{h}'$, it is easy to check that $\mathfrak{h}' \models_f \mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0)$. Similarly, $\mathfrak{h}'' \models_f \mathtt{1comp}$. This implies that $\mathfrak{h} \models_f \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0))$.

So, suppose that the almost u-3cut heap of $\mathfrak{h}$ satisfies: $\mathfrak{h}' \models_f \phi(u)$. This means that $\mathfrak{h}'' \models_f \mathtt{1comp}$ and $\mathfrak{h}' \models_f (\mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0) \wedge \phi(u))$. Hence, $\mathfrak{h} \models_f \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0) \wedge \phi(u))$.

Now, suppose that $\mathfrak{h} \models_f \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0) \wedge \phi(u))$. There are heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathfrak{h}_2 \models \mathtt{1comp}$ and $\mathfrak{h}_1 \models_f (\mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0) \wedge \phi(u))$. In particular, this means that $\mathfrak{h}_1 \models_f \mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0)$.

Let us show that there is a unique pair $(\mathfrak{h}_1, \mathfrak{h}_2)$ of heaps satisfying that property and $\mathfrak{h}_1 = \mathfrak{h}'$, which will entail that $\mathfrak{h}' \models_f \phi(u)$. First note that

$$\{f(u), \mathfrak{h}(f(u)), \mathfrak{h}^2(f(u)), \mathfrak{h}^3(f(u)), \mathfrak{h}^4(f(u))\} \subseteq \mathbf{dom}(\mathfrak{h}_1) \quad \mathfrak{h}^5(f(u)) \notin \mathbf{dom}(\mathfrak{h}_1)$$

Since $\mathfrak{h}_1 \models_f (\sharp u^{+4} = 1)$, all the predecessors of $\mathfrak{h}^4(f(u))$, apart from $\mathfrak{h}^3(f(u))$, are in $\mathrm{dom}(\mathfrak{h}_2)$ and there are more than two such predecessors since $\mathfrak{h}^4(f(u))$ is on the main path of $\mathfrak{h}$ and therefore has at least three predecessors in $\mathfrak{h}$.

Hence, $\mathfrak{h}_1$ contains also all the ancestors of $\mathfrak{h}^3(f(u))$, otherwise $\mathfrak{h}_2$ would have at least two distinct connected components. So, the u-3cut of $\mathfrak{h}$ is also a subheap of $\mathfrak{h}_1$.

Now, it is easy to check that if any location in $\mathrm{dom}(\mathfrak{h}'')$ that is not a predecessor of $\mathfrak{h}^4(f(u))$ were in $\mathrm{dom}(\mathfrak{h}_1)$, then $\mathfrak{h}_1$ would have more than two connected components. Hence, $\mathfrak{h}_1$ is the almost u-3cut heap of $\mathfrak{h}$ and therefore $\mathfrak{h}' \models_f \phi(u)$. $\square$

Let us build on Lemma 4.8 so as to be able to specify properties on the u-3cut heap.

LEMMA 4.9. *Let* $\mathfrak{h} \models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp u^{+4} \geq 0) \wedge (\sharp u \geq \alpha + 3))$ *and* $\phi(u)$ *be the formula* $(\mathtt{size} = 1) * (\neg(\sharp u^{+4} \geq 0) \wedge \phi_{\mathrm{UC}}(u))$. *Then,* $\mathfrak{h} \models_f \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0) \wedge \phi(u))$ *iff the* u*-3cut of* $\mathfrak{h}$*, say* $\mathfrak{h}'$*, satisfies:* $\mathfrak{h}' \models_f \phi_{\mathrm{UC}}(u)$.

We write $\phi_{\mathrm{AUC}}(u)$ to denote the formula $\mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp u^{+4} = 1) \wedge \neg(\sharp u^{+5} \geq 0) \wedge \phi(u))$ with $\phi(u)$ equal to $(\mathtt{size} = 1) * (\neg(\sharp u^{+4} \geq 0) \wedge \phi_{\mathrm{UC}}(u))$.

The proof for Lemma 4.9 is also by an easy verification by observing that an almost u-3cut heap is equal to the u-3cut plus one memory cell (see Figure 8).

By combining Lemma 4.7–4.9, we get the following proposition by performing a case analysis depending whether $\sharp u^{+4} \geq 0$ holds true or not on the heap $\mathfrak{h}$.

PROPOSITION 4.10. *Let* $\mathfrak{h}$ *be a heap and* f *be an assignment such that* $\mathfrak{h} \models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp u^{+3} \geq 0) \wedge (\sharp u \geq \alpha + 3))$. *We have* $\mathfrak{h} \models_{f[\overline{u} \mapsto \mathfrak{h}^3(f(u))]} \varphi(u, \overline{u})$ *iff* $\mathfrak{h} \models_f \varphi^{+3}(u)$ *with the formula* $\varphi^{+3}(u)$ *defined below:*

$$\varphi^{+3}(u) \stackrel{\text{def}}{=} (\neg(\sharp u^{+4} \geq 0) \wedge \phi_{\mathrm{UC}}(u)) \vee ((\sharp u^{+4} \geq 0) \wedge \phi_{\mathrm{AUC}}(u))$$

