Separation Logic with One Quantified Variable
Stephane Demri, Didier Galmiche, Dominique Larchey-Wendling, Daniel Mery

To cite this version:

HAL Id: hal-01258821
https://hal.archives-ouvertes.fr/hal-01258821
Submitted on 19 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives| 4.0 International License
We investigate first-order separation logic with one record field restricted to a unique quantified variable (1SL1). Undecidability is known when the number of quantified variables is unbounded and the satisfiability problem is PSPACE-complete for the propositional fragment. We show that the satisfiability problem for 1SL1 is PSPACE-complete and we characterize its expressive power by showing that every formula is equivalent to a Boolean combination of atomic properties. This contributes to our understanding of fragments of first-order separation logic that can specify properties about the memory heap.
of programs with singly-linked lists. All the fragments we consider contain the magic wand operator and first-order quantification over a single variable.

**Keywords** Separation logic · Quantifier elimination · Model Checking · Satisfiability · Computational Complexity

1 Introduction

1.1 Separation Logic for Verifying Programs with Pointers

Separation logic (Reynolds 2002) is a well-known logic for analysing programs with pointers stemming from BI logic (Ishtiaq and O’Hearn 2001). Such programs have specific errors to be detected and separation logic is used as an assertion language for Hoare-like proof systems (Reynolds 2002) that are dedicated to verify programs manipulating heaps. Any procedure mechanizing the proof search requires subroutines that check the satisfiability or the validity of formulæ (more precisely entailment) from the assertion language. That is why, characterizing the computational complexity of separation logic and its fragments and designing optimal decision procedures remain essential tasks. Separation logic contains a structural separating connective and its adjoint (the separating implication, also known as the magic wand).

Concise and modular proofs can be derived using these connectives, since they can express properties such as non-aliasing (Reynolds 2002).

The main concern of the paper is to study a non-trivial fragment of first-order separation logic with one record field as far as expressive power, decidability and complexity are concerned. Herein, the models of separation logic are pairs made of a variable valuation (store) and a partial function with finite domain (heap), also known as memory states.

1.2 Decidability and Complexity Results for Separation Logic

The complexity of satisfiability and model-checking problems for separation logic fragments have been quite studied (Calcagno et al 2001; Reynolds 2002; Cook et al 2011) (see also new decidability results in (Iosif et al 2013) or undecidability results in (Brotherston and Kanovich 2010; Larchey-Wendling and Galmiche 2010) in an alternative setting). Separation logic is equivalent to a Boolean propositional logic (Lozes 2004a,b) if first-order quantifiers are disabled. Separation logic without first-order quantifiers is decidable, but it becomes undecidable with first-order quantifiers (Calcagno et al 2001). For instance, model-checking and satisfiability for propositional separation logic are PSPACE-complete problems (Calcagno et al 2001). Decidable fragments with first-order quantifiers can be found in (Galmiche and Méry 2010; Brochenin et al 2012).

In order to study decidability or complexity issues for separation logic, two tracks have been observed in the literature. There is the verification approach
with decision procedures for fragments of practical use, see e.g. (Berdine et al 2005; Cook et al 2011; Haase et al 2013). Alternatively, fragments, extensions or variants of separation logic are considered from a logical viewpoint, see e.g. (Calcagno et al 2001; Brotherston and Kanovich 2010; Larchey-Wendling and Galmiche 2010).

1.3 Our Contributions

In this paper, we study first-order separation logic with one quantified variable, with an unbounded number of program variables and with one record field (herein called 1SL1).

1. We introduce test formulæ that state simple properties about the memory states and we show that every formula in 1SL1 is equivalent to a Boolean combination of test formulæ, extending what was done in (Lozes 2004b; Brochenin et al 2009) for the propositional case. For instance, separating connectives can be eliminated in a controlled way as well as first-order quantification over the single variable. In that way, we show a quantifier elimination property similar to the one for Presburger arithmetic (in that case, the test formulæ are linear and periodicity constraints). This result extends previous ones on propositional separation logic (Lozes 2004a,b; Brochenin et al 2009) and as far as we know, this is the first time that this approach is extended to a first-order version of separation logic with the magic wand operator. However, it is the best we can hope for since 1SL with two quantified variables and no program variables (1SL2) has been recently shown undecidable in (Demri and Deters 2014). Of course, other extensions of 1SL1 could be considered, for instance to add a bit of arithmetical constraints, but herein we focus on 1SL1 that is theoretically nicely designed, even though it is still unclear how much 1SL1 is useful for formal verification. We also establish that the satisfiability problem for Boolean combinations of test formulæ is \( \text{NP} \)-complete thanks to a saturation algorithm for the theory of memory states with test formulæ, paving the way to use SMT solvers to decide 1SL1; see e.g. the use of such solvers in (Piskac et al 2013).

By way of comparison with first-order predicate logic, propositional calculus is \( \text{NP} \)-complete and FO1 (first-order logic with one variable) is \( \text{NP} \)-complete too. We knew that propositional separation logic is \( \text{PSPACE} \)-complete and herein we also establish that 1SL1 satisfiability has the same worst-case complexity.

2. Even though Boolean combinations of test formulæ and 1SL1 have identical expressive power, we obtain \( \text{PSPACE} \)-completeness for model-checking and satisfiability in 1SL1. The conciseness of 1SL1 explains the difference between these two complexities.

We show that the satisfiability problem for Boolean combinations of test formulæ is \( \text{NP} \)-complete whereas we establish that model-checking and satisfiability problems for 1SL1 are \( \text{PSPACE} \)-complete. The difference between
these two complexities here is due to the conciseness of 1SL1. PSpace-completeness is still a relatively low complexity but this result can be extended with more than one record field (but still with one quantified variable). This is the best we can hope for with one quantified variable and with the magic wand, that is notoriously known to easily increase computational complexity.

3. Moreover, we provide a simple algorithm to compute from a formula in 1SL1, an equivalent Boolean combination of test formulæ.

Structure of the paper. Section 2 is mainly dedicated to preliminary definitions about separation logic, basic properties that can be expressed in 1SL1 and several preliminary definitions and results about partitions on memory states. In Section 3, test formulæ are introduced as well as several relations between locations based on such test formulæ. The second part of this section contains a series of technical lemmas that are useful to establish the correctness of the abstraction based on test formulæ, which is shown in Section 4. Decidability of 1SL1 satisfiability problem as well as admissibility of quantifiers are established in Section 4 too. Section 4 concludes by showing that the satisfiability and model-checking problems can be solved in polynomial space. In Section 5, we prove a result stated earlier in the paper, namely the satisfiability status of conjunctions of literals made of test formulæ can be decided in polynomial time. Strictly speaking, this result is not used to establish complexity results about 1SL1 (and that is why, its proof has been postponed a bit) but it paves the way to decide 1SL1 with SMT solvers, this is at least our hope. Whereas Section 6 contains concluding remarks, a technical appendix concludes the paper and it contains the proofs that are not present in the main body of the paper.

2 Separation Logic 1SL and its Heap Memory Model

2.1 First-Order Separation Logic with One Selector 1SL

Let $\text{PVAR}$ be a countably infinite set of program variables and $\text{FVAR}$ be a countably infinite set of quantified variables. We write $x, y, \ldots, x_1, x_2, \ldots$ to denote program variables and $u, u_1, u_2, \ldots$ to denote quantified variables. A memory state (also called a model) is a pair $(s, h)$ such that

- $s$ is a variable valuation of the form $s : \text{PVAR} \rightarrow \mathbb{N}$ (the store),
- $h$ is a partial function $h : \mathbb{N} \rightarrow \mathbb{N}$ with finite domain (the heap) and we write $\text{dom}(h)$ to denote its domain and $\text{ran}(h)$ to denote its range.

Two heaps $h_1$ and $h_2$ are said to be disjoint, noted $h_1 \perp h_2$, if their domains are disjoint; when this holds, we write $h_1 \uplus h_2$ to denote the heap corresponding to the disjoint union of the graphs of $h_1$ and $h_2$, hence $\text{dom}(h_1 \uplus h_2) = \text{dom}(h_1) \uplus \text{dom}(h_2)$. When the domains of $h_1$ and $h_2$ are not disjoint, the composition $h_1 \uplus h_2$ is not defined even if $h_1$ and $h_2$ have the same values on $\text{dom}(h_1) \cap \text{dom}(h_2)$.
The empty heap, denoted $\square$, is the only heap with an empty domain. It is a neutral element for $\sqcup$: we have $h \sqcup \square = \square \sqcup h = h$ for any heap $h$. The heap $h_1$ is a subheap of $h_2$, noted $h_1 \subseteq h_2$, if $\text{dom}(h_1) \subseteq \text{dom}(h_2)$ and $h_1(l) = h_2(l)$ for any $l \in \text{dom}(h_1)$. Alternatively, we say that the heap $h_2$ is an extension of the heap $h_1$. Obviously, $h_1 \subseteq h_1 \sqcup h_2$ whenever the composition is defined. Moreover, any extension of the heap $h_1$ has the form $h_1 \sqcup h_2$ for some heap $h_2$. A heap is atomic if its domain is a singleton set. We write $[l_1 \mapsto l_2]$ for the unique atomic heap $h$ such that $\text{dom}(h) = \{l_1\}$ and $h(l_1) = l_2$.

Formulæ in ISL are built from expressions (composed of either program or quantified variables) and atomic formulæ (either equality tests or points-to). Formulæ in ISL are closed under Boolean connectives, first-order quantification (as in first-order classical logic) but also under separating conjunction $\ast$, its unit $\text{emp}$, and the separating implication $\rightarrow$ usually called the magic wand.

**Definition 2.1** The 1SL formulæ are defined by the following grammar:

$$
\begin{align*}
\text{e} &::= x \mid u \quad \text{where } x \in \text{FVAR}, u \in \text{FVAR} \\
\pi &::= \text{e} = \text{e}' \mid \text{e} \mapsto \text{e}' \\
A &::= \pi \mid \bot \mid \text{emp} \mid A \land B \mid \neg A \mid A \ast B \mid A \rightarrow B \mid \exists u A \quad \text{where } u \in \text{FVAR}
\end{align*}
$$

We make use of standard notations for the derived connectives of classical logic. The size of a formula $A$, written $|A|$, is defined as the number of symbols required to write it. An assignment is a map $f : \text{FVAR} \rightarrow \mathbb{N}$. The satisfaction relation $\models$ is parameterized by assignments (clauses for Boolean connectives are omitted):

- $(s, h) \models_{\pi} e = e'$ iff $[e] = [e']$ where $[x] \triangleq s(x)$ and $[u] \triangleq f(u)$.
- $(s, h) \models_{\pi} e \mapsto e'$ iff $[e] \in \text{dom}(h)$ and $h([e]) = [e']$.
- $(s, h) \models_{\pi} \text{emp}$ iff $h = \square$.
- $(s, h) \models_{\pi} A_1 \ast A_2$ iff $h = h_1 \sqcup h_2$, $(s, h_1) \models_{\pi} A_1$, $(s, h_2) \models_{\pi} A_2$ for some $h_1, h_2$.
- $(s, h) \models_{\pi} A_1 \rightarrow A_2$ iff for all $h'$, if $h \sqcup h'$ and $(s, h') \models_{\pi} A_1$ then $(s, h \sqcup h') \models_{\pi} A_2$.
- $(s, h) \models_{\pi} \exists u A$ iff there is $l \in \mathbb{N}$ such that $(s, h) \models_{(u \mapsto l)} A$ where $f[u \mapsto l]$ is the assignment equal to $f$ except that $u$ takes the value $l$.

Whereas $\exists u$ is clearly a first-order quantifier, the connectives $\ast$ and $\rightarrow$ are known to be second-order quantifiers. In the paper, we show how to eliminate these three connectives when only one quantified variable is used. The logic 1SL is not minimal for its expressive power; e.g. $\text{emp}$ is logically equivalent to $\forall u \neg((u \mapsto u) \mapsto \bot)$.

**Proposition 2.2** Let $s, s' : \text{FVAR} \rightarrow \mathbb{N}$ be two stores, $h : \mathbb{N} \rightarrow \mathbb{N}$ be a heap, $f, f' : \text{FVAR} \rightarrow \mathbb{N}$ be two assignments and let $A$ be an ISL formulæ. If $s(x) = s'(x)$ holds for every program variable $x$ that occurs in $A$ and $f(u) = f'(u)$ holds for any quantified variable $u$ that occurs freely in $A$ then the equivalence $(s, h) \models_{f} A \iff (s', h) \models_{f'} A$ holds.
The proof is by an easy verification and it is left to the reader.

As a consequence of Proposition 2.2, we might abusively use the notation \((s, h) \models_A \varphi\) when the “store” \(s : \{x_1, \ldots, x_q\} \to \mathbb{N}\) is only defined on a superset of the program variables that occur in \(A\), because the interpretation of the program variables that do not occur in \(A\) does not matter.

**Proposition 2.3** Let \(\varphi : \mathbb{N} \to \mathbb{N}\) be permutation on locations, i.e. a one-to-one map. For any 1SL formula \(A\), any memory state \((s, h)\) and any assignment \(f : \text{FVAR} \to \mathbb{N}\), we have \((s, h) \models_A \varphi\) iff \((\varphi \circ s, \varphi \circ h \circ \varphi^{-1}) \models_{\varphi f} A\).

In the above statement, it is worth noting that the domain of \(\varphi \circ h \circ \varphi^{-1}\) is \(\varphi(\text{dom}(h))\). The proof of Proposition 2.3 is left to the reader.

Since models for a formula \(A\) are closed under permutations on locations, without any loss of generality, we can assume that for every variable \(x\) in \(A\) being either a program variable or a freely occurring quantified variable \(s(x) \leq m\) where \(A\) contains at most \(m\) program or free variables.

**Corollary 2.4** Let \(A\) be an 1SL formula, \(V \subseteq \text{FVAR}\) and \(F \subseteq \text{FVAR}\) be finite subsets such that \(V\) contains the program variables that occur in \(A\) and \(F\) contains the quantified variables that occur freely in \(A\). If there exist a memory state \((s, h)\) and an assignment \(f : \text{FVAR} \to \mathbb{N}\) such that \((s, h) \models_A A\), then there exist a memory state \((s', h')\) and an assignment \(f' : \text{FVAR} \to \mathbb{N}\) such that \((s', h') \models_{f'} A\) and \(s'(V) \cup f'(F) \subseteq \{0, 1, \ldots, m-1\}\) with \(m = \text{card}(V) + \text{card}(F)\).

**Proof** The set \(s(V) \cup f(F)\) is a finite subset of \(\mathbb{N}\). Its cardinal is less than \(m = \text{card}(V) + \text{card}(F)\). Hence, there exists a permutation on locations \(\varphi : \mathbb{N} \to \mathbb{N}\) such that \(\varphi(s(V) \cup f(F)) \subseteq \{0, 1, \ldots, m-1\}\). We apply Proposition 2.3 and get \(s' = \varphi \circ s\), \(h' = \varphi \circ h \circ \varphi^{-1}\) and \(f' = \varphi \circ f\). \(\square\)

We write 1SL0 to denote the propositional fragment of 1SL where no variable from \(\text{FVAR}\) occurs. Similarly, we write 1SL1 to denote the fragment of 1SL restricted to a single quantified variable, say \(u\). In that case, the satisfaction relation can be denoted by \(\models_{l}\) where \(l\) is understood as the value \(l = f(u)\) of the quantified variable \(u\) under the assignment \(f\).

Let \(L\) be a logic among 1SL, 1SL1 and 1SL0. As usual, the satisfiability problem for \(L\) takes as input a formula \(A\) in \(L\) and asks whether there is a memory state \((s, h)\) and an assignment \(f\) such that \((s, h) \models_l A\). The model-checking problem for \(L\) takes as input a formula \(A\) in \(L\), a memory state \((s, h)\) and an assignment \(f\) for free variables from \(A\) and asks whether \((s, h) \models_f A\).

When checking the satisfiability status of a formula \(A\) in 1SL1, we assume that its program variables are contained in \(\{x_1, \ldots, x_q\}\) for some \(q \geq 1\) and the quantified variable is \(u\). So, \(\text{FVAR}\) is unbounded but as usual, when dealing with a specific formula, the set of program variables is finite.