PROOF. We distinguish two cases depending whether $\mathfrak{h}$ is itself its u-3cut or not.
Case 1: $\neg(\sharp u^{+4} \geq 0)$, i.e. $\mathfrak{h}$ is its own u-3cut heap. By Lemma 4.7, if $\mathfrak{h} \models_{f[\overline{u} \mapsto \mathfrak{h}^3(f(u))]} \varphi(u, \overline{u})$, then $\mathfrak{h} \models_f \phi_{\mathrm{UC}}(u)$ and therefore $\mathfrak{h} \models_f \varphi^{+3}(u)$. Conversely, if $\mathfrak{h} \models_f \varphi^{+3}(u)$, then $\mathfrak{h} \models_f \phi_{\mathrm{UC}}(u)$ since $(\sharp u^{+4} \geq 0)$ does not hold on $\mathfrak{h}$. Again, by Lemma 4.7, we get that $\mathfrak{h} \models_{f[\overline{u} \mapsto \mathfrak{h}^3(f(u))]} \varphi(u, \overline{u})$.
Case 2: $(\sharp u^{+4} \geq 0)$. By Lemma 4.7, if $\mathfrak{h} \models_{f[\overline{u} \mapsto \mathfrak{h}^3(f(u))]} \varphi(u, \overline{u})$, then $\mathfrak{h}' \models_f \phi_{\mathrm{UC}}(u)$ where $\mathfrak{h}'$ is the u-3cut of $\mathfrak{h}$. By Lemma 4.9, this implies that $\mathfrak{h} \models_f \phi_{\mathrm{AUC}}(u)$ and therefore $\mathfrak{h} \models_f$

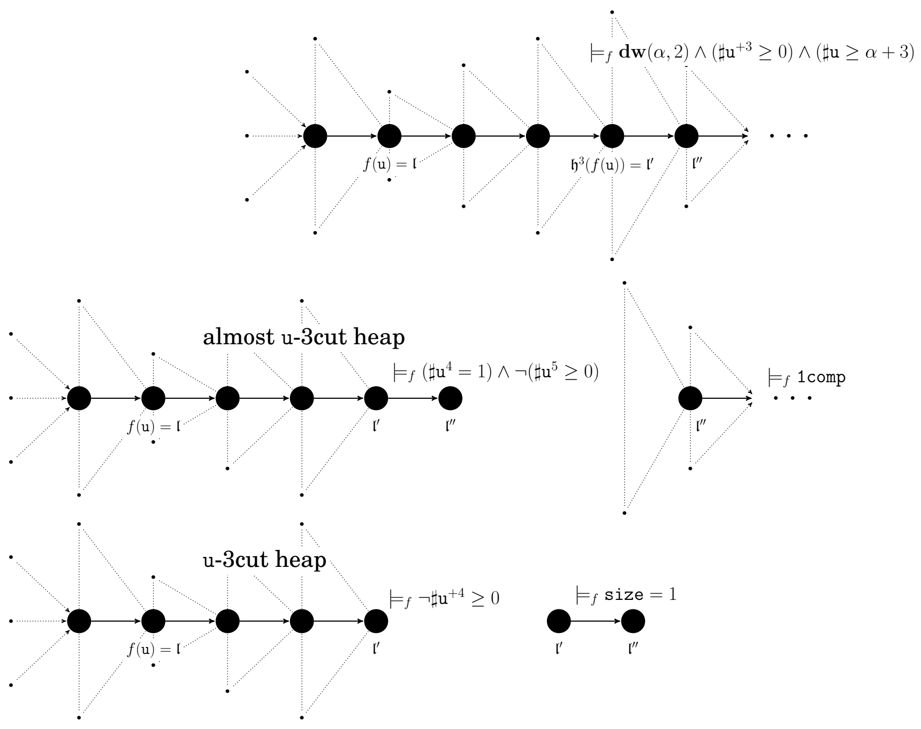
$\models_f \mathbf{dw}(\alpha, 2) \wedge (\sharp \mathrm{u}^{+3} \geq 0) \wedge (\sharp \mathrm{u} \geq \alpha + 3)$
$f(\mathrm{u}) = \mathfrak{l}$
$\mathfrak{h}^3(f(\mathrm{u})) = \mathfrak{l}'$
$\mathfrak{l}''$
almost u-3cut heap
$\models_f (\sharp \mathrm{u}^4 = 1) \wedge \neg(\sharp \mathrm{u}^5 \geq 0)$
$f(\mathrm{u}) = \mathfrak{l}$
$\mathfrak{l}'$
$\mathfrak{l}''$
$\models_f$ 1comp
$\mathfrak{l}''$
u-3cut heap
$\models_f \neg \sharp \mathrm{u}^{+4} \geq 0$
$f(\mathrm{u}) = \mathfrak{l}$
$\mathfrak{l}'$
$\models_f$ size $= 1$
$\mathfrak{l}'$
$\mathfrak{l}''$

Fig. 8. How to get a u-3cut – Decomposition in two stages.