**Theorem 2.5** (Calcagno et al 2001; Brochenin et al 2012; Demri and Deters 2014) The satisfiability and model-checking problems for 1SL0 are \(\text{PSPACE}\)-complete. The satisfiability problem for 1SL is undecidable, even when restricted to two quantified variables.
2.2 A First Glimpse of Properties that can be Stated in 1SL1

The logic 1SL1 allows to express different types of properties on memory states. The examples below indeed illustrate the expressivity of 1SL1 and in the paper we characterize precisely what can be expressed in 1SL1:

- the variable $e$ is allocated in the heap:
  \[ \text{alloc}(e) \overset{def}{=} (e \mapsto e) \not\downarrow \]

  where $e$ is either $x_i$ or $u$. Intuitively: it is not possible to add a loop at $e$;

- the program variables $x_i$ and $x_j$ have the same value (without explicit equality predicate and with only one quantified variable):
  \[ \forall u \ (u \mapsto x_i) \not\rightarrow (u \mapsto x_j) \]

- the variable $x_i$ points to a location that is a loop:
  \[ \text{toloop}(x_i) \overset{def}{=} \exists u \ (x_i \mapsto u \land u \mapsto u) \]

- the variable $x_i$ points to a location that is allocated:
  \[ \text{toalloc}(x_i) \overset{def}{=} \exists u \ (x_i \mapsto u \land \text{alloc}(u)) \]

- the variables $x_i$ and $x_j$ point to a shared location:
  \[ \text{conv}(x_i, x_j) \overset{def}{=} \exists u \ (x_i \mapsto u \land x_j \mapsto u) \]

- there is a location between $x_i$ and $x_j$:
  \[ \text{btwn}(x_i, x_j) \overset{def}{=} \exists u \ (x_i \mapsto u \land u \mapsto x_j) \]

- the domain of the heap has at least $k$ elements:
  \[ \# \text{dom} \geq k \overset{def}{=} \neg \text{emp} \cdots \neg \text{emp} \]

where $\neg \text{emp}$ occurs $k$ times (in the example, $k = 3$ on the right-hand side of the equality symbol);

- the memory heap has exactly one memory cell at address $x_i$ (expressed in 1SL0):
  \[ \text{atomic}(x_i) \overset{def}{=} \text{alloc}(x_i) \land \neg(\# \text{dom} \geq 2) \]

- the location interpreted by $x_i$ has no predecessor:
  \[ \# \text{pred}(x_i) = 0 \overset{def}{=} \neg(\exists u \ u \mapsto x_i) \]

- the location interpreted by $x_i$ has exactly one predecessor:
  \[ \# \text{pred}(x_i) = 1 \overset{def}{=} (\exists u \ u \mapsto x_i) \land \neg(\exists u \ u \mapsto x_i \land \exists u \ u \mapsto x_i) \]
the location interpreted by $x_i$ has exactly $k > 0$ predecessors:

$$\# \text{pred}(x_i) = k \overset{\text{df}}{=} (# \text{pred}(x_j) = 1) \ast \cdots \ast (# \text{pred}(x_j) = 1)$$

where $\# \text{pred}(x_j) = 1$ occurs $k$ times;

the heap contains at least three self-loops:

$$\# \text{loop} \geq 3 \overset{\text{df}}{=} (\exists u u \mapsto u) \ast (\exists u u \mapsto u) \ast (\exists u u \mapsto u)$$

We also illustrate briefly the expressive power of 1SL2, i.e. with two quantified variables. But be aware that 1SL2 is strictly more expressive than 1SL1; see for instance Corollary 4.12. Moreover, 1SL2 is proved non recursively enumerable in (Demri and Deters 2014) whereas in this paper we show that 1SL1 is PSPACE-complete. In 1SL2, it is possible to express the existence of paths/lists within memory states:

the heap is composed of exactly a path of strictly positive length from $x_i$ to $x_j$ together with an arbitrary number of cycles, written $\text{ls}'(x_i, x_j)$:

$$\# \text{pred}(x_i) = 0 \land \text{alloc}(x_i)$$
$$\land \# \text{pred}(x_j) = 1 \land \neg \text{alloc}(x_j)$$
$$\land \forall u_1 (\# \text{pred}(u_1) = 0 \land \text{alloc}(u_1)) \Rightarrow u_1 = x_i$$
$$\land \forall u_1 (\# \text{pred}(u_1) \neq 0 \land u_1 \neq x_j) \Rightarrow (\# \text{pred}(u_1) = 1 \land \text{alloc}(u_1))$$

there is a path from $x_i$ to $x_j$ can be expressed by

$$\text{ls}'(x_i, x_j) \overset{\text{df}}{=} (x_i = x_j) \lor (\text{ls}'(x_i, x_j) \ast \top)$$

to be found originally in (Brochenin et al 2012, Lemma 2.4); a similar property was established for graph logics in (Dawar et al 2007).

2.3 Decomposition and Graphical Representation

We fix a finite set of $q$ distinct program variables $\mathcal{V} = \{x_1, \ldots, x_q\} \subseteq \text{PVAR}$; to each memory state $(s, h)$, we associate several subsets of locations that together define two partitions of $\text{dom}(h)$. Beware that these subsets and partitions depend on the choice of $q$ and $\mathcal{V}$:

- one partition takes care of self-loops and predecessors of interpretations of program variables;
- the other one takes care of locations which are “close” to the interpretations of program variables; see below.

This allows us to decompose the heap domains in such a way that we can easily identify the properties that can be indeed expressed by the formulæ of 1SL1 that contain only the program variables of $\mathcal{V}$.
We introduce a first partition of the heap domain by distinguishing the selfloops and the predecessors of variable interpretations on the one hand, and the remaining locations in the domain on the other hand:

\[
\text{pred}(s, h, i) \equiv \{ l \mid h(l) = s(x_i) \} \quad \text{pred}(s, h) \equiv \text{pred}(s, h, 1) \cup \cdots \cup \text{pred}(s, h, q) \\
\text{loop}(s, h) \equiv \{ l \mid h(l) = l \} \\
\text{rem}(s, h) \equiv \text{dom}(h) \setminus (\text{pred}(s, h) \cup \text{loop}(s, h))
\]

where \(\text{pred}(s, h, i)\) is defined for any \(i \in [1, q]\). From the definition of \(\text{rem}(s, h)\), we derive the obvious identity \(\text{dom}(h) = \text{rem}(s, h) \cup (\text{pred}(s, h) \cup \text{loop}(s, h))\).

However, the sets \(\text{pred}(s, h)\) and \(\text{loop}(s, h)\) are not necessarily disjoint. As a consequence of \(h\) being a partial function, the sets \(\text{pred}(s, h, i)\) and \(\text{pred}(s, h, j)\) intersect only if \(s(x_i) = s(x_j)\), in which case the identity \(\text{pred}(s, h, i) = \text{pred}(s, h, j)\) holds.

We introduce a second partition of \(\text{dom}(h)\) by distinguishing the locations related to a memory cell involving a program variable interpretation on the one hand, and the remaining locations in the domain on the other hand. So, the sets below depend also implicitly on \(V\):

\[
\text{ref}(s, h) \equiv \text{dom}(h) \cap s(V) \\
\text{acc}(s, h) \equiv \text{dom}(h) \cap h(s(V)) \\
\bigodot(s, h) \equiv \text{ref}(s, h) \cup \text{acc}(s, h) \\
\bigtriangledown(s, h) \equiv \text{dom}(h) \setminus \bigodot(s, h)
\]

The core of the memory state \((s, h)\), written \(\bigtriangledown(s, h)\), contains the locations \(l\) in \(\text{dom}(h)\) such that either \(l\) is the interpretation of a program variable or it is an image by \(h\) of a program variable (that is also in the domain). We have \(\text{ref}(s, h) \subseteq s(V)\), whence \(\text{card}(\text{ref}(s, h)) \leq q\). Since \(h\) is a partial function, from \(\text{acc}(s, h) \subseteq h(s(V))\) we deduce \(\text{card}(\text{acc}(s, h)) \leq q\). As a consequence, we have \(\text{card}(\bigtriangledown(s, h)) \leq 2q\). Hence, the core of a memory heap is always a “small part” of the domain, small meaning of bounded size when \(q\) is fixed.

In the sequel, we need to consider locations that belong to the intersection of those sets from different partitions. Here are their formal definitions:

\[
\text{pred}_{\bigodot}(s, h, i) \equiv \text{pred}(s, h, i) \setminus \bigodot(s, h) \\
\text{pred}_{\bigtriangledown}(s, h) \equiv \text{pred}(s, h) \setminus \bigtriangledown(s, h) \\
\text{loop}_{\bigodot}(s, h) \equiv \text{loop}(s, h) \setminus \bigodot(s, h) \\
\text{loop}_{\bigtriangledown}(s, h) \equiv \text{loop}(s, h) \setminus \bigtriangledown(s, h) \\
\text{rem}_{\bigodot}(s, h) \equiv \text{rem}(s, h) \setminus \bigodot(s, h) \\
\text{rem}_{\bigtriangledown}(s, h) \equiv \text{rem}(s, h) \setminus \bigtriangledown(s, h)
\]

For instance, \(\text{pred}_{\bigodot}(s, h)\) contains the set of locations \(l\) in \(\text{dom}(h)\) that are predecessors of a variable interpretation but no program variable \(x_i\) in \(\{x_1, \ldots, x_q\}\) satisfies \(s(x_i) = l\) or \(h(s(x_i)) = l\) (which means \(l \notin \bigtriangledown(s, h)\)).

We insist on the fact that the definitions of subsets like \(\text{ref}(s, h)\), \(\text{acc}(s, h)\) and \(\bigtriangledown(s, h)\) and hence \(\text{pred}_{\bigodot}(s, h, i)\), \(\text{pred}_{\bigtriangledown}(s, h, i)\), \(\text{loop}_{\bigodot}(s, h)\), ... depend on a particular choice of \(q\) and \(V = \{x_1, \ldots, x_q\}\) even though our notation does not reflect that dependency. We made this choice for the sake of readability. However, in most of developments of this paper, and unless stated otherwise, we assume a fixed choice for \(q\) and \(V\).
This graph represents an example of memory state \((s, h)\) with the variables \(x_1, \ldots, x_4\). Nodes labelled by ‘♥’ belong to \(\heartsuit_s(h)\); those labelled by ‘ö’ belong to \(\heartsuit_p(s, h)\); those labelled by ‘p’ belong to \(\heartsuit_p(s, h)\) and those labelled by ‘r’ belong to \(\heartsuit_r(s, h)\).

Fig. 2.1 The decomposition of a memory state

A memory state \((s, h)\) restricted to the finite set of program variables \(x_1, \ldots, x_q\) can be represented by a finite graph encoding the (graph) representation of the heap \(h\). Moreover, each variable \(x_i\) labels the location \(s(x_i)\), which may add a few more nodes in the case \(s(x_i)\) does not belong to the domain and codomain of \(h\). The graph of Figure 2.1 illustrates the previous definitions of subsets on a simple memory state.

The introduction of the above sets provides a canonical way to decompose the heap domains, which will be helpful in the sequel.

**Lemma 2.6 (Canonical decomposition)** For all stores \(s\) and all heaps \(h\), the following identity holds:

\[
\text{dom}(h) = \heartsuit(s, h) \cup \heartsuit_p(s, h) \cup \heartsuit_r(s, h)
\]

The proof is by straightforward verification using the fact that \(\heartsuit_p(s, h) \cap \heartsuit_r(s, h) \subseteq \heartsuit(s, h)\).

**Proposition 2.7** \(\{\heartsuit_p(s, h, i) \mid i \in [1, q]\}\) is a partition of \(\heartsuit_p(s, h)\).

The (easy) proof is left to the reader.

Remember that both \(\heartsuit_p(s, h, i) = \heartsuit_p(s, h, j)\) and \(\heartsuit_p(s, h, i) = \emptyset\) are possible. Below, we present properties about the canonical decomposition.

**Proposition 2.8 (Canonical decomposition and splitting)** Let us assume \(s, h, h_1, h_2\) such that \(h = h_1 \uplus h_2\). With the notation \(X\) for \(\mathbb{N}\setminus X\), the identities

\[
\begin{align*}
\heartsuit(s, h) \cap \text{dom}(h_1) &= \heartsuit(s, h_1) \cup \Delta(s, h_1, h_2) \\
\heartsuit(s, h) \cap \text{dom}(h_2) &= \heartsuit(s, h_2) \cup \Delta(s, h_2, h_1)
\end{align*}
\]

hold with \(\Delta(s, h_1, h_2) \triangleq \text{dom}(h_1) \cap h_2(s(\mathcal{V})) \cap \overline{s(\mathcal{V})} \cap \overline{h_1(s(\mathcal{V}))}\).

The proof is by Boolean computations.

The set \(\Delta(s, h_1, h_2)\) contains the locations belonging to the core of \(h\) and to the domain of \(h_1\), without being in the core of \(h_1\). Its expression in Proposition 2.8 uses only basic set-theoretical operations. From Proposition 2.8, we conclude that \(\heartsuit(s, h_1 \uplus h_2)\) can be different from \(\heartsuit(s, h_1) \cup \heartsuit(s, h_2)\).
Proposition 2.9  Let $s, h, h_1, h_2$ be such that $h = h_1 \cup h_2$ and let $i \in [1, q]$. The following identities hold:
1. $\text{pred}(s, h_1, i) = (\text{pred}(s, h, i) \cap \text{dom}(h_1)) \cup (\text{pred}(s, h, i) \cap \Delta(s, h_1, h_2))$;
2. $\text{loop}(s, h_1) = (\text{loop}(s, h) \cap \text{dom}(h_1)) \cup (\text{loop}(s, h) \cap \Delta(s, h_1, h_2))$;
3. $\text{rem}(s, h_1) = (\text{rem}(s, h) \cap \text{dom}(h_1)) \cup (\text{rem}(s, h) \cap \Delta(s, h_1, h_2))$.

The proof can be found in Appendix A starting at page 50.

Remark that $\cup$ is commutative, hence symmetric identities hold for the subsets $\text{pred}(s, h_2, i)$, $\text{loop}(s, h_2)$, and $\text{rem}(s, h_2)$. The following propositions describe the changes that occur in the canonical decomposition of a memory state when exactly one memory cell is added to the heap. Recall that we write $[l_1 \rightarrow l_2]$ for the unique atomic heap $h$ such that $\text{dom}(h) = \{l_1\}$ and $h(l_1) = l_2$. We write $p\cup(s, h)$ to denote the set $s(\mathcal{V}) \cup h(s(\mathcal{V}))$.

Proposition 2.10  Let $(s, h)$ be a memory state, $l_1 \in \mathbb{N}\setminus \text{dom}(h)$ and $l_2 \in \mathbb{N}$. Let us write $h_{1 \rightarrow 2}$ for $h_{\uplus}[l_1 \rightarrow l_2]$ and let $i \in [1, q]$. The following identities hold:

\[
\begin{align*}
\text{dom}(h_{1 \rightarrow 2}) &= \text{dom}(h) \cup \{l_1\} \\
\text{pred}(s, h_{1 \rightarrow 2}, i) &= \text{pred}(s, h, i) \cup \{l_1\} \text{ if } l_2 = s(x_i) \\
&\quad \cup \mathcal{O} \text{ if } l_2 \neq s(x_i) \\
\text{loop}(s, h_{1 \rightarrow 2}) &= \text{loop}(s, h) \cup \{l_1\} \text{ if } l_1 = l_2 \\
&\quad \cup \mathcal{O} \text{ if } l_1 \neq l_2 \\
\text{rem}(s, h_{1 \rightarrow 2}) &= \text{rem}(s, h) \cup \{l_1\} \text{ if } l_2 \neq s(\mathcal{V}) \cup \{l_1\} \\
&\quad \cup \mathcal{O} \text{ if } l_2 \in s(\mathcal{V}) \cup \{l_1\} \\
\end{align*}
\]

$\cup(s, h_{1 \rightarrow 2}) = \cup(s, h) \cup \begin{cases} 
\{l_1\} & \text{if } l_2 \in s(\mathcal{V}) \text{ and } l_2 \notin \cup(s, h) \\
\{l_1\} & \text{if } l_1 \notin s(\mathcal{V}) \text{ and } l_1 \notin h(s(\mathcal{V})) \\
\mathcal{O} & \text{if } l_1 \notin p\cup(s, h) 
\end{cases}$

The proof can be found in Appendix A starting at page 50.