$\varphi^{+3}(\mathrm{u})$ (thanks to its second disjunction). Conversely, if $\mathfrak{h} \models_f \varphi^{+3}(\mathrm{u})$, then $\mathfrak{h} \models_f \phi_{\mathrm{AUC}}(\mathrm{u})$ since $(\sharp \mathrm{u}^{+4} \geq 0)$ holds on $\mathfrak{h}$. Again, by Lemma 4.9, we get that $\mathfrak{h}' \models_{f[\bar{\mathrm{u}} \mapsto \mathfrak{h}^3(f(\mathrm{u}))]} \phi_{\mathrm{UC}}(\mathrm{u})$ and by Lemma 4.7, we conclude $\mathfrak{h} \models_{f[\bar{\mathrm{u}} \mapsto \mathfrak{h}^3(f(\mathrm{u}))]} \varphi(\mathrm{u}, \bar{\mathrm{u}})$. $\Box$

Note that the reasoning performed in this section cannot be extended to an arbitrary formula $\varphi(\mathrm{u}, \bar{\mathrm{u}})$ since taking a u-3cut or an almost u-3cut preserves the number of predecessors of $f(\mathrm{u})$ and $\mathfrak{h}^3(f(\mathrm{u}))$ but may not preserve more general properties. Nevertheless, this is sufficient for our needs in Section 4.5.

4.5. Master reduction from halting problem for Minsky machines

We shall use formulae of the form $\varphi^{+3}(\mathrm{u})$ when $\varphi(\mathrm{u}, \bar{\mathrm{u}})$ expresses one of the following arithmetical constraints: $\sharp \mathrm{u} = \sharp \bar{\mathrm{u}}$, $\sharp \mathrm{u} = \sharp \bar{\mathrm{u}} + 1$ and $\sharp \bar{\mathrm{u}} = \sharp \mathrm{u} + 1$ (see Section 4.4). For each instruction $I \in [1, \alpha - 1]$, we build a formula ϕ_I so that the Minsky machine M halts iff the formula $\phi^\star \ \wedge\ \bigwedge_{I \in [1, \alpha-1]} \varphi_I$ is satisfiable in 1SL2. It remains to define φ_I for each instruction I.

If instruction I is of the form "I: $\mathtt{C}_j := \mathtt{C}_j + 1$; goto J" then we need to check the following properties:

(1) If a location $\mathfrak{l}$ encodes the instruction I on the main path (i.e. $\widetilde{\sharp \mathfrak{l}} = I + 2$) and $\mathfrak{h}^3(\mathfrak{l})$ is defined, then the location $\mathfrak{h}^3(\mathfrak{l})$ encodes the instruction J.
(2) If a location $\mathfrak{l}$ encodes the value for the counter $\mathtt{C}_j$ in a configuration with instruction I (i.e., $\widetilde{\sharp \mathfrak{l}} \geq \alpha + 3$ and the jth ancestor of $\mathfrak{l}$ has $I + 2$ predecessors) and $\mathfrak{h}^3(\mathfrak{l})$ is defined, then $\widetilde{\sharp \mathfrak{l}} + 1 = \widetilde{\sharp \mathfrak{h}^3(\mathfrak{l})}$.

(3) Similarly, if a location $\mathfrak{l}$ encodes the value for the counter $\mathtt{C}_{3-j}$ (i.e., the counter $\mathtt{C}_{3-J}$ is not updated after instruction I) in a configuration with instruction I (i.e., $\widetilde{\sharp\mathfrak{l}} \geq \alpha+3$ and the jth ancestor of $\mathfrak{l}$ has $I+2$ predecessors) and $\mathfrak{h}^3(\mathfrak{l})$ is defined, then $\widetilde{\sharp\mathfrak{l}} = \widetilde{\sharp\mathfrak{h}^3(\mathfrak{l})}$.

The properties can be expressed by the formula φ_I below:

$$\forall\ \mathrm{u}\ (\sharp\mathrm{u}^{+3} \geq 0) \Rightarrow [\overbrace{((\sharp\mathrm{u} = I+2) \Rightarrow (\sharp\mathrm{u}^{+3} = J+2))}^{(1)} \wedge$$

$$\overbrace{((\sharp\mathrm{u} \geq \alpha+3) \wedge (\sharp\mathrm{u}^{-j} = I+2) \Rightarrow (\sharp\mathrm{u} = \sharp\mathrm{u}^{+3} - 1))}^{(2)} \wedge$$

$$\overbrace{((\sharp\mathrm{u} \geq \alpha+3) \wedge (\sharp\mathrm{u}^{j-3} = I+2) \Rightarrow (\sharp\mathrm{u} = \sharp\mathrm{u}^{+3}))}^{(3)}]$$

Each subformula decorated by a curly bracket with (i) expresses exactly the property (i) above. Note that $\sharp\mathrm{u}^{+3} = J+2$ states that the number of predecessors of $\mathfrak{h}^3(f(\mathrm{u}))$ is $J+2$, which is quite easy to express in 1SL2 (see Section 2.2). By contrast, the formula $\sharp\mathrm{u} = \sharp\mathrm{u}^{+3} - 1$ states that the number of predecessors of $\mathfrak{h}^3(f(\mathrm{u}))$ is equal to the number of predecessors of $f(\mathrm{u})$ plus one, which requires the more sophisticated formulae introduced in Section 4.3 and in Section 4.4.