Proposition 2.11  Let $(s, h)$ be a memory state, $l_1 \in \mathbb{N}\setminus \text{dom}(h)$ and $l_2 \in \mathbb{N}$. Let us write $h_{1 \rightarrow 2}$ for $h_{\uplus}[l_1 \rightarrow l_2]$ and let $i \in [1, q]$. The following identities hold:

\[
\begin{align*}
\text{pred}(s, h_{1 \rightarrow 2}, i) &= \begin{cases} 
\text{pred}(s, h, i) \cup \{l_1\} & \text{if } l_1 \notin p\cup(s, h) \text{ and } l_2 = s(x_i) \\
\text{pred}(s, h, i) - \{l_2\} & \text{if } l_1 \in s(\mathcal{V}) \text{ and } l_2 \in \text{pred}(s, h, i) \\
\text{pred}(s, h, i) & \text{otherwise} 
\end{cases} \\
\text{loop}(s, h_{1 \rightarrow 2}) &= \begin{cases} 
\text{loop}(s, h) \cup \{l_1\} & \text{if } l_1 \notin p\cup(s, h) \text{ and } l_1 = l_2 \\
\text{loop}(s, h) - \{l_2\} & \text{if } l_1 \in s(\mathcal{V}) \text{ and } l_2 \in \text{loop}(s, h) \\
\text{loop}(s, h) & \text{otherwise} 
\end{cases} \\
\text{rem}(s, h_{1 \rightarrow 2}) &= \begin{cases} 
\text{rem}(s, h) \cup \{l_1\} & \text{if } l_1 \notin p\cup(s, h) \text{ and } l_2 \notin s(\mathcal{V}) \cup \{l_1\} \\
\text{rem}(s, h) - \{l_2\} & \text{if } l_1 \in s(\mathcal{V}) \text{ and } l_2 \in \text{rem}(s, h) \\
\text{rem}(s, h) & \text{otherwise} 
\end{cases} 
\end{align*}
\]

where $X - \{l_2\}$ means that the location $l_2$ already belongs to the set $X$ and is (strictly) removed from it.
2.4 How to Count in 1SL1

In this section, let us consider a fixed memory state \( (s, h) \) and a fixed location \( l \). We explain how to measure the cardinal of some finite sets of locations using 1SL1 formulæ, in particular those of the form \( X \updownarrow \text{pred}(s, h, j) \), \( \text{loop}(s, h) \) and \( \text{rem}(s, h) \). The ground idea is the following: using the identity
\[
X = (X \cap \updownarrow(s, h)) \cup (X \setminus \updownarrow(s, h))
\]
the cardinal of \( X \updownarrow \text{pred}(s, h, j) \) can be obtained from the cardinal of \( X \setminus \updownarrow(s, h) \) (by expressing their difference in 1SL1).

In 1SL1, it is easy to detect if there is one element in either \( \text{pred}(s, h, j) \) or \( \text{loop}(s, h) \) using the formulæ
\[
\# \text{pred}(x_j) \geq 1 \overset{\text{def}}{=} \exists u \leftarrow x_j \quad \text{and} \quad \# \text{loop} \geq 1 \overset{\text{def}}{=} \exists u \leftarrow u
\]
Hence, using the separating conjunction \( * \), we can measure the cardinal of \( \text{pred}(s, h, j) \) or \( \text{loop}(s, h) \) with
\[
\# \text{pred}(x_j) \geq k \overset{\text{def}}{=} \# \text{pred}(x_j) \geq 1 * * * \# \text{pred}(x_j) \geq 1 \text{ repeated } k \text{ times} \\
\# \text{loop} \geq k \overset{\text{def}}{=} \# \text{loop} \geq 1 * * * \# \text{loop} \geq 1 \text{ repeated } k \text{ times}
\]
This encoding works smoothly thanks to the identities \( \text{pred}(s, h_1 \uplus h_2, j) = \text{pred}(s, h_1, j) \cup \text{pred}(s, h_2, j) \), \( \text{loop}(s, h_1 \uplus h_2) = \text{loop}(s, h_1) \cup \text{loop}(s, h_2) \).

Let us explain how to evaluate the cardinal of \( \text{rem}(s, h) \). To do so, we define the following 1SL1 formulæ for every \( e \in \{x_1, \ldots, x_q\} \cup \{u\} \):
\[
\text{inrem}(e) \overset{\text{def}}{=} \text{alloc}(e) \land \neg e \leftarrow e \land \bigwedge_{j \in [1,q]} \neg e \leftarrow x_j \\
\text{torem}(x_i) \overset{\text{def}}{=} \text{toalloc}(x_i) \land \text{toloop}(x_i) \land \bigwedge_{j \in [1,q]} \text{btwn}(x_i, x_j)
\]

**Proposition 2.12** With \( [u] = l \) and \( [x_i] = s(x_i) \), we have:
\[
(s, h) \models_l \text{inrem}(e) \iff [e] \in \text{rem}(s, h) \\
(s, h) \models_l \text{torem}(x_i) \iff h(s(x_i)) \in \text{rem}(s, h)
\]
The proof is left to the reader.

Hence, the formula \( \# \text{rem} \geq 1 \overset{\text{def}}{=} \exists u \\text{inrem}(u) \) detects if \( \text{rem}(s, h) \) is non-empty. Using the separating conjunction \( * \), we can measure the cardinal of \( \text{rem}(s, h) \) in 1SL1 by
\[
\# \text{rem} \geq k \overset{\text{def}}{=} \# \text{rem} \geq 1 * * * \# \text{rem} \geq 1 \text{ repeated } k \text{ times}
\]
because the identity \( \text{rem}(s, h_1 \uplus h_2) = \text{rem}(s, h_1) \cup \text{rem}(s, h_2) \) holds too.
To count the number of elements in e.g. $\text{pred}_\mathcal{O}(s, h, j)$, it is thus sufficient to count the number of elements in $\text{pred}_\mathcal{O}'(s, h, j)$ and then use the identity

$$\text{pred}(s, h, j) = \text{pred}_\mathcal{O}(s, h, j) \cup \text{pred}_\mathcal{O}(s, h, j)$$

However, splitting $\text{pred}_\mathcal{O}(s, h, j)$ into $k$ parts using the separating conjunction like we did for $\text{pred}(s, h, j)$ will not work because $\mathcal{O}(s, h_1 \cup h_2)$ is not necessarily equal to $\mathcal{O}(s, h_1) \cup \mathcal{O}(s, h_2)$; see Proposition 2.8 for instance.

By contrast, one possible trick consists in directly enumerating the elements in $\text{pred}(s, h, j) \cap \text{ref}(s, h)$ and then use the identity

$$\text{pred}_\mathcal{O}(s, h, j) = (\text{pred}(s, h, j) \cap \text{ref}(s, h)) \cup (\text{acc}(s, h) \cap \text{ref}(s, h))$$

For any subset $I \subseteq [1, q]$, let us define the 1SL1 formulae below

$$\text{ref}_I \eqdef \bigwedge_{i \in I} \neg(x_i = x_j) \land \bigwedge_{i \in I} \text{alloc}(x_i)$$

$$\text{acc}_I \eqdef \bigwedge_{i \in I} \neg\text{conv}(x_i, x_j) \land \bigwedge_{i \in I} \text{toalloc}(x_i) \land \bigwedge_{i \in I, j \in [1, q]} \neg(x_i \hookrightarrow x_j)$$

**Proposition 2.13** For $I \subseteq [1, q]$ with $s I \eqdef \{s(x_i) \mid i \in I\}$, we have

$$\begin{align*}
(s, h) \vDash_I \text{ref}_I & \iff s I \subseteq \text{ref}(s, h) \text{ and } \text{card}(s I) = \text{card}(I) \\
(s, h) \vDash_I \text{acc}_I & \iff h(s I) \subseteq \text{acc}(s, h) \cap \text{ref}(s, h) \text{ and } \text{card}(h(s I)) = \text{card}(I)
\end{align*}$$

The proof is left to the reader.

Hence, $\text{ref}_I$ holds iff $s I$ is a subset of $\text{ref}(s, h)$ of size $\text{card}(I)$ and thus $\text{card}(I)$ gives a lower bound for the cardinal of $\text{ref}(s, h)$. Moreover, $\text{acc}_I$ provides us a way to give a lower bound for the cardinal of $\text{acc}(s, h) \cap \text{ref}(s, h)$.

To illustrate the usefulness of $\text{ref}_I$ and $\text{acc}_I$, we show how to measure the cardinal of the core. Using the identity $\mathcal{O}(s, h) = \text{ref}(s, h) \cup \text{acc}(s, h) \cap \text{ref}(s, h)$, we can express the fact that $\mathcal{O}(s, h)$ has cardinal at least $k$ by

$$\# \text{core} \geq k \eqdef \bigvee \{ \text{ref}_R \land \text{acc}_A \mid R \cup A \subseteq [1, q] \text{ and } \text{card}(R) + \text{card}(A) \geq k \}$$

But we can easily measure the cardinal of subsets of the core such as e.g. $\text{pred}_\mathcal{O}(s, h, j)$. Its cardinal is at least $k$ iff the following formula is satisfied

$$\# \text{pred}_\mathcal{O}(x_j) \geq k \eqdef \bigvee \left\{ \text{ref}_R \land \bigwedge_{x \notin R} x_j \hookrightarrow x_j \land \text{acc}_A \land \bigwedge_{x \in A} \text{btwn}(x_i, x_j) \mid R \cup A \subseteq [1, q] \text{ and } \text{card}(R) + \text{card}(A) \geq k \right\}$$

and, using $\text{pred}(s, h, j) = \text{pred}_\mathcal{O}(s, h, j) \cup \text{pred}_\mathcal{O}(s, h, j)$ we conclude that the cardinal of $\text{pred}_\mathcal{O}(s, h, j)$ is at least $k$ iff the following formula is satisfied

$$\# \text{pred}_\mathcal{O}(x_j) \geq k \eqdef \bigvee_{p \leq 2q} \# \text{pred}(x_j) \geq k + p \land \neg \# \text{pred}_\mathcal{O}(x_j) \geq p + 1$$

Notice that we can stop at $2q$ because of the inclusion $\text{pred}_\mathcal{O}(s, h, j) \subseteq \mathcal{O}(s, h)$ and the upper bound $\text{card}(\mathcal{O}(s, h, j)) \leq 2q$. 

Similarly, measuring the sizes of $\text{loop}_\varnothing(s,h)$ and $\text{rem}_\varnothing(s,h)$ is done with
\[
\#\text{loop}_\varnothing \geq k \overset{\text{def}}{=} \bigvee_{p \leq 2q} \#\text{loop} \geq k + p \land \neg \#\text{loop}_\varnothing \geq p + 1
\]
and
\[
\#\text{rem}_\varnothing \geq k \overset{\text{def}}{=} \bigvee_{p \leq 2q} \#\text{rem} \geq k + p \land \neg \#\text{rem}_\varnothing \geq p + 1
\]

Lemma 2.14 For any $k \geq 1$ and for any $i \in [1,q]$, there exist $1SL1$ formulae denoted $\#\text{pred}_\varnothing(x_i) \geq k$, $\#\text{loop}_\varnothing \geq k$ and $\#\text{rem}_\varnothing \geq k$ respectively such that, for any memory state $(s,h)$ and for any location $l \in \mathbb{N}$ the following equivalences hold:
1. $(s,h) \models_1 \#\text{pred}_\varnothing(x_i) \geq k \iff \text{card}(\text{pred}_\varnothing(s,h,i)) \geq k$;
2. $(s,h) \models_1 \#\text{loop}_\varnothing \geq k \iff \text{card}(\text{loop}_\varnothing(s,h)) \geq k$;
3. $(s,h) \models_1 \#\text{rem}_\varnothing \geq k \iff \text{card}(\text{rem}_\varnothing(s,h)) \geq k$.

The proof can be found in Appendix A starting at page 53.

2.5 Equipotence for Comparing Cardinalities

We introduce the notion of equipotence and we state several properties about it. This will be useful in the forthcoming developments.

Definition 2.15 We say that $x/y$ respect $X/Y$ if $(x \in X \iff y \in Y)$ holds.

Proposition 2.16 Let us consider four sets $X \subseteq X'$ and $Y \subseteq Y'$. Let $R \subseteq X' \times Y'$ be a bijective relation between $X'$ and $Y'$. Let us assume that for any $x,y$, if $x R y$ then $x/y$ respects $X/Y$. Then $R \cap X \times Y$ is a bijective relation between $X$ and $Y$.

The proof is left to the reader.

Definition 2.17 ($\alpha$-equipotence) Let $\alpha \in \mathbb{N}$. We say that two finite sets $X$ and $Y$ are $\alpha$-equipotent and we write $X \sim_\alpha Y$ if, either $\text{card}(X) = \text{card}(Y)$ or both $\text{card}(X)$ and $\text{card}(Y)$ are greater that $\alpha$. We say that two finite sets $X$ and $Y$ are equipotent and we write $X \sim_\infty Y$ when $\text{card}(X) = \text{card}(Y)$.

Proposition 2.18 For any $\alpha \in \mathbb{N}$ and any finite sets $X$ and $Y$, the following conditions are equivalent:
Proof
There are three cases:

1. \( X \sim_{\alpha} Y \);
2. \( \text{card}(X) \geq k \iff \text{card}(Y) \geq k \) for any \( k \leq \alpha \);
3. \( \text{card}(X) = \text{card}(Y) < \alpha \) or \( \text{card}(X) \geq \alpha \) and \( \text{card}(Y) \geq \alpha \);
4. \( \min(\text{card}(X), \alpha) = \min(\text{card}(Y), \alpha) \).

The proof is left to the reader.

It is thus obvious that \( X \sim_{\alpha} Y \) holds whenever \( \text{card}(X) = \text{card}(Y) \), i.e.
\( \sim_{\infty} \subseteq \sim_{\alpha} \). The equipotence relation is also decreasing, i.e. \( \sim_{\alpha_{2}} \subseteq \sim_{\alpha_{1}} \) holds for all \( \alpha_{1} \leq \alpha_{2} \). It is easy to verify \( \sim_{\infty} = \bigcap_{\alpha} \sim_{\alpha} \). Hence the notation \( \sim_{\infty} \) is consistent with the intuitive idea of a downward limit. We state below two lemmas that will be helpful in the sequel.

Lemma 2.19 Let \( \alpha \in \mathbb{N} \) and \( X, X', Y, Y' \) be finite sets such that \( X \cap X' = \emptyset \), \( Y \cap Y' = \emptyset \), \( X \sim_{\alpha} Y \) and \( X' \sim_{\infty} Y' \) hold. Then \( X \uplus X' \sim_{\alpha + 1} Y \uplus Y' \) holds where \( n = \text{card}(X') \).

The proof is left to the reader.

Proposition 2.20 Let \( X \) and \( Y \) be two finite sets and let \( x \notin X \) and \( y \notin Y \). If the equipotence \( X \uplus \{x\} \sim_{\alpha + 1} Y \uplus \{y\} \) holds then \( X \sim_{\alpha} Y \) holds.

The proof is left to the reader.

Lemma 2.21 Let \( \alpha_{1}, \alpha_{2} \in \mathbb{N} \) and \( X, X', Y_{0} \) be finite sets such that \( X \uplus X' \sim_{\alpha_{1} + \alpha_{2}} Y_{0} \) holds. Then there are two finite sets \( Y, Y' \) such that \( Y_{0} = Y \uplus Y' \), \( X \sim_{\alpha_{1}} Y \) and \( X' \sim_{\alpha_{2}} Y' \) hold.

The proof can be found in Appendix A starting at page 53.

Lemma 2.22 Let \( \alpha_{1}, \alpha_{2} \geq 1 \) and \( X, X', Y_{0} \) be finite sets and \( x, y \) be elements such that \( X \uplus X' \sim_{\alpha_{1} + \alpha_{2}} Y_{0} \) holds and \( x/y \) respect \( X \uplus X'/Y_{0} \). Then there are two finite sets \( Y \) and \( Y' \) such that \( Y_{0} = Y \uplus Y' \), \( X \sim_{\alpha_{1}} Y \) and \( X' \sim_{\alpha_{2}} Y' \) hold, and \( x/y \) respect both \( X/Y \) and \( Y/Y' \).

Proof There are three cases:

- if \( x \in X \) then we have \( x \in X \uplus X' \) and as a consequence \( y \in Y_{0} \). Let us define \( X'' = X \setminus \{x\} \) and \( Y''_{0} = Y_{0} \setminus \{y\} \). As \( \alpha_{1} \geq 1 \), by Proposition 2.20 we deduce \( X'' \uplus X' \sim_{\alpha_{1} + \alpha_{2}} Y''_{0} \). By Lemma 2.21, we obtain \( Y'' \) and \( Y' \) such that \( Y''_{0} = Y'' \uplus Y' \), \( X'' \sim_{\alpha_{1}} Y'' \) and \( X' \sim_{\alpha_{2}} Y' \). Observe that \( y \notin Y'' \) and \( y \notin Y' \) because \( Y'' \uplus Y' \subseteq Y_{0} \setminus \{y\} \). We define \( Y = Y'' \uplus \{y\} \). By Lemma 2.19, we have \( X = X'' \uplus \{x\} \sim_{\alpha_{1}} Y'' \uplus \{y\} = Y \). Since \( x \in X \) and \( y \in Y \), \( x/y \) respects \( X/Y \); and since \( x \notin X' \) (because \( X \cap X' = \emptyset \)) and \( y \notin Y' \), \( x/y \) respects \( X'/Y' \);
- the case \( x \in X' \) is the symmetric case;
- if \( x \notin X \uplus X' \) then \( y \notin Y_{0} \). Using Proposition 2.21 we choose \( Y, Y' \) such that \( Y_{0} = Y \uplus Y' \), \( X \sim_{\alpha_{1}} Y \) and \( X' \sim_{\alpha_{2}} Y' \). From \( x \notin X \uplus X' \) and \( y \notin Y \uplus Y' \), we deduce \( x \notin X, x \notin X', y \notin Y \) and \( y \notin Y' \) hence \( x/y \) respect both \( X/Y \) and \( Y/Y' \). \(\Box\)
Proposition 2.23 Let $X$ and $Y$ be two 2-equivalent finite sets, i.e. $X \sim_2 Y$. Let $x$ and $y$ be such that $x/y$ respect $X/Y$. For any $u \in X\setminus\{x\}$ there exists $v \in Y\setminus\{y\}$.

Proof Let $u \in X\setminus\{x\}$. We have two cases: either $x \in X$ and then we have $u \neq x \in X$ thus $\text{card}(X) \geq 2$. Using $X \sim_2 Y$ we deduce $\text{card}(Y) \geq 2$ and thus $\text{card}(Y\setminus\{y\}) \geq 1$; or $x \notin X$ and then we have $y \notin Y$ and thus $X\setminus\{x\} = X \sim_2 Y = Y\setminus\{y\}$. Hence if $X\setminus\{x\}$ is non-empty then so is $Y\setminus\{y\}$.

Proposition 2.24 For any $\alpha \geq 0$ and any finite set $X$, there exists a partition $X = X_1 \cup X_2$ of $X$ such that $X_1 \sim_\alpha X_1 \cup X_2$ and $\text{card}(X_1) \leq \alpha$.

Proof If $\text{card}(X) \leq \alpha$ then, we define $X_1 = X$ and $X_2 = \emptyset$. If $\text{card}(X) > \alpha$ then choose $X_1 \subseteq X$ such that $\text{card}(X_1) = \alpha$ (e.g. $X_1$ is composed of the least $\alpha$ elements of $X$) and $X_2 = X \setminus X_1$.

3 Test formulæ and Pointed Memory States

In this section, we consider a fixed set $\mathcal{V} = \{x_1, \ldots, x_q\}$ of $q \geq 1$ distinct program variables. The value $q$ can always be chosen large enough to accommodate a formula that contains many program variables. Below, we introduce test formulæ stating simple properties and we show that every formula in 1SL1 is equivalent to a Boolean combination of test formulæ.

3.1 Test Formulae for 1SL1

Test formulæ express simple properties about the memory states; this includes properties about program variables but also global properties about numbers of predecessors or loops, following the decomposition in Section 2.3. These test formulæ allow us to characterize the expressive power of 1SL1, similarly to what has been done in (Lozes 2004a,b; Brochenin et al 2009) for 1SL0. Moreover, we aim at defining the class of test formulæ as small as possible in order to nail down the very expressive power of 1SL1.

Since every formula in 1SL1 is shown equivalent to a Boolean combination of test formulæ (forthcoming Theorem 4.11), this process can be viewed as a means to eliminate separating connectives in a controlled way; elimination is not total since the test formulæ require such separating connectives. However, this is analogous to quantifier elimination in Presburger arithmetic (Presburger 1929) for which simple modulo constraints need to be introduced in order to eliminate the quantifiers (of course, modulo constraints are defined with quantifiers but in a controlled way too).

In Sections 2.2 and 2.4, we explained how to express the following test formulæ in 1SL1. In particular in Lemma 2.14, we show how to define the formulæ $\# \text{loop} \geq k$ [resp. $\# \text{rem} \geq k$ and $\# \text{pred}(x_i) \geq k$] in 1SL1. Notice however that the precise way they are actually defined in 1SL1 does not impact our developments.
Definition 3.1 (Test formulæ) Given $\alpha \geq 0$, we define sets of test formulæ:

- **Equality** $\overset{\text{def}}{=} \{ x_i = x_j \mid i, j \in [1, q] \}$
- **Pattern** $\overset{\text{def}}{=} \{ x_i \xrightarrow{} x_j, \text{conv}(x_i, x_j), \text{btwn}(x_i, x_j) \mid i, j \in [1, q] \} \\
  \cup \{ \text{toalloc}(x_i), \text{toloop}(x_i) \mid i \in [1, q] \}$
- **Extra** $\overset{\text{def}}{=} \{ u = u \xrightarrow{} u, \text{alloc}(u) \} \\
  \cup \{ x_i = u, u = x_i, x_i \xrightarrow{} u, u \xrightarrow{} x_i \mid i \in [1, q] \}$
- **Size** $\overset{\text{def}}{=} \{ \# \text{pred}_\sigma(x_i) \geq k \mid i \in [1, q], k \in [1, \alpha] \} \\
  \cup \{ \# \text{loop}_\sigma \geq k, \# \text{rem}_\sigma \geq k \mid k \in [1, \alpha] \}$
- **Basic** $\overset{\text{def}}{=} \text{Equality} \cup \text{Pattern}$
- **Test** $\overset{\text{def}}{=} \text{Basic} \cup \text{Size} \cup \{ \bot \}$
- **Basic** $\overset{\text{def}}{=} \text{Basic} \cup \text{Extra}$
- **Test** $\overset{\text{def}}{=} \text{Test} \cup \text{Extra}$

We observe that Size is an empty set of formulæ. The formula alloc($x_i$) is not included because we use the logically equivalent conv($x_i, x_i$). Notice however that alloc($u$) (defined by ($u \xrightarrow{} u \rightarrow \bot$) cannot be replaced because conv($x_i, x_i$) needs the quantifier $\exists u$ to be defined in 1SL. Unlike alloc($x_i$), the test formula alloc($u$) cannot be defined from conv.

It is important to note that the sets Basic, Size and Test depend on a particular choice of $q$ and $\alpha$; and again our notation does not reflect that dependency. By way of example, the definition of # loop $\geq k$ and its semantics inherently depend on $q$ (and on $V = \{ x_1, \ldots, x_q \}$); see Proposition 2.14. As an illustration, the conjunction of test formulæ

$$x_1 \xrightarrow{} x_2 \wedge x_2 \xrightarrow{} x_1 \wedge \neg (x_1 = x_2) \wedge u \xrightarrow{} u \wedge \neg \# \text{ loop} \geq 1$$

is satisfiable if $q \geq 3$ and unsatisfiable if $q = 2$ because in any model, the interpretation of $u$ (a self-loop) must belong the interpretation of programs variables $V = \{ x_1, \ldots, x_q \}$. But $u$ cannot be interpreted by either $x_1$ or $x_2$ unless $x_1 = x_2$ is satisfied as well.

Proposition 3.2 (Monotonicity of Basic) Let $s, h_1, h_2$ and $l \in \mathbb{N}$ be such that $h_1 \leq h_2$. For any formula $B \in \text{Basic}$, if $(s, h_1) \models B$ then $(s, h_2) \models B$.

The proof is left to the reader.

3.2 Satisfiability of Boolean Combinations of Test Formulae

Definition 3.3 (Literals and atoms) A literal is a test formula in Test or the negation of a test formula in Test. An atom is a saturated conjunction of literals from Test, i.e. each formula from Test occurs exactly once, possibly negated.

Hence atoms are either inconsistent (and equivalent to $\bot$) or minimal elements in the Boolean algebra generated by Test; any Boolean combination of formulæ of Test is equivalent to a disjunction of atoms.
Definition 3.4 The satisfiability problem for Boolean combinations of test formulæ is defined by:

**Input:** a set of program variables $V = \{x_1, \ldots, x_q\}$; a Boolean combination $A$ of formulæ from $\bigcup_{\alpha \geq 1} \text{Test}^\alpha_\alpha$ built on $V$ (the bounds $k$ in the $\text{Size}_\alpha$ formulæ are encoded in binary).

**Question:** is $A$ satisfiable with test formulæ understood using the set $V$?

Indeed, we need to specify the set $\{x_1, \ldots, x_q\}$ as an input because the meaning of test formulæ like $\#_1\text{loop} \geq 1$ depends on $\{x_1, \ldots, x_q\}$; see the previous section.

Theorem 3.5 The satisfiability problem for Boolean combinations of test formulæ is $\text{NP}$-complete.

*Proof* $\text{NP}$-hardness follows from a reduction from SAT, assuming that the number of program variables is unbounded. Indeed, each propositional variable $p_i$ can be encoded by the equality $x_{2i} = x_{2i+1}$ where the program variables $x_{2i}$ and $x_{2i+1}$ are dedicated to $p_i$ only. The proof of the $\text{NP}$ upper bound is given in Section 5.

Checking the satisfiability status of a Boolean combination of test formulæ is typically the kind of tasks that could be performed by an SMT solver, see e.g. (de Moura and Björner 2008; Barrett et al 2011).

3.3 Observing Pointed Memory States with Test Formulæ

We introduce the notion of pointed memory state and three kinds of equivalence relations between pointed memory states: $\equiv_\alpha$-equivalence denoted $\triangleright_\alpha$ and basic equivalence denoted $\triangleright_b$.

**Definition 3.6 (Pointed memory state, maxval)** The triple $m = (s, h, l)$ is a pointed memory state if $(s, h)$ is a memory state and $l \in \mathbb{N}$ is a location. We define

$$\text{maxval}(h) \overset{\text{def}}{=} \max \{\text{dom}(h) \cup \text{ran}(h)\}$$

$$\text{maxval}(s, h) \overset{\text{def}}{=} \max \{s(V) \cup \text{dom}(h) \cup \text{ran}(h)\}$$

$$\text{maxval}(m) \overset{\text{def}}{=} \max \{s(V) \cup \text{dom}(h) \cup \text{ran}(h) \cup \{l\}\}$$

**Definition 3.7 (Pseudo-core)** The pseudo-core of a memory state $(s, h)$, written $p\triangleright(s, h)$, is defined as $p\triangleright(s, h) \overset{\text{def}}{=} s(V) \cup h(s(V))$. The pseudo-core of a pointed memory state $m = (s, h, l)$ is defined by $p\triangleright(m) \overset{\text{def}}{=} s(V) \cup h(s(V)) \cup \{l\}$.

We observe that for any $u > \text{maxval}(s, h)$, we have $u \notin \text{dom}(h) \cup p\triangleright(s, h)$ and if $u > \text{maxval}(s, h, l)$ then $u \notin \text{dom}(h) \cup p\triangleright(s, h, l)$. Note also that the identity $\triangleright(s, h) = p\triangleright(s, h) \cap \text{dom}(h)$ holds. Moreover, $\triangleright(s, h) \subseteq p\triangleright(s, h)$ and $p\triangleright(s, h)$ may contain locations that are not in $\text{dom}(h)$, unlike the core $\triangleright(s, h)$.

Below, we introduce equivalence relations depending on whether memory states are indistinguishable with respect to some sets of test formulæ.
Definition 3.8 (Equivalences) Given pointed memory states \( m = (s, h, l) \) and \( m' = (s', h', l') \), we say that \( m \) and \( m' \) are \( \alpha \)-equivalent and we write \( m \approx_\alpha m' \) when the equivalence

\[(s, h) \models \mathcal{B} \iff (s', h') \models \mathcal{B} \text{ holds for any } \mathcal{B} \in \text{Test}_\alpha^u.\]

We also define the basic equivalence, written \( \approx_b \), by using \( \text{Basic}^u \) instead of \( \text{Test}_\alpha^u \).

It is obvious that \( \approx_\alpha \) and \( \approx_b \) are indeed equivalence relations between pointed memory states. The pointed memory states \( m \) and \( m' \) are basically equivalent (resp. \( \alpha \)-equivalent) if and only if they cannot be distinguished by the formulæ of \( \text{Basic}^u \) (resp. \( \text{Test}_\alpha^u \)). Since the inclusion \( \text{Basic}^u \subseteq \text{Test}_\alpha^u \) holds, it is obvious that the inclusion \( \approx_\alpha \subseteq \approx_b \) holds. Also observe that the identity \( \approx_b = \approx_0 \) holds because the set of formulæ \( \text{Test}_0^u \) is identical to \( \text{Basic}^u \cup \{ \bot \} \).

Nevertheless, we think it is clearer to keep a separate notation for \( \approx_b \).

Proposition 3.9 Let \( (s, h, l) \) and \( (s', h', l') \) be two pointed memory states such that \( (s, h, l) \approx_b (s', h', l') \). Then \( (s, \square, l) \approx_b (s', \square, l') \) holds.

The proof is left to the reader; remember that \( \square \) denotes the empty heap.

Proposition 3.10 For any \( \alpha \geq 0 \), the relation \( (s, h, l) \approx_\alpha (s', h', l') \) holds if and only if the four following conditions hold:

1. \( (s, h, l) \approx_b (s', h', l') \);
2. \( \text{pred}(s, h, i) \approx_\alpha \text{pred}(s', h', i) \text{ for any } i \in [1, q] \);
3. \( \text{loop}(s, h) \approx_\alpha \text{loop}(s', h') \);
4. \( \text{rem}(s, h) \approx_\alpha \text{rem}(s', h') \).

The proof follows from \( \text{Test}_\alpha^u = \{ \bot \} \cup \text{Basic}^u \cup \text{Size}_\alpha \) and Lemma 2.14.

Proposition 3.11 Let \( m = (s, h, l) \) and \( m' = (s', h', l') \) be two pointed memory states such that \( \text{dom}(h) \subseteq \bigcup(s, h) \) and \( \text{dom}(h') \subseteq \bigcup(s', h') \). For any \( \alpha \geq 0 \), \( m \approx_\alpha m' \) iff \( m \approx_b m' \).

This is a direct consequence of Proposition 3.10.

3.4 A Relational View of Basic Equivalence

In this section, we assume two pointed memory states \( m = (s, h, l) \) and \( m' = (s', h', l') \). We name some properties which will be used to define binary relations between locations.

Definition 3.12 For \( u, v \in \mathbb{N} \), we define the following properties:
We say that e.g. \(u/v\) verify (3) if this property holds.

We emphasize the fact that Properties (9) and (1) depend on \(m/m'\). More precisely, (9) and (1) depend on \(l/l'\), (2) and (1) depend on \(s/s'\), (4) depend on \(h/h'\), and the remaining (3) and (5) depend on \((s, h)/(s', h')\). When the context does not single out a unique choice for \(m/m'\), we will explicitly say that e.g. \(u/v\) verify (3) with respect to \(m/m'\).

The Properties (2–6) characterize precisely the positions of \(u\) (resp. \(v\)) in the canonical decomposition of \(m\) (resp. \(m'\)). The Property (1) characterizes the positions of \(u/v\) with respect to \(l/l'\). As witnessed in upcoming Lemma 3.25 and Propositions 3.29 and 3.31, we do not need all this precision to establish that basic or \(\alpha\)-equivalence is preserved by atomic extensions.

**Proposition 3.13** Let \(u, v \in \mathbb{N}\). For (10)–(20) defined as

1. (10) \(u \in s(V) \text{ iff } v \in s'(V)\);
2. (11) \(u \in h(s(V)) \text{ iff } v \in h'(s'(V))\);
3. (12) \(u \in p \backslash s(h) \text{ iff } v \in p \backslash s'(h')\);
4. (13) \(u \in \bigcup(s, h) \text{ iff } v \in \bigcup(s', h')\);
5. (14) \(u \in \text{pred}(s, h, i) \text{ iff } v \in \text{pred}(s', h', i)\) for any \(i \in [1, q]\);
6. (15) \(u \in \text{pred}(s, h) \text{ iff } v \in \text{pred}(s', h')\);
7. (16) \(u \in \text{loop}(s, h) \text{ iff } v \in \text{loop}(s', h')\);
8. (17) \(u \in \text{rem}(s, h) \text{ iff } v \in \text{rem}(s', h')\);
9. (18) \(u \in \text{rem}(s, h, i) \text{ iff } v \in \text{rem}(s', h', i)\) for any \(i \in [1, q]\);
10. (19) \(u \in \text{loop}(s, h) \text{ iff } v \in \text{loop}(s', h')\);
11. (20) \(u \in \text{rem}(s, h) \text{ iff } v \in \text{rem}(s', h')\);
12. (21) \(u \in p \backslash (m) \text{ iff } v \in p \backslash (m')\).

the following propositions hold:

1. (2) implies (10);
2. (3) implies (11);
3. (2–3) imply (12);
4. (2–4) imply (13);
5. (2–6) imply (10–20);
6. (11–3) imply (21).