Similarly, let I be the instruction "I: if $\mathtt{C}_j = 0$ then goto J_1 else ($\mathtt{C}_j := \mathtt{C}_j - 1$; goto J_2)" then φ_I is defined as follows:

$$\forall\ \mathrm{u}\ (\sharp\mathrm{u}^{+3} \geq 0) \Rightarrow [\overbrace{((\sharp\mathrm{u} = I+2) \wedge (\sharp\mathrm{u}^{+j} = \alpha+3) \Rightarrow (\sharp\mathrm{u}^{+3} = J_1+2))}^{(4)} \wedge$$

$$\overbrace{((\sharp\mathrm{u} = I+2) \wedge (\sharp\mathrm{u}^{+j} > \alpha+3) \Rightarrow (\sharp\mathrm{u}^{+3} = J_2+2))}^{(5)} \wedge$$

$$\overbrace{((\sharp\mathrm{u} > \alpha+3) \wedge (\sharp\mathrm{u}^{-j} = I+2) \Rightarrow (\sharp\mathrm{u}^{+3} = \sharp\mathrm{u} - 1))}^{(6)} \wedge$$

$$\overbrace{((\sharp\mathrm{u} = \alpha+3) \wedge (\sharp\mathrm{u}^{-j} = I+2) \Rightarrow (\sharp\mathrm{u}^{+3} = \alpha+3))}^{(7)} \wedge$$

$$\overbrace{((\sharp\mathrm{u} \geq \alpha+3) \wedge (\sharp\mathrm{u}^{j-3} = I+2) \Rightarrow (\sharp\mathrm{u} = \sharp\mathrm{u}^{+3}))}^{(8)}]$$

The subformula decorated by a curly bracket with (4) states that if a location $\mathfrak{l}$ encodes the instruction I and $\mathfrak{h}^j(\mathfrak{l})$ has $\alpha+3$ predecessors (i.e., counter $\mathtt{C}_j$ has value zero), then the location $\mathfrak{h}^3(\mathfrak{l})$ has J_1+2 predecessors (i.e., the next instruction is J_1). Similarly, the subformula decorated by a curly bracket with (5) states that if a location $\mathfrak{l}$ encodes the instruction I and $\mathfrak{h}^j(\mathfrak{l})$ has strictly more than $\alpha+3$ predecessors (i.e., counter $\mathtt{C}_j$ has non-zero value), then the location $\mathfrak{h}^3(\mathfrak{l})$ has J_2+2 predecessors (i.e., the next instruction is J_2). Moreover, the subformula decorated by a curly bracket with (6) states that if a location $\mathfrak{l}$ has at least $\alpha+3$ predecessors and its jth ancestor has $I+2$ predecessors (i.e., counter $\mathtt{C}_j$ has non-zero value and we are really dealing with instruction I), then the number of predecessors of $\mathfrak{h}^3(\mathfrak{l})$ is equal to the number of predecessors of $\mathfrak{l}$ minus one, which corresponds to encode a decrement on counter $\mathtt{C}_j$. Subformulae (7) and (8) admit a similar reading.

It is now easy to show the following lemma since we have seen that all the constraints between consecutive configurations can be encoded in 1SL2, assuming that the heap encodes a data word in $([1,\alpha] \times \mathbb{N}^2)^+$.

LEMMA 4.11. *M has a halting run iff*

$$\mathbf{dw}(\alpha, 2) \ \wedge \ \phi_{first} \ \wedge \ \phi_{last} \ \wedge \ \bigwedge_{I \in [1,\alpha]} \varphi_I$$

is satisfiable in 1SL2.

Below, we conclude by the main result of the paper.

THEOREM 4.12. *1SL2 satisfiability problem is undecidable.*

We know that if the number of quantified variables is not restricted, 1SL($-\!\!*$) is undecidable too [Brochenin et al. 2012] and recently the satisfiability problem for 1SL2($-\!\!*$) has been shown undecidable as well [Demri and Deters 2014], but this requires a far more complex proof passing via an equivalence to weak second-order logic.

4.6. Note on a variant proof using $\bigcup_{\alpha \geq 1}$ FO2$_{\alpha,1}(<, +1, =, \sim, \prec)$

As mentioned earlier, there exist many formalisms to specify properties about data words; among them can be found first-order languages. Below, we recall a few standard definitions as well as the main results. Finally, we sketch the proof of a reduction from an undecidable variant of first-order logic on data words into 1SL2. These results show interesting relationships between first-order logics on data words and separation logics.

Let us present the first-order language FO2$_{\alpha,\beta}(<, +1, =, \sim, \prec)$ to interpret data words in $([1,\alpha] \times \mathbb{N}^\beta)^+$ following developments from [Bojańczyk et al. 2011]. Most of the time, a fragment of the full language is needed, but it is helpful to provide the most general definition once and uniformly.