The proof can be found in Appendix B starting at page 54.
Definition 3.14 (Binary relations between locations) We define binary relations between locations denoted \( R_{m,m'} \), \( R_{m,m',\ell} \), \( D_{m,m'} \), \( I_{m,m'} \) and \( T_{m,m'} \) by

\[
\begin{align*}
& R_{m,m'} u \iff (R2) \text{ or } (R3) \\
& R_{m,m',\ell} u \iff (R1) \text{ or } (R2) \text{ or } (R3) \\
& D_{m,m'} u \iff (T4) \\
& I_{m,m'} u \iff (T2) \text{ and } \ldots \text{ and } (T6) \\
& T_{m,m'} u \iff (T1) \text{ and } \ldots \text{ and } (T6)
\end{align*}
\]

In the rest of this section, we simply denote \( R, R', \ldots \) instead of \( R_{m,m'}, R_{m,m',\ell}, \ldots \). Obviously the inclusions \( R \subseteq R' \) and \( T \subseteq T' \) hold.

Proposition 3.15 The following inclusions hold:

1. \( R \subseteq p \bowtie (s, h) \times p \bowtie (s', h') \)
2. \( T \cap p \bowtie (s, h) \times N \subseteq R \)
3. \( T \cap N \times p \bowtie (s', h) \subseteq R \)
4. \( R' \subseteq p \bowtie (m) \times p \bowtie (m') \)
5. \( T' \cap p \bowtie (m) \times N \subseteq R' \)
6. \( T' \cap N \times p \bowtie (m') \subseteq R' \)

The proof can be found in Appendix B starting at page 55.

Proposition 3.16 The following properties hold:

1. The relation \( T \) restricted to \( p \bowtie (s, h) \times p \bowtie (s', h') \) is functional and injective;
2. The relation \( T' \) restricted to \( p \bowtie (m) \times p \bowtie (m') \) is functional and injective;
3. For any \( u \notin \text{dom}(h) \cup p \bowtie (s, h), v \notin \text{dom}(h') \cup p \bowtie (s', h') \), we have \( u \not\in T v \);
4. For any \( u \notin \text{dom}(h) \cup p \bowtie (m), v \notin \text{dom}(h') \cup p \bowtie (m') \), we have \( u \not\in T' v \).

The proof can be found in Appendix B starting at page 55.

Theorem 3.17 \( m \approx_h m' \) if and only if \( R' \subseteq T' \).

The proof can be found in Appendix B starting at page 55.

Proposition 3.18 If \( m \approx_h m' \) then the following properties hold:

1. The relation \( R \) is total and surjective between \( p \bowtie (s, h) \) and \( p \bowtie (s', h') \);
2. The relation \( R' \) is total and surjective between \( p \bowtie (m) \) and \( p \bowtie (m') \).

The proof can be found in Appendix B starting at page 56.

Lemma 3.19 (Bijections between pseudo-cores) When \( m \) and \( m' \) are basically equivalent, i.e. \( m \approx_h m' \), the following properties hold:

1. the inclusions \( R \subseteq T \) and \( R' \subseteq T' \) hold;
2. both \( R = T \cap p \bowtie (s, h) \times p \bowtie (s', h') \) and \( R' = T' \cap p \bowtie (m) \times p \bowtie (m') \) hold;
3. the relation \( R \) is bijective between \( p \bowtie (s, h) \) and \( p \bowtie (s', h') \);
4. the relation \( R' \) is bijective between \( p \bowtie (m) \) and \( p \bowtie (m') \);
5. the relation \( R' \cap p \bowtie (s, h) \times p \bowtie (s', h') \) is bijective between \( p \bowtie (s, h) \) and \( p \bowtie (s', h') \).
Proposition 3.20 If \( m \) and \( m' \) are 2-equivalent (i.e. \( m \simeq_2 m' \)), then \( 
abla \) is a total relation on \( N \): for any \( u \in N \), there exists \( v \leq \maxval(m') + 1 \) such that \( u \nabla v \).

Proof Since \( \simeq_2 \subseteq \simeq_0 \), we have \( \nabla \subseteq \nabla'l \) by Theorem 3.17. Let us consider \( u \in N \). We have to show that there exists \( v \in N \) such that \( u \nabla v \) holds. We determine the value of \( v \) according to the first condition satisfied in the list below.

- If \( u \in \pred'(s, h) \) then we define \( v \) as the unique location in \( \pred'(m') \) such that \( u \nabla v \). Derive \( u \nabla v \) holds. The relation \( v \leq \maxval(m') + 1 \) holds because \( v \in \pred'(m') \);

- If \( u \notin \pred'(s, h) \) for some \( j \in [1, q] \) then we know that \( u \neq l \) because the case \( u = l \in \pred(m) \) occurs earlier in the list.

Hence we have \( u \notin \pred'(s, h) \) for some \( j \in [1, q] \). From \( l \nabla_l l' \) we deduce \( l \nabla l' \). Hence by Proposition 3.13 (\( \nabla l' \) respect \( \pred'(s, h) \)) by Proposition 3.10. Hence by Proposition 2.23, we can choose a location \( v \in \pred'(s', h', j) \setminus \{l'\} \). The relation \( v \leq \maxval(m') + 1 \) holds because \( v \in \dom(h') \). Let us establish \( u \nabla v \). We have \( u \in \pred'(s, h, j) \setminus \{l\} \) and \( v \in \pred'(s', h', j) \setminus \{l'\} \).

As a consequence, we deduce \( u \neq \pred'(m) \) and \( v \neq \pred'(m') \). Hence, Properties (\( \nabla \)) hold. We also have \( u \in \dom(h) \) and \( v \in \dom(h') \), whence Property (\( \nabla ' \)) holds. We have \( h(u) = s(x_i) \) and \( h'(v) = s'(x_i) \). We deduce \( h(u) = h'(v) \) and thus \( h(u) \nabla h'(v) \).

Let us prove Property (\( \nabla \)) for \( u/v \): we use (\( \nabla l' \)) for \( h(u)/h'(v) \) and get \( h(u) = s(x_i) \) iff \( h'(v) = s'(x_i) \) for any \( i \in [1, q] \). Let us prove Property (\( \nabla \)): the identity \( u = h(u) \) implies \( u = s(x_j) \) which contradicts \( u \neq \pred(m) \). Hence \( u \neq h(u) \) and for similar reasons, \( v \neq h'(v) \).
In the remaining cases we have $u \notin p \mathcal{R} (m)$ and $v \notin p \mathcal{R} (m')$. Hence Properties (\textbf{3.1}–\textbf{3}) hold. We also have $u \in \text{dom}(h)$ and $v \in \text{dom}(h')$; hence Property (\textbf{4}) holds. We have $h(u) = u$ and $h'(v) = v$, whence Property (\textbf{5}) holds. We have already proved that Property (\textbf{2}) holds for $u/v$. As $h(u) = u$ and $h'(v) = v$ we deduce that Property (\textbf{2}) holds for $h(u)/h'(v)$. Hence Property (\textbf{5}) holds for $u/v$.

- If $u \in \text{rem}_{\mathcal{T}}(s, h)$ then $u \in \text{rem}_{\mathcal{T}}(s, h) \backslash \{l\}$. By Proposition 3.13 (\textbf{2}0), $l'/l$ respect $\text{rem}_{\mathcal{T}}(s, h)/\text{rem}_{\mathcal{T}}(s', h')$. We have $\text{rem}_{\mathcal{T}}(s, h) \sim_2 \text{rem}_{\mathcal{T}}(s', h')$ by Proposition 3.10. Hence by Proposition 2.23, we can choose a location $v \in \text{rem}_{\mathcal{T}}(s', h') \backslash \{l'\}$.

The relation $v \leq \maxval(m') + 1$ holds because $v \in \text{dom}(h')$. Let us check that $u \mathcal{T} v$ holds. We have $u \in \text{rem}_{\mathcal{T}}(s, h) \backslash \{l\}$ and $v \in \text{rem}_{\mathcal{T}}(s', h') \backslash \{l'\}$.

- In the remaining cases we have $u \notin \text{dom}(h) \cup p \mathcal{R} (m)$. Let $v = \maxval(m') + 1$. Then we have $v \notin \text{dom}(h') \cup p \mathcal{R} (m')$ and by Proposition 3.16 item 4, we conclude $u \not\mathcal{T} v$. □

If we do not require Property (\textbf{1}), i.e. we work with $\mathcal{T}$ instead of $\mathcal{S}$, then only $\sim_1$ is needed to establish that $\mathcal{T}$ is total from $\mathbb{N}$ to $[0, m + 1]$ with $m = \maxval(s', h')$.

**Proposition 3.21** If $m$ and $m'$ satisfy $m \sim_1 m'$, then $\mathcal{T}$ is a total relation on $\mathbb{N}$: for any $u \in \mathbb{N}$, there exists $v \leq \maxval(s', h') + 1$ such that $u \mathcal{T} v$.

The proof can be found in Appendix B starting at page 56.

### 3.5 Basic Equivalence and Heap Splitting

In this section, we consider two stores $s$ and $s'$, two locations $l$ and $l'$, and two heaps $h = h_1 \uplus h_2$ and $h' = h'_1 \uplus h'_2$ that are divided into two disjoint subheaps. Let us denote

$$
\begin{align*}
\mathbf{m} &= (s, h, l) & \mathbf{m'} &= (s', h', l') \\
\mathbf{m_1} &= (s, h_1, l) & \mathbf{m_1'} &= (s', h'_1, l') \\
\mathbf{m_2} &= (s, h_2, l) & \mathbf{m_2'} &= (s', h'_2, l') \\
\mathring{\mathbf{H}}_l &= \mathring{\mathbf{H}}_{m, m'}^l & \mathfrak{T}^l &= \mathfrak{T}_{m, m'}^l \\
\mathring{\mathbf{H}}_1 &= \mathring{\mathbf{H}}_{m_1, m_1'}^1 & \mathfrak{T}_1 &= \mathfrak{T}_{m_1, m_1'}^1 \\
\mathring{\mathbf{H}}_2 &= \mathring{\mathbf{H}}_{m_2, m_2'}^2 & \mathfrak{T}_2 &= \mathfrak{T}_{m_2, m_2'}^2 \\
\mathfrak{D}_1 &= \mathfrak{D}_{m_1, m_1'} & \mathfrak{D}_2 &= \mathfrak{D}_{m_2, m_2'}
\end{align*}
$$
It is trivial to check the inclusion $\mathcal{R}_t^\dagger \subseteq \mathcal{R}$ because $h_1 \subseteq h$ and $h'_1 \subseteq h'$. The inclusion $\mathcal{R}_t^\dagger \subseteq \mathcal{R}$ holds by symmetry.

Let us study under which conditions the splits $h = h_1 \uplus h_2$ and $h' = h'_1 \uplus h'_2$ preserve basic equivalence, i.e. when do $m_1 \simeq_b m_1'$ and $m_2 \simeq_b m_2'$ hold, provided that $m \simeq_b m'$ already holds.

**Proposition 3.22** Let us assume $\mathcal{R}^\dagger \subseteq \mathcal{T}^\dagger$ (or equivalently $m \simeq_b m'$). Then the following statements are equivalent:

1. $\mathcal{R}^\dagger \subseteq \mathcal{D}_1 \cap \mathcal{D}_2$;
2. $\mathcal{R}^\dagger \subseteq \mathcal{T}_1 \cap \mathcal{T}_2$;
3. $m_1 \simeq_b m_1'$ and $m_2 \simeq_b m_2'$.

The proof can be found in Appendix B starting at page 57.

If the splits $h = h_1 \uplus h_2$ and $h' = h'_1 \uplus h'_2$ preserve basic equivalence, then some subsets of the core are equipotent.

**Proposition 3.23** Let us assume $m \simeq_b m'$, $m_1 \simeq_b m_1'$ and $m_2 \simeq_b m_2'$. With the notation $\mathcal{X}$ for $\mathcal{N} \setminus \mathcal{X}$, the following properties hold:

1. $\text{pred}(s, h, i) \cap \Delta_c \sim_{\mathcal{X}} \text{pred}(s', h', i) \cap \Delta'_c$ for any $i \in [1, q]$;
2. $\text{loop}(s, h) \cap \Delta_c \sim_{\mathcal{X}} \text{loop}(s', h') \cap \Delta'_c$;
3. $\text{rem}(s, h) \cap \Delta_c \sim_{\mathcal{X}} \text{rem}(s', h') \cap \Delta'_c$.

where $c \in \{1, 2\}$ and \[
\Delta_c = \text{dom}(h_c) \cap h_{3-c}(s(V)) \cap s(V) \cap h_c(s(V)) \quad \Delta'_c = \text{dom}(h'_c) \cap h'_{3-c}(s'(V)) \cap s'(V) \cap h'_c(s'(V))
\]

**Proof** The definition of $\Delta(s, h, h_2)$ corresponds to that of Propositions 2.8 and 2.9. We have $\Delta_1 = \Delta(s, h_1, h_2)$, $\Delta_2 = \Delta(s, h_2, h_1)$, $\Delta'_1 = \Delta(s', h_1', h_2')$ and $\Delta'_2 = \Delta(s', h_2', h_1')$. We easily verify that the inclusions $\Delta_1 \cup \Delta_2 \subseteq \triangledown(s, h)$ and $\Delta'_1 \cup \Delta'_2 \subseteq \triangledown(s', h')$ hold. We invite the reader to check the following equivalences:

\[
u \in \text{pred}(s, h, j) \cap \Delta_1 \iff h_1(u) = s(x_j) \quad \text{and} \quad u \in h_2(s(V)) \quad \text{and} \quad u \notin s(V) \quad \text{and} \quad u \neq h_1(s(V))
\]

\[
u \in \text{loop}(s, h) \cap \Delta_1 \iff h_1(u) = u \quad \text{and} \quad u \in h_2(s(V)) \quad \text{and} \quad u \notin s(V) \quad \text{and} \quad u \neq h_1(s(V))
\]

\[
u \in \text{rem}(s, h) \cap \Delta_1 \iff u \in \text{dom}(h_1) \quad \text{and} \quad h_1(u) \notin s(V) \quad \text{and} \quad h(u) \neq u \quad \text{and} \quad u \in h_2(s(V)) \quad \text{and} \quad u \notin s(V) \quad \text{and} \quad u \neq h_1(s(V))
\]

From Proposition 3.22 and Theorem 3.19 we deduce $\mathcal{R}_t^\dagger \subseteq \mathcal{T}_1 \cap \mathcal{T}_1 \cap \mathcal{T}_2$. Moreover, by Lemma 3.19, the relation $\mathcal{R}_t^\dagger \cap \triangledown(s, h) \times \triangledown(s', h')$ is a bijection between $\triangledown(s, h)$ and $\triangledown(s', h')$.

Let us prove Property 1 with $c = 1$ for instance. We use Proposition 2.16: we have $\text{pred}(s, h, i) \cap \Delta_1 \subseteq \triangledown(s, h)$ and $\text{pred}(s', h', i) \cap \Delta'_1 \subseteq \triangledown(s', h')$. Hence let us show that if $u \in \triangledown(s, h)$ and $v \in \triangledown(s', h')$ verify $u \mathcal{R}_t^\dagger v$ then $u/v$ respect $\text{pred}(s, h, i) \cap \Delta_1$ and $\text{pred}(s', h', i) \cap \Delta'_1$: we use the first of the three above equivalences and $u \mathcal{T}_1 v$ with ($\mathcal{F}5$), $u \mathcal{T}_2 v$ with ($\mathcal{F}11$), $u \mathcal{T}_1 v$ with ($\mathcal{F}10$), and $u \mathcal{T}_1 v$ with ($\mathcal{F}11$).

Hence by Proposition 2.16, there is a bijection between $\text{pred}(s, h, i) \cap \Delta_1$ and $\text{pred}(s', h', i) \cap \Delta'_1$ and thus $\text{pred}(s, h, i) \cap \Delta_1 \sim_{\mathcal{X}} \text{pred}(s', h', i) \cap \Delta'_1$ holds. We use similar arguments for Property 1 ($c = 2$) and Properties 2–3. $\square$
3.6 Basic Equivalence and Location Update

In this section, we study under which conditions an update of the location \( l \) in the pointed memory state \((s, h, l)\) preserves basic equivalence.

**Proposition 3.24** Let \((s, h, l)\) and \((s', h', l')\) be basically equivalent pointed memory states, i.e., \((s, h, l) \approx_b (s', h', l')\). For any \( l_0, l'_0 \in \mathbb{N} \), if \( l_0/l'_0 \) verify (\( \Sigma 2–6 \)) then \((s, h, l_0) \approx_b (s', h', l'_0)\) holds.