Let FO2$_{\alpha,\beta}(<, +1, =, \sim, \prec)$ be the set of formulae defined below:

$$\phi \ ::= \ \mathtt{a}(\mathtt{v}) \ \mid \ \mathtt{v} \sim_j \mathtt{v} \ \mid \ \mathtt{v} \prec_j \mathtt{v} \ \mid \ \mathtt{v} < \mathtt{v} \ \mid \ \mathtt{v} = 1 + (\mathtt{v}) \ \mid \ \mathtt{v} = \mathtt{v} \ \mid \ \neg\phi \ \mid \ \phi \wedge \phi \ \mid \ \exists \, \mathtt{v} \, \phi$$

with $\mathtt{v} ::= \mathtt{u}_1 \mid \mathtt{u}_2$, $j \in [1, \beta]$ and $\mathtt{a} \in [1, \alpha]$. When $\beta = 0$, this implies that there is no atomic formula using $\sim_j$ or $\prec_j$. We write FO2$_{\alpha,\beta}(<, +1, =, \sim)$ to denote the restriction of FO2$_{\alpha,\beta}(<, +1, =, \sim, \prec)$ without $\prec$. Formulae in FO2$_{\alpha,\beta}(<, +1, =, \sim, \prec)$ are interpreted over data words

$$\mathfrak{dw} = (\mathtt{a}^1, \mathfrak{d}_1^1, \ldots, \mathfrak{d}_\beta^1) \cdots (\mathtt{a}^L, \mathfrak{d}_1^L, \ldots, \mathfrak{d}_\beta^L)$$

in $([1,\alpha] \times \mathbb{N}^\beta)^+$ via the satisfaction relation $\models_f$ parameterized by $f : \{\mathtt{u}_1, \mathtt{u}_2\} \to [1, L]$ (Boolean clauses are omitted, and $i, i' \in \{1, 2\}$):

— $\mathfrak{dw} \models_f \mathtt{a}(\mathtt{u}_i) \ \overset{\text{def}}{\Leftrightarrow} \ \mathtt{a}^{f(\mathtt{u}_i)} = \mathtt{a}$,
— $\mathfrak{dw} \models_f \mathtt{u}_i \sim_j \mathtt{u}_{i'} \ \overset{\text{def}}{\Leftrightarrow} \ \mathfrak{d}_j^{f(\mathtt{u}_i)} = \mathfrak{d}_j^{f(\mathtt{u}_{i'})}$,
— $\mathfrak{dw} \models_f \mathtt{u}_i \prec_j \mathtt{u}_{i'} \ \overset{\text{def}}{\Leftrightarrow} \ \mathfrak{d}_j^{f(\mathtt{u}_i)} < \mathfrak{d}_j^{f(\mathtt{u}_{i'})}$,
— $\mathfrak{dw} \models_f \mathtt{u}_i = \mathtt{u}_{i'} \ \overset{\text{def}}{\Leftrightarrow} \ f(\mathtt{u}_i) = f(\mathtt{u}_{i'})$,
— $\mathfrak{dw} \models_f \mathtt{u}_i = 1 + (\mathtt{u}_{i'}) \ \overset{\text{def}}{\Leftrightarrow} \ f(\mathtt{u}_i) = f(\mathtt{u}_{i'}) + 1$,

— $\mathfrak{dw} \models_f \mathtt{u}_i < \mathtt{u}_{i'} \overset{\text{def}}{\Leftrightarrow} f(\mathtt{u}_i) < f(\mathtt{u}_{i'})$,
— $\mathfrak{dw} \models_f \exists\, \mathtt{u}_i\ \phi \overset{\text{def}}{\Leftrightarrow}$ there is $p \in [1, L]$ such that $\mathfrak{dw} \models_{f[\mathtt{u}_i \mapsto p]} \phi$.

A sentence ϕ in $\mathrm{FO2}_{\alpha,\beta}(<, +1, =, \sim, \prec)$ is satisfiable $\overset{\text{def}}{\Leftrightarrow}$ there is a data word $\mathfrak{dw}$ in $([1,\alpha] \times \mathbb{N}^{\beta})^{+}$ such that $\mathfrak{dw} \models \phi$ (no need to specify a variable assignment since ϕ is closed).

Let us recall major results about FO2 on data words; $\mathrm{FO2}_{\alpha,0}(<, +1, =)$ was introduced in Section 2.3 and the others just above.

THEOREM 4.13.

(I) The satisfiability problem for $\bigcup_{\alpha \geq 1} \mathrm{FO2}_{\alpha,0}(<, +1, =)$ is NEXPTIME*-complete [Etessami et al. 1997] (see also [Weis 2011, Corollary 2.2.4]).*

(II) The satisfiability problem for $\bigcup_{\alpha \geq 1} \mathrm{FO2}_{\alpha,1}(<, +1, =, \sim)$ is decidable and closely related to the reachability problem for Petri nets [Bojańczyk et al. 2011; David 2009, Theorem 3].

(III) The satisfiability problem for $\bigcup_{\alpha \geq 1} \mathrm{FO2}_{\alpha,2}(<, +1, =, \sim)$ is undecidable [Bojańczyk et al. 2011; David 2009, Proposition 27].