**Proof** Since any formula \( B \in \text{Basic} \) contains no free occurrence of \( u \), by Proposition 2.2 we have \( (s, h) \models_{l_0} B \) iff \((s', h') \models_{l'_0} B\). As the identity \( \text{Basic}^a = \text{Basic} \cup \text{Extra}^a \) holds, to get \((s, h, l_0) \approx_b (s', h', l'_0)\) it is sufficient to prove the property \((s, h) \models_{l_0} B \) iff \((s', h') \models_{l'_0} B\) for any formula \( B \in \text{Extra}^a \). We proceed by a case analysis on \( B \); we display the *only if case*, the *if case* being proved in a symmetric way:

- \( B \) is \( u \rightarrow u \): from \((s, h) \models_{l_0} u \rightarrow u\) we get \( h(l_0) = l_0 \). Since \( l_0/l'_0 \) verify (\( \Sigma 6 \)), we deduce \( h'(l'_0) = l'_0\) and thus \((s', h') \models_{l'_0} u \rightarrow u\);
- \( B \) is \( \text{alloc}(u) \): from \((s, h) \models_{l_0} \text{alloc}(u)\) we get \( l_0 \in \text{dom}(h)\). Since \( l_0/l'_0 \) verify (\( \Sigma 4 \)), we deduce \( l'_0 \in \text{dom}(h')\) and thus \((s', h') \models_{l'_0} \text{alloc}(u)\);
- \( B \) is \( x_i = u \): from \((s, h) \models_{l_0} x_i = u\) we get \( s(x_i) = l_0\). Since \( l_0/l'_0 \) verify (\( \Sigma 2 \)), we deduce \( s'(x_i) = l'_0\) and thus \((s', h') \models_{l'_0} x_i = u\);
- \( B \) is \( x_i \rightarrow u \): from \((s, h) \models_{l_0} x_i \rightarrow u\) we get \( h(s(x_i)) = l_0 \). Since \( l_0/l'_0 \) verify (\( \Sigma 3 \)), we deduce \( h'(s'(x_i)) = l'_0\) and thus \((s', h') \models_{l'_0} x_i \rightarrow u\);
- \( B \) is \( u \rightarrow x_j \): from \((s, h) \models_{l_0} u \rightarrow x_j\) we get \( h(l_0) = s(x_j)\). Since \( l_0/l'_0 \) verify (\( \Sigma 5 \)), we deduce \( h'(l'_0) = s'(x_j)\) and thus \((s', h') \models_{l'_0} u \rightarrow x_j\). \( \square \)

3.7 Atomic Extensions and \( \alpha \)-Equivalence

Recall that we write \([l_1 \rightarrow l_2]\) to denote the (atomic) heap \( h \) such that \( \text{dom}(h) = \{l_1\}, h(l_1) = l_2 \) and \( \text{ran}(h) = \{l_2\} \). We study under which conditions atomic extensions preserve \( \alpha \)-equivalence.

**Lemma 3.25** Let \( \alpha \geq 1 \) and let \((s, h, l)\) and \((s', h', l')\) be two pointed memory states. Let \( l_1, l_2, l'_1, l'_2 \in \mathbb{N} \) be such that \( l_1 \notin \text{dom}(h) \) and \( l'_1 \notin \text{dom}(h') \). We assume that one of the conditions below holds:

- (C1) \( l_1/l'_1 \) verify (\( \Sigma 1–3 \)), \( l_2/l'_2 \) verify (\( \Sigma 6 \)), and \( l_2 = l_1 \) iff \( l'_2 = l'_1\);
- (C2) \( l_1 \notin s(V) \), \( l_1/l'_1 \) verify (\( \Sigma 1–3 \)), \( l_2/l'_2 \) verify (\( \Sigma 2 \)), and \( l_2 = l_1 \) iff \( l'_2 = l'_1\).

If \((s, h, l) \approx_\alpha (s', h', l')\) then \((s, h \upharpoonright [l_1 \rightarrow l_2], l) \approx_\beta (s', h' \upharpoonright [l'_1 \rightarrow l'_2], l')\) where \( \beta = \alpha - 1 \) if \( l_1 \in s(V) \), and \( \beta = \alpha \) otherwise.

The proof can be found in Appendix B starting at page 58.

Now let us present sufficient conditions under which an atomic extension does not change a pointed memory state up to \( \alpha \)-equivalence.
Proposition 3.26 Let \( m = (s, h, l) \) be a pointed memory state and \( l_1, l_2 \in \mathbb{N} \) be such that \( l_1 \notin \text{dom}(h) \cup \text{p Governors}(m) \). We have \((s, h \uplus [l_1 \mapsto l_2], l) \approx_{\alpha} (s, h, l)\). Moreover, given \( \alpha \geq 0 \), if we assume that one of the following conditions hold

- (C1) \( l_2 = s(x_i) \) and \( \text{card}(\text{pred}_{V}(s, h, i)) \geq \alpha \) for some \( i \in [1, q] \);
- (C2) \( l_2 = l_1 \) and \( \text{card}(\text{loop}_{V}(s, h)) \geq \alpha \);
- (C3) \( l_2 \notin s(V) \cup \{l_1\} \) and \( \text{card}(\text{rem}_{V}(s, h)) \geq \alpha \).

then we have \((s, h \uplus [l_1 \mapsto l_2], l) \approx_{\alpha} (s, h, l)\).

The proof can be found in Appendix B starting at page 61.

We extend the previous result to more general extensions that avoid adding locations in the pseudo-core.

Corollary 3.27 Let \( \alpha \geq 0 \). Let \( m = (s, h, l) \) be a pointed memory state and \( h' \) be a heap such that \( \text{dom}(h') \cap (\text{dom}(h) \cup \text{p Governors}(m)) = \emptyset \). If for any \( u \in \text{dom}(h') \) one of the following conditions holds

- (C1) \( h'(u) = s(x_i) \) and \( \text{card}(\text{pred}_{V}(s, h, i)) \geq \alpha \) for some \( i \in [1, q] \);
- (C2) \( h'(u) = u \) and \( \text{card}(\text{loop}_{V}(s, h)) \geq \alpha \);
- (C3) \( h'(u) \notin s(V) \cup \{u\} \) and \( \text{card}(\text{rem}_{V}(s, h)) \geq \alpha \).

then we have \((s, h \uplus h', l) \approx_{\alpha} (s, h, l)\).

The proof can be found in Appendix B starting at page 62.

3.8 Transposing Heap Extensions through \( \alpha \)-Equivalence

In this section, we assume \( s, s', h_0, h', h, l \) and \( l' \) such that \( h_0 \perp h \) and \( h_0' \perp h' \). We denote

\[
\begin{align*}
    m &= (s, h, l) \quad m_0 = (s, h_0 \uplus h, l) \\
    m' &= (s', h', l') \quad m_0' = (s', h_0' \uplus h', l') \\
    \mathbb{R}^1 &= \mathbb{R}^1_{m, m'} \quad \mathbb{T}^1 = \mathbb{T}^1_{m, m'} \\
    \mathbb{R}_0^1 &= \mathbb{R}_0^1_{m_0, m_0'}. \quad \mathbb{T}_0^1 = \mathbb{T}_0^1_{m_0, m_0'}
\end{align*}
\]

We insist that the heap of \( m_0 \) is \( h_0 \uplus h \), not \( h_0 \); the short notation might be a bit confusing here. Because \( h \subseteq h_0 \uplus h \) and \( h' \subseteq h_0' \uplus h' \), it is trivial to check that the inclusion \( \mathbb{R}^1 \subseteq \mathbb{R}_0^1 \) holds.

Proposition 3.28 We assume \( m \approx_{h} m', m_0 \approx_{h} m_0' \) and \( u, v \in \mathbb{N} \) such that \( u \nleq T_0 v \). If either \( u \in \text{p Governors}(m) \) or \( v \in \text{p Governors}(m') \) then \( u \nleq T_0^1 v \).

Proof We assume \( u \in \text{p Governors}(m) \) and we show \( u \nleq T_0^1 v \). From \( \text{p Governors}(m) \subseteq \text{p Governors}(m_0) \), we deduce \( u \in \text{p Governors}(m_0) \). By Proposition 3.15 item 5, from \( u \nleq T_0 v \) we deduce \( u \nleq T_0^1 v \) and thus \( v \in \text{p Governors}(m_0') \) by Proposition 3.15 item 4. Since \( u \in \text{p Governors}(m) \), by Lemma 3.19 there exists a unique location \( w \in \text{p Governors}(m') \) such that \( u \nleq T_0 w \). From \( \mathbb{R}^1 \subseteq \mathbb{R}_0^1 \) we deduce \( u \nleq T_0 w \). Hence, we have \( (u \nleq T_0 v \text{ and } u \nleq T_0 w) \) and by Proposition 3.15 item 4 and Lemma 3.19, \( \mathbb{R}_0^1 \) is a bijection. We deduce \( v = w \), and then \( u \nleq T_0^1 v \). \( \square \)
Proposition 3.29 Let \( \alpha \geq 1 \). We assume that the following conditions hold:

(a) \( m \equiv_{\alpha+1} m' \);
(b) \( m_0 \equiv_{\alpha+1} m'_0 \);
(c) \( \text{dom}(h) \subseteq p \forall (m) \);
(d) \( \text{dom}(h') \subseteq p \forall (m') \).

Let \( l_1 \in s(V) \setminus \text{dom}(h_0 \cup h) \) and \( l_2 \in \mathbb{N} \). There exist \( l'_1, l'_2 \in \mathbb{N} \) such that

1. \( l'_1 \in s'(V) \setminus \text{dom}(h'_0 \cup h') \);
2. \( l'_1, l'_2 \leq \text{maxval}(m'_0) + 1 \);
3. \( (s, h \uplus [l_1 \rightarrow l_2], l) \equiv_{\alpha} (s', h' \uplus [l'_1 \rightarrow l'_2], l') \);
4. \( (s, h_0 \uplus h \uplus [l_1 \rightarrow l_2], l) \equiv_{\alpha} (s', h'_0 \uplus h' \uplus [l'_1 \rightarrow l'_2], l') \).

The proof can be found in Appendix B starting at page 63.

Corollary 3.30 Let \( \alpha \geq 1 \). Let \( h_0 \perp h_1, \text{dom}(h_1) \subseteq s(V) \) and \( (s, h_0, l) \equiv_{p+\alpha} (s', h'_0, l') \) with \( p = \text{card}(\text{dom}(h_1)) \). Then there exists a heap \( h'_1 \) such that \( h_0 \perp h'_1, \text{dom}(h'_1) \subseteq s'(V), (s, h_1, l) \equiv_{\alpha} (s', h'_1, l'), (s, h_0 \uplus h_1, l) \equiv_{\alpha} (s', h'_0 \uplus h'_1, l') \) and \( \text{maxval}(s', h'_1) \leq \text{maxval}(s', h'_0, l') + p \).

The proof is by induction on the cardinality of \( \text{dom}(h_1) \) using Proposition 3.29. Proposition 3.31 below is a slight variant of Proposition 3.29.

Proposition 3.31 Let \( \alpha \geq 1 \). We assume that the following conditions hold:

(a) \( m \equiv_{\alpha} m' \);
(b) \( m_0 \equiv_{\alpha} m'_0 \);

Let \( l_1 \notin \text{dom}(h_0 \cup h) \cup s(V) \) and \( l_2 \in \mathbb{N} \). There exist \( l'_1, l'_2 \in \mathbb{N} \) such that

1. \( l'_1 \notin \text{dom}(h'_0 \cup h') \cup s'(V) \);
2. \( l'_1, l'_2 \leq \text{maxval}(m'_0) + 2 \);
3. \( (s, h \uplus [l_1 \rightarrow l_2], l) \equiv_{\alpha} (s', h' \uplus [l'_1 \rightarrow l'_2], l') \);
4. \( (s, h_0 \uplus h \uplus [l_1 \rightarrow l_2], l) \equiv_{\alpha} (s', h'_0 \uplus h' \uplus [l'_1 \rightarrow l'_2], l') \).

The proof can be found in Appendix B starting at page 64.

Corollary 3.32 Let \( \alpha \geq 1 \). Let \( \text{dom}(h_1) \cap (\text{dom}(h_0 \cup h) \cup s(V)) = \emptyset, (s, h, l) \equiv_{\alpha} (s', h', l') \) and \( (s, h_0 \uplus h, l) \equiv_{\alpha} (s', h'_0 \uplus h', l') \). Then there exists a heap \( h'_1 \) such that \( \text{dom}(h'_1) \cap (\text{dom}(h'_0 \cup h') \cup s'(V)) = \emptyset, (s, h \uplus h_1, l) \equiv_{\alpha} (s', h'_0 \uplus h' \uplus h'_1, l') \) and \( \text{maxval}(s', h'_1) \leq \text{maxval}(s', h'_0 \uplus h', l') + 2 \cdot \text{card}(\text{dom}(h_1)) \).

The proof is by induction on the cardinality of \( \text{dom}(h_1) \) using Proposition 3.31.
3.9 Correctness of the Abstraction

Lemmas 3.33, 3.34 and 3.35 below roughly state that the relation $\simeq_\alpha$ (and therefore the set of test formulæ we have introduced) behaves properly. Each lemma corresponds to a given quantifier, respectively separating conjunction $\ast$, separating implication $\rightarrow$ and first-order quantifier $\exists \alpha$. We combine these three lemmas in the proof of Correctness Theorem 4.3.

Lemma 3.33 below states how two equivalent memory states can be split.

The precision is split accordingly.

**Lemma 3.33 (Distributivity)** Let $\alpha, \alpha_1, \alpha_2 \geq 1$ such that $\alpha = \alpha_1 + \alpha_2$. Let us consider two $\alpha$-equivalent pointed memory states $(s, h, l)$ and $(s', h', l')$, i.e. $(s, h, l) \simeq_\alpha (s', h', l')$. For every split $h = h_1 \uplus h_2$ of $h$, there exists a split $h' = h'_1 \uplus h'_2$ of $h'$ such that $(s, h_1, l) \simeq_\alpha (s', h'_1, l')$ and $(s, h_2, l) \simeq_\alpha (s', h'_2, l')$.

**Proof** Let $m = (s, h, l)$ and $m' = (s', h', l')$ be such that $(s, h, l) \simeq_\alpha (s', h', l')$. Let us denote by $\mathcal{R}^l$ (resp. $\mathcal{T}$) the relation $\mathcal{R}^{l,m'}_m$ (resp. $\mathcal{T}^{l,m'}_m$) from Definition 3.14. From $m \simeq_\alpha m'$ we deduce $m \simeq_\alpha m'$ and then by Theorem 3.17, we have the inclusion $\mathcal{R}^l \subseteq \mathcal{T}$. Moreover by Lemma 3.19, we know that $\mathcal{R}^l_{s,h} \equiv \mathcal{R}^l \uplus \Diamond(s, h) \times \Diamond(s', h')$ is a bijection relation between $\Diamond(s, h)$ and $\Diamond(s', h')$.

Let us define $J = \{ j \in [1, q] \mid \text{for all } k \in [1, q], s(x_j) = s(x_k) \text{ implies } j \leq k \}$. Since $s(x_j) \mathcal{R}^l s'(x_i)$ and $\mathcal{R}^l \subseteq \mathcal{T}$, using (T2) we deduce $s(x_j) = s(x_j)$ if $s'(x_i) = s'(x_j)$ for all $i, j \in [1, q]$. Hence $J$ is a subset of $[1, q]$ that verifies

(J1) for any $i \in [1, q]$, there exists $j \in J$ such that $s(x_i) = s(x_j)$ and $s'(x_i) = s'(x_j)$;

(J2) for all $i, j \in J$, $s(x_i) = s(x_j)$ or $s'(x_i) = s'(x_j)$ implies $i = j$.

For every $c \in \{1, 2\}$ and for every $j \in J$, let us consider the following notations:

\[
\begin{align*}
D &= \text{dom}(h) & D_c &= \text{dom}(h_c) & D' &= \text{dom}(h') \\
C &= \Diamond(s, h) & C_c &= C \cap D_c & C' &= \Diamond(s', h') \\
P(j) &= \text{pred}_{\Diamond}(s, h, j) & P_c(j) &= P(j) \cap D_c & P'(j) &= \text{pred}_{\Diamond}(s', h', j) \\
L &= \text{loop}_{\Diamond}(s, h) & L_c &= L \cap D_c & L' &= \text{loop}_{\Diamond}(s', h') \\
R &= \text{rem}_{\Diamond}(s, h) & R_c &= R \cap D_c & R' &= \text{rem}_{\Diamond}(s', h')
\end{align*}
\]

According to Lemma 2.6 and Properties (J1) and (J2), we have the following canonical decompositions:

\[
D = C \uplus \biguplus_{j \in J} P(j) \uplus L \uplus R \quad D' = C' \uplus \biguplus_{j \in J} P'(j) \uplus L' \uplus R'
\]

We know that $\mathcal{R}_{C_2}$ is a one-to-one relation between $C$ and $C'$, and from Proposition 3.10, we have $P(j) \sim_\alpha P'(j)$, $L \sim_\alpha L'$ and $R \sim_\alpha R'$.

Using the bijection $\mathcal{R}_{C_2}$, for $c = 1$ or 2, let us define $C'_c = \mathcal{R}_{C_2}(C_c)$. Then, we have $C = C_1 \uplus C_2$ and $C' = C'_1 \uplus C'_2$. Let us show that $l/l'$ respect both $C_1/C'_1$ and $C_2/C'_2$. We have $l \mathcal{R}^l l'$ by definition and hence $l \mathcal{T} l'$. By (T13), we deduce that $l/l'$ respect $C/C'$. Hence if $l \in C_1$ then $l \in C$ and thus $l' \in C'$.
As a consequence, \( l \mathcal{R}_{c} l' \) and thus as \( C'_{1} = \mathcal{R}_{c}(C_{1}) \), we deduce \( l' \in C'_{1} \). For similar reasons, if \( l \in C_{2} \) then \( l' \in C'_{2} \). Now, if \( l' \in C'_{1} \) then \( l' \in C'_{2} \) hence \( l \in C = C_{1} \cup C_{2} \). The case \( l \in C_{2} \) would lead to \( l' \in C'_{1} \) hence \( l' \in C'_{1} \cap C'_{2} \) which is impossible. Hence we have \( l \in C_{1} \). For similar reasons, if \( l' \in C'_{2} \) then \( l \in C_{2} \) We conclude that \( l/l' \) respect both \( C_{1}/C'_{1} \) and \( C_{2}/C'_{2} \).

Let us verify that \( l/l' \) respect both \( P'(j)/P_{l}(j) \): \( l \in P(j) \) iff \( h(l) = s(x_{j}) \) and \( l \notin \ominus(s,h) \) iff \( h'(l') = s'(x_{j}) \) and \( l' \notin \ominus(s',h') \) iff \( l' \in P'(j) \) using \( l \mathcal{T}(l') \) with (65) and (13). By Lemma 2.22, from \( P(j) \approx_{\alpha_{1}+\alpha_{2}} P'(j) \) and \( \alpha_{1}, \alpha_{2} \geq 1 \), we compute \( P_{l}(j) \) such that \( P'(j) = P_{l}(j) \cup P_{l}'(j) \), \( P_{l}(j) \approx_{\alpha_{1}} P_{l}'(j) \), \( P_{l}(j) \approx_{\alpha_{2}} P_{l}'(j) \) and \( l/l' \) respect both \( P_{l}(j)/P_{l}'(j) \) and \( P_{l}(j)/P_{l}'(j) \).

By a similar argument, we get \( L'_{1} \) and \( L'_{2} \) (resp. \( R'_{1} \) and \( R'_{2} \)) such that \( L' = L_{1} \cup L_{2}, L_{1} \approx_{\alpha_{1}} L'_{1}, L_{2} \approx_{\alpha_{2}} L'_{2} \) (resp. \( R' = R'_{1} \cup R'_{2}, R_{1} \approx_{\alpha_{1}} R'_{1}, R_{2} \approx_{\alpha_{2}} R'_{2} \)) and \( l/l' \) respect both \( L_{1}/L'_{1} \) and \( L_{2}/L'_{2} \) (resp. \( R_{1}/R'_{1} \) and \( R_{2}/R'_{2} \)).

Now let us define a partition \( D' = D'_{1} \cup D'_{2} \) by

\[
D'_{1} = C_{1} \cup \bigcup_{j \in J} P_{l}(j) \cup L_{1} \cup R_{1} \quad D'_{2} = C_{2} \cup \bigcup_{j \in J} P_{l}'(j) \cup L_{2} \cup R_{2}
\]

and \( h'_{1}, h'_{2} \) such that \( h'_{1} = h'_{2} \), \( \text{dom}(h'_{1}) = D'_{1} \) and \( \text{dom}(h'_{2}) = D'_{2} \).

We point out that the defining equation of \( D'_{c} \) is not necessarily the canonical decomposition of \( D'_{c} \) according to \((s', h'_{c})\). Since the identities

\[
D_{1} = C_{1} \cup \bigcup_{j \in J} P_{l}(j) \cup L_{1} \cup R_{1} \quad D_{2} = C_{2} \cup \bigcup_{j \in J} P_{l}(j) \cup L_{2} \cup R_{2}
\]

hold, we observe that \( l/l' \) respect both \( D_{1}/D'_{1} \) and \( D_{2}/D'_{2} \).

Using Proposition 3.22, we check that the basic equivalences \((s, h_{1}, l) \approx_{b} (s', h'_{1}, l')\) and \((s, h_{2}, l) \approx_{b} (s', h'_{2}, l')\) hold. For \( c = 1 \) or \( c = 2 \), let us prove \( \mathcal{R}_{c} \subseteq D_{c} \):

- we already proved that \( l/l' \) respect \( \text{dom}(h_{c})/\text{dom}(h'_{c}) \);
- if \( s(x_{j}) \in \text{dom}(h_{c}) \) then \( s(x_{j}) \in \text{dom}(h_{c}) \) and thus \( s(x_{j}) \in C \). We derive \( s(x_{j}) \in C_{c} = C \cap D_{c} \). Since \( s(x_{j}) \in \ominus(s, h) \) we derive \( s'(x_{j}) \in \ominus(s', h') \) using \( s(x_{j}) \mathcal{T}(s'(x_{j})) \) and (13). Thus \( s(x_{j}) \mathcal{R}_{c} s'(x_{j}) \) holds and we deduce \( s'(x_{j}) \in C'_{c} \) hence \( s'(x_{j}) \in D'_{c} = \text{dom}(h'_{c}) \). The reverse implication \( "s'(x_{j}) \in \text{dom}(h'_{c})" \) implies \( s(x_{j}) \in \text{dom}(h_{c})\) is proved by symmetric arguments;
- if \( h(s(x_{j})) \in \text{dom}(h_{c}) \) then \( h(s(x_{j})) \in \text{dom}(h_{c}) \) and thus \( h(s(x_{j})) \in C \). We derive \( h(s(x_{j})) \in C_{c} = C \cap D_{c} \). From \( h(s(x_{j})) \mathcal{R}_{c} h'(s'(x_{j})) \), we deduce \( h'(s'(x_{j})) \in C'_{c} \) hence \( h'(s'(x_{j})) \in D'_{c} = \text{dom}(h'_{c}) \). The reverse implication \( "h'(s'(x_{j})) \in \text{dom}(h'_{c})" \) implies \( h(s(x_{j})) \in \text{dom}(h_{c})\) is proved by symmetric arguments.

Hence we have \((s, h, l) \approx_{b} (s', h', l'), (s, h_{1}, l) \approx_{b} (s', h'_{1}, l')\) and \((s, h_{2}, l) \approx_{b} (s', h'_{2}, l')\). According to the above definitions and Proposition 2.9, we get the following identities:

\[
pred_{p}(s, h, j) = P_{l}(j) \cup \{ \text{pred}(s, h, j) \cap \Delta(s, h_{c}, h_{3-c}) \}
\]

\[
pred_{p}(s', h', j) = P_{l}'(j) \cup \{ \text{pred}(s', h', j) \cap \Delta(s', h'_{c}, h'_{3-c}) \}
\]

According to Proposition 3.23 item 1 we have

\[
\text{pred}(s, h, j) \cap \Delta(s, h_{c}, h_{3-c}) \approx_{\infty} \text{pred}(s', h', j) \cap \Delta(s', h'_{c}, h'_{3-c})
\]
For any \( j \in J \), from \( P_c(j) \sim_{\alpha_c} P'_c(j) \) we get \( \text{pred}(s, h_c, j) \sim_{\alpha_c} \text{pred}(s', h'_c, j) \) by Lemma 2.19. In fact we get more precision but we do not need it here.

By similar arguments using Proposition 3.23 items 2 and 3, we establish \( \text{loop}(s, h_c) \sim_{\alpha_c} \text{loop}(s', h'_c) \) and \( \text{rem}(s, h_c) \sim_{\alpha_c} \text{rem}(s', h'_c) \).

Using Properties (J1), we have \( \text{pred}(s, h_c, i) \sim_{\alpha_c} \text{pred}(s', h'_c, i) \) for any \( i \in [1, q] \). Hence, by Proposition 3.10, we deduce \((s, h_1, l) \equiv_{\alpha_1} (s', h'_1, l')\) and \((s, h_2, l) \equiv_{\alpha_2} (s', h'_2, l')\). □

Lemma 3.34 below states how to extend a memory state with a heap while preserving equivalence. Some precision (not exceeding \( q \)) is lost in the process.

**Lemma 3.34 (Compositionality)** Let us consider \( \alpha \geq 1 \), two pointed memory states \((s, h_0, l)\) and \((s', h'_0, l')\) such that \((s, h_0, l) \equiv_{q+\alpha} (s', h'_0, l')\). For any \( h \) such that \( h \perp h_0 \) there exists \( h' \) such that \( h' \perp h'_0 \) and

1. \((s, h, l) \equiv_{\alpha} (s', h', l')\);
2. \((s, h_0 \uplus h, l) \equiv_{\alpha} (s', h'_0 \uplus h', l')\);
3. \(\text{maxval}(s', h') \leq \text{maxval}(s', h'_0, l') + (2\alpha + 3)(q + 2) - 4\).

**Proof** Let \( J = \{ j \in [1, q] \mid \forall k \in [1, q], s(x_j) = s(x_k) \implies j \leq k \} \). So \( J \) is a subset of \([1, q]\) that verifies:

(J1) for any \( i \in [1, q] \), there exists \( j \in J \) such that \( s(x_i) = s(x_j) \);

(J2) for all \( i, j \in J, s(x_i) = s(x_j) \) implies \( i = j \).

We define set following subsets of \( \text{dom}(h) \):

- \( S = \text{dom}(h) \cap s(V) \);
- \( H = (\text{dom}(h) \cap \mathcal{P}(s, h_0 \uplus h, l)) \setminus s(V) \);
- \( \text{loop}(s, h, j) \mathcal{P}(s, h_0 \uplus h, l) \) and \( P_j \sim_{\alpha} P_j \cup P'_j \) and \( \text{card}(P_j) \leq \alpha \) for any \( j \in J \);
- \( L \uplus L' = \text{loop}(s, h) \mathcal{P}(s, h_0 \uplus h, l) \) and \( L \sim_{\alpha} L \uplus L' \) and \( \text{card}(L) \leq \alpha \);
- \( R \uplus R' = \text{rem}(s, h) \mathcal{P}(s, h_0 \uplus h, l) \) and \( R \sim_{\alpha} R \uplus R' \) and \( \text{card}(R) \leq \alpha \).

where \( (P_j \cup P'_j)_{j \in J}, L/L' \) and \( R/R' \) are obtained using Proposition 2.24 and from e.g. \( L \sim_{\alpha} L \uplus L' \) and \( \text{card}(L) \leq \alpha \) we deduce either \( \text{card}(L) = \alpha \) or \( L' = \emptyset \). Let us check that

\[
\text{dom}(h) = S \uplus H \uplus \bigcup_{j \in J} (P_j \cup P'_j) \uplus (L \uplus L') \uplus (R \uplus R')
\]  

(3.1)

is indeed a partition of the domain of \( h \). Obviously \( S \uplus H = \text{dom}(h) \cap \mathcal{P}(s, h_0 \uplus h, l) \). Then \( \text{pred}(s, h) = \bigcup_{j \in J} \text{pred}(s, h, j) \) because of Properties (J1) and (J2).

From \( \text{dom}(h) = (\text{pred}(s, h) \cup \text{loop}(s, h)) \uplus \text{rem}(s, h) \), we deduce

\[
\text{dom}(h) \mathcal{P}(s, h_0 \uplus h, l) = (\bigcup_{j \in J} (P_j \cup P'_j) \cup (L \uplus L')) \uplus (R \uplus R')
\]

Then the only remaining point is to show that \( (P_j \cup P'_j) \cap (L \uplus L') = \emptyset \). If \( u \in (P_j \cup P'_j) \cap (L \uplus L') \) then we have \( u \in \text{pred}(s, h, j) \) and \( u \in \text{loop}(s, h) \).
Hence \( h(u) = s(x_j) \) and \( h(u) = u \). We deduce \( u = s(x_j) \in p^\ominus(s, h_0 \vdash h, l) \) which contradicts \( u \in L \cup L' \).

We observe that \( \text{card}(S) \leq q \) because \( S \subseteq s(V) \) and that \( \text{card}(H) \leq q + 1 \) because \( H \subseteq (h_0 \vdash h)(s(V)) \cup \{l\} \). Let us define

- \( h_1 \) as the restriction of \( h \) to \( S \), i.e. \( h_1 \subseteq h \) and \( \text{dom}(h_1) = S \);
- \( h_2 \) as the restriction of \( h \) to \( H \cup \bigcup_j P_j \cup L \cup R \);
- \( h_3 \) as the restriction of \( \bigcup_j P_j \cup L' \cup R' \).

Then we have \( h = h_1 \cup h_2 \cup h_3 \), \( \text{card}(\text{dom}(h_1)) \leq q \), and \( \text{card}(\text{dom}(h_2)) \leq \text{card}(H) + \sum_{j \in J} \text{card}(P_j) + \text{card}(L) + \text{card}(R) \leq (q + 1) + q + \alpha + \alpha + \alpha = (\alpha + 1)(q + 2) - 1 \).

Let us write \( p = \text{card}(S) = \text{card}(\text{dom}(h_1)) \leq q \) and \( m = \text{maxval}(s', h_0', l') \).

We deduce \( (s, h_0, l) \approx_{p+\alpha} (s', h_0', l') \). By Corollary 3.30, we get \( h_1' \) such that:

- \( \text{dom}(h_1') \subseteq s'(V) \);
- \( \text{maxval}(s', h_1') \leq \text{maxval}(s', h_0', l') + p \);
- \( (s, h_1, l) \approx_{\alpha} (s', h_1', l') \);
- \( (s, h_0 \vdash h_1, l) \approx_{\alpha} (s', h_0' \vdash h_1', l') \).

We deduce \( \text{maxval}(s', h_1') \leq m + q \) and thus also \( \text{maxval}(s', h_0' \vdash h_1') \leq m + q \).

Then we use Corollary 3.32 for \( h_2 \). We have indeed \( \text{dom}(h_2) \cap (\text{dom}(h_0 \vdash h_1) \cup s(V)) \subseteq \text{dom}(h_2) \cap (\text{dom}(h_0) \cup s(V)) \subseteq (\text{dom}(h_2) \cap s(V)) \subseteq \emptyset \cup \emptyset \subseteq \emptyset \), \( (s, h_1, l) \approx_{\alpha} (s', h_1', l') \) and \( (s, h_0 \vdash h_1, l) \approx_{\alpha} (s', h_0' \vdash h_1', l') \). We obtain a heap \( h_2' \) such that:

- \( \text{dom}(h_2') \cap (\text{dom}(h_0' \vdash h_1') \cup s'(V)) = \emptyset \);
- \( \text{maxval}(s', h_2') \leq \text{maxval}(s', h_0' \vdash h_1', l') + 2 \cdot \text{card}(\text{dom}(h_2)); \)
- \( (s, h_1 \vdash h_2, l) \approx_{\alpha} (s', h_1' \vdash h_2', l') \);
- \( (s, h_0 \vdash h_1 \vdash h_2, l) \approx_{\alpha} (s', h_0' \vdash h_1' \vdash h_2', l') \).

We deduce \( \text{maxval}(s', h_2') \leq (m + q) + 2((\alpha + 1)(q + 2) - 1) = m + 2(\alpha + 3)(q + 2) - 4 \) and thus also \( \text{maxval}(s', h_0' \vdash h_2') \leq m + (2\alpha + 3)(q + 2) - 4 \).