(IV) The satisfiability problem for $\bigcup_{\alpha \geq 1} \mathrm{FO2}_{\alpha,1}(<, +1, =, \sim, \prec)$ is undecidable [Bojańczyk et al. 2011; David 2009].

Theorem 4.13(IV) shall be used in this section but decidability can be regained, as shown in [Schwentick and Zeume 2012], where finite satisfiability of FO2 over data words with a linear order on the positions and a linear order and a corresponding successor relation on the data values shown in EXPSPACE [Schwentick and Zeume 2012].

A slightly simpler undecidability proof can be also obtained from the undecidability of the satisfiability problem for $\bigcup_{\alpha \geq 1} \mathrm{FO2}_{\alpha,1}(<, +1, =, \sim, \prec)$ on data words [Bojańczyk et al. 2011] (see Theorem 4.13(IV)). In order to obtain a reduction from the halting problem for Minsky machines, we had to deal with encoding of instructions, which is a bit tedious in some places but the gain has been to obtain a *master reduction*. Master reductions are understood as reductions from decision problems involving, for instance, Turing machines, Minsky machines or any other standard class of computational models. These reductions are always preferred because no intermediate decision problems are involved and therefore this limits the sources of flaws for example. Nevertheless, the reduction from the satisfiability problem $\bigcup_{\alpha \geq 1} \mathrm{FO2}_{\alpha,1}(<, +1, =, \sim, \prec)$ is a bit simpler if undecidability has to be explained in the most concise way.

Let us briefly provide the main ingredients for such a proof. We define a logarithmic-space translation t as follows. A position $\mathtt{u}$ in the data word corresponds to a location on the main path of the fishbone encoding the same position but for the (unique) part related to the (unique) datum. In the translation process, we freely use macros defined earlier ($i, j \in \{1, 2\}$).

— t is homomorphic for Boolean connectives,
— $t(\mathtt{u}_i = \mathtt{u}_j) \overset{\text{def}}{=} \mathtt{u}_i = \mathtt{u}_j$,
— $t(\mathtt{u}_i < \mathtt{u}_j) \overset{\text{def}}{=} \mathtt{ls}(\mathtt{u}_i, \mathtt{u}_j) \wedge \mathtt{u}_i \neq \mathtt{u}_j$,
— $t(\mathtt{u}_j = 1 + (\mathtt{u}_i)) \overset{\text{def}}{=} \top * (\mathtt{ls}'(\mathtt{u}_i, \mathtt{u}_j) \wedge (\sharp \mathtt{u}_i^{+3} = 1) \wedge \neg(\sharp \mathtt{u}_i^{+4} \geq 0))$,
— $t(\mathtt{a}(\mathtt{u}_i)) \overset{\text{def}}{=} \exists\, \mathtt{u}_{3-i}\ (\mathtt{u}_{3-i} \hookrightarrow \mathtt{u}_i) \wedge (\sharp \mathtt{u}_{3-i} = \mathtt{a} + 2)$,
— $t(\mathtt{u}_i \sim_1 \mathtt{u}_j) \overset{\text{def}}{=} \sharp \mathtt{u}_i = \sharp \mathtt{u}_j$,
— $t(\mathtt{u}_i \prec_1 \mathtt{u}_j) \overset{\text{def}}{=} \sharp \mathtt{u}_i + 1 \leq \sharp \mathtt{u}_j$,

— $t(\exists\, \mathrm{u}_i\ \phi) \overset{\text{def}}{=} \exists\, \mathrm{u}_i\ (\sharp \mathrm{u}_i \geq \alpha + 3) \wedge t(\phi)$.

LEMMA 4.14. *Let ϕ be a formula in FO2$_{\alpha,1}(<, +1, =, \sim, \prec)$.*

(I) For every data word $\mathfrak{dw}$ in $([1,\alpha] \times \mathbb{N})^+$, $\mathfrak{dw} \models \phi$ iff $\mathfrak{h}_{\mathfrak{dw}} \models \mathbf{dw}(\alpha, 1) \wedge t(\phi)$.
(II) Let $\mathfrak{h}$ be a heap such that $\mathfrak{h} \models \mathbf{dw}(\alpha, 1) \wedge t(\phi)$, then there is a data word $\mathfrak{dw}$ in $([1,\alpha] \times \mathbb{N})^+$ such that $\mathfrak{h}$ and $\mathfrak{h}_{\mathfrak{dw}}$ are isomorphic and $\mathfrak{dw} \models \phi$.

As a corollary, 1SL2 is undecidable, since for any ϕ in FO2$_{\alpha,1}(<, +1, =, \sim, \prec)$, ϕ is satisfiable iff $\mathbf{dw}(\alpha, 1) \wedge t(\phi)$ is satisfiable in 1SL2. Note that the gain with the proof sketched above is not that significant; however, we do not need the material from Section 4.4. (Still, though, we need to express arithmetical constraints between the numbers of predecessors.) Furthermore, we need to formally prove Lemma 4.14, which is of a complexity comparable to the material in Section 4.5. Observe also that Section 4.4 is interesting in its own right because it establishes a result about the expressive power of 1SL2 by performing surgical cuts.

5. CONCLUSION

In the paper, we have shown that two-variable first-order separation logic (1SL2) is undecidable by designing a simple master reduction from the halting problem for Minsky machines and, if we drop the magic wand operator (a fragment called 1SL2($*$)), we get decidability but with non-elementary complexity. Our contribution is related to the hardness results when only two variables are used (no program variables, only two quantified variables, no unary predicate symbols). Heavy recycling of variables is done, following the classical result for modal logic [Gabbay 1981]. In order to get these results, we have shown a simple encoding of data words with multiple attributes that is used both for the undecidability and for the non-elementarity result. This nice relationship with data logics [Bojańczyk et al. 2011] is also complemented by the fact that we have shown how to encode propositional interval temporal logic (PITL) [Moszkowski 2004] into 1SL2($*$). Hence, we believe we have established promising bridges between data logic(s), interval temporal logic(s), and separation logic(s), apart from providing additional evidence of the importance of intervals in logics for computer science. Moreover, we have introduced a separation modal logic MLH. While its decidability status remains open, we have proved that its restriction to separating conjunction is decidable with non-elementary complexity. This has been not only an opportunity to better understand the expressive power of separation logic fragments but also to significantly populate 1SL2's inner circle (see Figure 1 for an overview).

ACKNOWLEDGMENT

We would like to thank Emanuel Kieroński, Ben Moszkowski, and Guido Sciavicco. for insightful exchanges about interval temporal logics and two-variable first-order logic(s). Furthermore, we are indebted to two anonymous referees for many helpful and insightful suggestions that helped us to improve the quality of this manuscript.

References

J. Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26, 11 (1983), 832–843.

K. Bansal, R. Brochenin, and E. Lozes. 2009. Beyond Shapes: Lists with Ordered Data. In *FOSSACS'09 (Lecture Notes in Computer Science)*, Vol. 5504. Springer, 425–439.

S. Benaim, M. Benedikt, W. Charatonik, E. Kieroński, R. Lenhardt, F. Mazowiecki, and J. Worrell. 2013. Complexity of two-variable logic over finite trees. In *ICALP'13 (Lecture Notes in Computer Science)*, Vol. 7966. Springer, 74–88.

P. Blackburn, M. de Rijke, and Y. Venema. 2001. *Modal Logic*. Cambridge University Press.

M. Bojańczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. 2011. Two-variable logic on data words. *ACM Transactions on Computational Logic* 12, 4 (2011), 27.

E. Börger, E. Grädel, and Y. Gurevich. 1997. *The Classical Decision Problem*. Springer.

P. Bouyer. 2002. A Logical Characterization of Data Languages. *Inform. Process. Lett.* 84, 2 (2002), 75–85.

R. Brochenin, S. Demri, and E. Lozes. 2012. On the Almighty Wand. *Information and Computation* 211 (2012), 106–137.

J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Perez. 2014. A Decision Procedure for Satisfiability in Separation Logic with Inductive Predicates. In *CSL-LICS'14*. ACM.

J. Brotherston and M. Kanovich. 2014. Undecidability of Propositional Separation Logic and Its Neighbours. *Journal of the Association for Computing Machinery* 61, 2 (2014).

C. Calcagno, P. O'Hearn, and H. Yang. 2001. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *FSTTCS'01 (Lecture Notes in Computer Science)*, Vol. 2245. Springer, 108–119.

W. Charatonik, E. Kieroński, and F. Mazowiecki. 2014. Decidability of Weak Logics with Deterministic Transitive Closure. In *CSL-LICS'14*. ACM.

B. Cook, Ch. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. 2011. Tractable Reasoning in a Fragment of Separation Logic. In *CONCUR'11 (Lecture Notes in Computer Science)*. Springer, 235–249.

J.-R. Courtault and D. Galmiche. 2013. A Modal BI Logic for Dynamic Resource Properties. In *LFCS'13 (Lecture Notes in Computer Science)*, Vol. 7734. Springer, 134–148.

C. David. 2009. *Analyse de XML avec données non-bornées*. Ph.D. Dissertation. LIAFA, Université Paris VII.

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

M. de Rijke. 1992. The modal logic of inequality. *The Journal of Symbolic Logic* 57, 2 (1992), 566–584.

N. Decker, P. Habermehl, M. Leucker, and D. Thoma. 2014. Ordered Navigation on Multi-attributed Data Words. In *CONCUR'14 (Lecture Notes in Computer Science)*, Vol. 8704. 497–511.

A. Degtyarev, M. Fisher, and A. Lisitsa. 2002. Equality and monodic first-order temporal logic. *Studia Logica* 72, 2 (2002), 147–156.

S. Demri and M. Deters. 2014. Expressive completeness of separation logic with two variables and no separating conjunction. In *CSL-LICS'14*. ACM, 37.

S. Demri, D. Galmiche, D. Larchey-Wendling, and D. Mery. 2014. Separation Logic with One Quantified Variable. In *CSR'14 (Lecture Notes in Computer Science)*, Vol. 8476. Springer, 125–138.

S. Demri and Ph. Schnoebelen. 2002. The complexity of Propositional Linear Temporal Logics in Simple Cases. *Information and Computation* 174, 1 (2002), 84–103.

K. Etessami, M. Vardi, and Th. Wilke. 1997. First-Order Logic with Two variables and Unary Temporal logics. In *LICS'97*. IEEE, 228–235.

D. Figueira. 2010. *Reasoning on words and trees with data*. Ph.D. Dissertation. ENS de Cachan.

D. Gabbay. 1981. Expressive Functional Completeness in Tense Logic. In *Aspects of Philosophical Logic*. Reidel, 91–117.

D. Gabbay and V. Shehtman. 1993. Undecidability of modal and intermediate first-order logics with two individual variables. *The Journal of Symbolic Logic* 58, 3 (1993), 800–823.

E. Grädel, Ph. Kolaitis, and M. Vardi. 1997a. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic* 3, 1 (1997), 53–69.

E. Grädel, M. Otto, and E. Rosen. 1997b. Two-Variable logic with counting is Decidable. In *LICS'97*. IEEE, 306–317.

E. Grädel, M. Otto, and E. Rosen. 1999. Undecidability Results on Two-Variable Logics. *Archive for Mathematical Logic* 38, 4–5 (1999), 313–354.

J. Halpern. 1995. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence* 75, 2 (1995), 361–372.

D. Harel, D. Kozen, and J. Tiuryn. 2000. *Dynamic Logic*. MIT Press.

M. Hennessy and R. Milner. 1980. On observing nondeterminism and concurrency.. In *ICALP'80 (Lecture Notes in Computer Science)*, Vol. 85. Springer, 299–309.

R. Iosif, A. Rogalewicz, and J. Simacek. 2013. The Tree Width of Separation Logic with Recursive Definitions. In *CADE'13 (Lecture Notes in Computer Science)*, Vol. 7898. Springer, 21–38.

A. S. Kahr, E. F. Moore, and H. Wang. 1962. Entscheidungsproblem reduced to the $\forall \exists \forall$ case. *Proc. Nat. Acad. Sci. U. S. A.* 48, 3 (1962), 365–377.

E. Kieroński, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. 2012. Two-Variable First-Order Logic with Equivalence Closure. In *LICS'12*. IEEE, 431–440.

D. Larchey-Wendling and D. Galmiche. 2013. Nondeterministic Phase Semantics and the Undecidability of Boolean BI. *ACM Transactions on Computational Logic* 14, 1 (2013).

H. Lewis. 1980. Complexity Results for classes of quantificational formulas. *J. Comput. System Sci.* 21 (1980), 317–353.

C. Lutz and U. Sattler. 2002. The complexity of reasoning with Boolean Modal Logics. In *Advances in Modal Logics 2000, Volume 3*. World Scientific, 329–348.

M. L. Minsky. 1967. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Ch. Morgan. 1976. Methods for automated theorem proving in non classical logics. *IEEE Trans. Comput.* 25, 8 (1976), 852–862.

M. Mortimer. 1975. On language with two variables. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 21 (1975), 135–140.

B. Moszkowski. 1983. *Reasoning about digital circuits*. Technical Report STAN–CS–83–970. Dept. of Computer Science, Stanford University, Stanford, CA.

B. Moszkowski. 2004. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics* 14, 1–2 (2004), 55–104.

M. Otto. 2001. Two Variable First-Order Logic over Ordered Domains. *The Journal of Symbolic Logic* 66, 2 (2001), 685–702.

L. Pacholski, W. Szwast, and L. Tendera. 1997. Complexity of Two-Variable logic with counting. In *LICS'97*. IEEE, 318–327.

A. Prior. 1967. *Past, Present and Future*. Oxford University Press.

M. Rabin. 1969. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.* 41 (1969), 1–35.

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

Th. Schwentick and Th. Zeume. 2012. Two-Variable Logic and Two Order Relations. *Logical Methods in Computer Science* 8 (2012), 1–27.

D. Scott. 1962. A decision method for validity of sentences in two variables. *The Journal of Symbolic Logic* 27 (1962), 377.

L. Stockmeyer. 1974. *The complexity of decision problems in automata theory and logic*. Ph.D. Dissertation. Department of Electrical Engineering, MIT.

W. Szwast and L. Tendera. 2013. FOˆ2 with one transitive relation is decidable. In *STACS'13 (LIPIcs)*, Vol. 20. 317–328.

J. van Benthem. 1976. *Correspondence Theory*. Ph.D. Dissertation. Mathematical Institute, University of Amsterdam.

Ph. Weis. 2011. *Expressiveness and Succinctness of First-Order Logic on Finite Words*. Ph.D. Dissertation. University of Massachussetts.

H. Yang. 2001. *Local Reasoning for Stateful Programs*. Ph.D. Dissertation. University of Illinois, Urbana-Champaign.

Zhou Chaochen. 2008. In *Logics of Specification Languages*, D. Bjorner and M. Henson (Eds.). Springer, Chapter Reviews on "Duration Calculus", 609–611. Monographs in Theoretical Computer Science. An EATCS Series.