Then, we use Corollary 3.27 to show that \( (s, h_1 \vdash h_2 \vdash h_3, l) \approx_{\alpha} (s, h_1 \vdash h_2, l) \) holds. By construction of \( h_3 \), it is clear that \( \text{dom}(h_3) \cap (\text{dom}(h_1 \vdash h_2) \cup p^\ominus(s, h_1 \vdash h_2)) = \emptyset \) because \( p^\ominus(s, h_1 \vdash h_2, l) \subseteq p^\ominus(s, h_0 \vdash h, l) \). It is thus sufficient to verify either (C1) or (C2) or (C3) for any \( l_1 \in \text{dom}(h_3) = \bigcup_j P_j \cup L' \cup R' \). We have three cases for \( l_1 \in \text{dom}(h_3) \):

- if \( l_1 \in P_j \) for some \( j \in J \). Then \( h(l_1) = s(x_j) \) and thus \( h_3(l_1) = s(x_j) \). Moreover \( P_j \neq \emptyset \) and thus we must have \( \text{card}(P_j) = \alpha \) (because \( P_j \cup P_j \cup \text{card}(P_j) \leq \alpha \)). Let us prove \( P_j \subseteq \text{pred}(s, h_1 \vdash h_2, j) \). We have \( P_j \subseteq (\text{pred}(s, h, j) \setminus p^\ominus(s, h_0 \vdash h, l)) \cup \text{dom}(h_2) \) hence we deduce \( P_j \subseteq (\text{pred}(s, h, j) \cap \text{dom}(h_2)) \setminus p^\ominus(s, h_0 \vdash h, l) \). But \( \text{pred}(s, h, j) \cap \text{dom}(h_2) \subseteq \text{pred}(s, h_1 \vdash h_2, j) \) because \( h_2 \subseteq h_1 \vdash h_2 \subseteq h \); and \( p^\ominus(s, h_1 \vdash h_2, l) \subseteq p^\ominus(s, h_0 \vdash h, l) \) because \( h_1 \vdash h_2 \subseteq h_0 \vdash h \). We deduce \( P_j \subseteq \text{pred}(s, h_1 \vdash h_2, j) \setminus p^\ominus(s, h_1 \vdash h_2, j) \). We deduce \( \text{card}(\text{pred}(s, h_1 \vdash h_2, j)) \geq \alpha \). Condition (C1) holds for \( l_1 \);
if \( l_1 \in L' \) then \( h(l_1) = l_1 \) hence \( h_3(l_1) = l_1 \). Since \( L' \) is not empty, we have \( \text{card}(L) = \alpha \) and we show that \( L \subseteq \text{loop}_n(s, h_1 \uparrow h_2) \).

We have \( L \subseteq (\text{loop}(s, h) \setminus \text{dom}(h_2)) \) hence \( L \subseteq (\text{loop}(s, h) \cap \text{dom}(h_2)) \) but \( \text{loop}(s, h) \cap \text{dom}(h_2) \subseteq \text{loop}(s, h_1 \uparrow h_2) \) and \( \text{dom}(h_1 \uparrow h_2) \subseteq \text{dom}(h_2) \). We get \( L \subseteq \text{loop}(s, h_1 \uparrow h_2) \setminus \text{dom}(h_2) \). We deduce \( \text{card}(\text{loop}_n(s, h_1 \uparrow h_2)) \geq \alpha \). Condition (C2) holds for \( l_1 \).

if \( l_1 \in R' \) then \( h(l_1) \notin s(\mathcal{V}) \cup \{ l_1 \} \) hence \( h_3(l_1) \notin s(\mathcal{V}) \cup \{ l_1 \} \). Since \( R' \) is not empty, we have \( \text{card}(R) = \alpha \) and we show that \( R \subseteq \text{rem}_n(s, h_1 \uparrow h_2) \).

We have \( R \subseteq (\text{rem}(s, h) \setminus \text{dom}(h_2)) \) hence \( R \subseteq (\text{rem}(s, h) \cap \text{dom}(h_2)) \) but \( \text{rem}(s, h) \cap \text{dom}(h_2) \subseteq \text{rem}(s, h_1 \uparrow h_2) \) and \( \text{dom}(h_1 \uparrow h_2) \subseteq \text{dom}(h_2) \). We get \( R \subseteq \text{rem}(s, h_1 \uparrow h_2) \setminus \text{dom}(h_2) \). We deduce \( \text{card}(\text{rem}_n(s, h_1 \uparrow h_2)) \geq \alpha \). Condition (C3) holds for \( l_1 \).

Hence, by Corollary 3.27, we deduce that \((s, h_1 \uparrow h_2 \uparrow h_3, l) \equiv_\alpha (s, h_1 \uparrow h_2, l) \) holds. By similar arguments, we show that \((s, h_0 \uparrow h_1 \uparrow h_2 \uparrow h_3, l) \equiv_\alpha (s, h_0 \uparrow h_1 \uparrow h_2, l) \) holds as well.

Let us finally show that \( h' = h'_1 \uparrow h'_2 \) satisfies the required conditions. We have already proved \( \text{maxval}(s', h') \leq \text{maxval}(s', h'_0, l') + (2\alpha + 3)(q + 2) - 4 \).

Then we have \((s, h, l) = (s, h_1 \uparrow h_2 \uparrow h_3, l) \equiv_\alpha (s, h_1 \uparrow h_2, l) \equiv_\alpha (s', h'_1 \uparrow h'_2, l') = (s', h', l') \) and \((s, h_0 \uparrow h_3, l) = (s, h_0 \uparrow h_1 \uparrow h_2 \uparrow h_3, l) \equiv_\alpha (s, h_0 \uparrow h_1 \uparrow h_2, l) \equiv_\alpha (s', h'_0 \uparrow h'_1 \uparrow h'_2, l') = (s', h'_0 \uparrow h'_1 \uparrow h'_2, l'). \)

\[ \square \]

Lemma 3.35 below states how to update the location of the quantified variable while preserving equivalence. No precision is lost here.

Lemma 3.35 (Existence) Let \( \alpha \geq 1 \) and \((s, h, l)\) and \((s', h', l')\) be two \( \alpha \)-equivalent pointed memory states, i.e. \((s, h, l) \equiv_\alpha (s', h', l')\). For every \( l_0 \in \mathbb{N} \), there exists \( l'_0 \leq \text{maxval}(s', h') + 1 \) such that \((s, h, l_0) \equiv_\alpha (s', h', l'_0)\).

Proof By Proposition 3.10, we have \((s, h, l) \equiv_\alpha (s', h', l')\) together with three \( \alpha \)-equivalence constraints: \( \text{pred}_{\alpha}(s, h, i) \sim_\alpha \text{pred}_{\alpha}(s', h', i) \) for any \( i \in [1, q] \), \( \text{loop}_{\alpha}(s, h) \sim_\alpha \text{loop}_{\alpha}(s', h') \), and \( \text{rem}_{\alpha}(s, h) \sim_\alpha \text{rem}_{\alpha}(s', h') \).

Let \( l_0 \in \mathbb{N} \). Let us consider the relation \( \mathbb{T}_{m, m'} \) of Definition 3.14 where \( m = (s, h, l) \) and \( m' = (s', h', l') \). By Proposition 3.21, there exists \( l'_0 \in \mathbb{N} \) such that \( l'_0 \leq \text{maxval}(s', h') + 1 \) and \( l_0 \mid \mathbb{T}_{m, m'} \mid l'_0 \) holds. By definition of \( \mathbb{T}_{m, m'} \), this means that \( l_0/l'_0 \) verifies \( (\mathbb{T}_2-6) \). Hence by Proposition 3.24, we get \((s, h, l_0) \equiv_\alpha (s', h', l'_0)\), and by Proposition 3.10, we conclude \((s, h, l_0) \equiv_\alpha (s', h', l'_0)\). \[ \square \]

4 Decidability, Expressiveness and Complexity

In this section, we show that two \( \alpha \)-equivalent pointed memory states cannot be distinguished by 1SL1 formulae of memory threshold less than \( \alpha \). We introduce model compression results and deduce that the unbounded quantifications used in the definitions of \((s, h) \models_1 \exists u \mathcal{A}\) and \((s, h) \models_1 \mathcal{A} \ightarrow \mathcal{B}\) can be replaced by bounded quantifications. We then derive decidability results for the model-checking and satisfiability problems in 1SL1 and a quantifier
elimination result for 1SL1: any formula of 1SL1 is equivalent to a Boolean combination of test formulæ in Test\(q\), for some threshold \(\alpha \geq 1\). Then we provide a \(\mathsf{pspace}\) complexity characterization for both model-checking and satisfiability in 1SL1 using a bounded model-checking algorithm.

4.1 Correctness of the Abstraction

Given the three previous results, Lemma 3.33 for distributivity, Lemma 3.34 for compositionality and Lemma 3.35 for existence, we design a notion of memory threshold that matches the loss of precision induced by the separating conjunction, the separating implication and the first-order quantification.

**Definition 4.1 (Memory Threshold)** Given \(q \geq 1\) and an 1SL1 formula \(\mathcal{A}\) built over the program variables \(x_1, \ldots, x_q\), we define its memory threshold \(\text{th}(q, \mathcal{A})\) inductively as follows:

\[
\begin{align*}
\text{th}(q, \neg \mathcal{A}_1) & \equiv \text{th}(q, \mathcal{A}_1) \\
\text{th}(q, \mathcal{A}_1 \land \mathcal{A}_2) & \equiv \max(\text{th}(q, \mathcal{A}_1), \text{th}(q, \mathcal{A}_2)) \\
\text{th}(q, \mathcal{A}_1 \ast \mathcal{A}_2) & \equiv \text{th}(q, \mathcal{A}_1) + \text{th}(q, \mathcal{A}_2) \\
\text{th}(q, \mathcal{A}_1 \rightarrow \mathcal{A}_2) & \equiv q + \max(\text{th}(q, \mathcal{A}_1), \text{th}(q, \mathcal{A}_2)) \\
\text{th}(q, \mathcal{A}_1) & \equiv 1 \\
\end{align*}
\]

For instance \(\text{th}(3, (x_1 \rightarrow x_1) \rightarrow \bot) = 3 + \max(1, 1) = 4\). The rationale for these inductive definitions comes from the proof of the upcoming correctness result and how Lemmas 3.33, 3.34 and 3.35 are used there. In the case of \(\mathcal{A}_1 \ast \mathcal{A}_2\), the use of the Distributivity Lemma 3.33 implies that precision is split in half. In the case of \(\mathcal{A}_1 \rightarrow \mathcal{A}_2\), the use of the Compositionality Lemma 3.34 implies a loss of precision bounded by \(q\).

**Proposition 4.2** Given \(q \geq 1\) and a formula \(\mathcal{A}\) in 1SL1 with program variables \(x_1, \ldots, x_q\), we have \(1 \leq \text{th}(q, \mathcal{A}) \leq q, |\mathcal{A}|\).

The proof is left to the reader.

Now we state the correctness result which means that test formulæ in Test\(q\) provide the proper abstraction for the formulæ of 1SL1 with a memory threshold bounded by \(\alpha\).

**Theorem 4.3 (Abstraction Correctness)** Let \(q \geq 1\). For any 1SL1 formula \(\mathcal{A}\) with program variables \(x_1, \ldots, x_q\), for any \(\alpha \geq 1\), if \(\text{th}(q, \mathcal{A}) \leq \alpha\) and \((s, h, l) \models_\alpha (s', h', l')\) hold then \((s, h) \models_{\ell} \mathcal{A}\) iff \((s', h') \models_{\ell'} \mathcal{A}\).

*Proof* The proof is by induction on the structure of \(\mathcal{A}\). Suppose that \((s, h, l) \models_\alpha (s', h', l')\) and \(\mathcal{A}\) be a formula with \(\text{th}(q, \mathcal{A}) \leq \alpha\). By structural induction, we show that \((s, h) \models_{\ell} \mathcal{A}\) if and only if \((s', h') \models_{\ell'} \mathcal{A}\) but we only display the proof of the only if implication, the converse implication is obtained by symmetry.

- \(\mathcal{A}\) is \(e \mapsto e'\) which is covered by one of the following cases: \(x_i \mapsto x_j\), \(x_i \mapsto u\), \(u \mapsto x_i\) and \(u \mapsto u\). All of these formulæ belong to \(\mathsf{Basic}^\alpha \subseteq \text{Test}^q\).
\(A\) is \(e = e'\) which is covered by one of the following cases: \(x_i = x_j, x_i = u, u = x_i\) and \(u = u\). The two first belong to \(\text{Basic}_a \subseteq \text{Test}_a\), \(u = x_i\) is logically equivalent to \(u = u\) and \(u = u\) is a tautology;

\(A\) is \(\text{emp}\) which is logically equivalent to

\[-(\forall, \text{conv}(x_i, x_i) \lor \forall, \text{talloc}(x_i) \lor \forall, \# \text{pred}_{\alpha}(x_i) \geq 1 \lor \# \text{loop}_{\alpha} \geq 1 \lor \# \text{rem}_{\alpha} \geq 1)\]

by the canonical decomposition of Lemma 2.6. This formula is a Boolean combination of formulae of \(\text{Test}_a \subseteq \text{Test}_a\). Remember that \(\text{conv}(x_i, x_i)\) is in place of \(\text{alloc}(x_i)\);

\(A\) is \(\exists u, A_1\) with \((s, h) \models_1 \exists u, A_1\) and \((s, h, l) \models_\alpha (s', h', l')\) and \(\text{th}(q, \exists u, A_1) \leq \alpha\). There exists \(l_0\) such that \((s, h) \models_{l_0} A_1\). By Lemma 3.35, there is \(l_1\) such that \((s, h, l_0) \models_\alpha (s', h', l_1)\). Since we have \((s, h) \models_{l_0} A_1\), by induction hypothesis we get \((s', h') \models_{l_1} A_1\) (note that \(\text{th}(q, A_1) = \text{th}(q, \exists u, A_1) \leq \alpha)\). Thus we conclude \((s', h') \models_{\nu} \exists u, A_1\);

\(A\) is \(A_1 \land A_2\) with \((s, h) \models_1 A_1 \land A_2\), \((s, h, l) \models_\alpha (s', h', l')\), \(\text{th}(q, A_1 \land A_2) \leq \alpha\). There are heaps \(h_1\) and \(h_2\) such that \(h = h_1 \uplus h_2\) and \((s, h_1) \models_1 A_1\) and \((s, h_2) \models_1 A_2\). As \(\alpha \geq \text{th}(q, A_1) + \text{th}(q, A_2)\), there exist \(\alpha_1\) and \(\alpha_2\) such that \(\alpha = \alpha_1 + \alpha_2\) and \(\alpha_1 \geq \text{th}(q, A_1)\) and \(\alpha_2 \geq \text{th}(q, A_2)\). By Lemma 3.33, there exist heaps \(h'_1\) and \(h'_2\) such that \(h' = h'_1 \uplus h'_2\) and \((s, h_1, l) \models_\alpha (s', h'_1, l')\) and \((s, h_2, l) \models_\alpha (s', h'_2, l')\). By the induction hypothesis, we get \((s', h'_1) \models_\nu A_1\) and \((s', h'_2) \models_\nu A_2\) (since \(\text{th}(q, A_1) \leq \alpha_1\) and \(\text{th}(q, A_2) \leq \alpha_2\)). Consequently we obtain \((s', h') \models_\nu A_1 \land A_2\);

\(A\) is \(A_1 \rightarrow A_2\) with \((s, h) \models_1 A_1 \rightarrow A_2\) and \((s, h, l) \models_\alpha (s', h', l')\) and

\[q + \beta = \text{th}(q, A_1 \rightarrow A_2) \leq \alpha\] with \(\beta = \max(\text{th}(q, A_1), \text{th}(q, A_2))\)

We deduce \((s, h, l) \models_{\nu + \beta} (s', h', l')\). Let us prove \((s', h') \models_\nu A_1 \rightarrow A_2\). We pick \(h'_1\) such that \(h'_1 \perp h'\) and \((s', h'_1) \models_\nu A_1\) and show that \((s', h' \uplus h'_1) \models_\nu A_2\). By Lemma 3.34, there is a heap \(h_1\) such that \(h_1 \perp h\) and \((s, h_1, l) \models_{\beta} (s', h'_1, l')\) and \((s, h \uplus h_1, l) \models_{\beta} (s', h' \uplus h'_1, l')\). We deduce that \((s, h_1) \models_1 A_1\) holds by the induction hypothesis (since \(\text{th}(q, A_1) \leq \beta\)). As \(h_1 \perp h\) and \((s, h) \models_1 A_1 \rightarrow A_2\) hold, we deduce \((s, h \uplus h_1) \models_1 A_2\). By induction hypothesis, we deduce \((s', h' \uplus h'_1) \models_\nu A_2\) (since \(\text{th}(q, A_2) \leq \beta\)). Hence we have \((s', h') \models_\nu A_1 \rightarrow A_2\).

The induction step for a Boolean outermost connective is straightforward and therefore this concludes the proof.

\(\square\)

4.2 Model Checking, Satisfiability and Expressive Completeness

Lemma 3.34 and the proof of Lemma 3.35 suggest a “compressor” lemma that can operate at any precision and compress either the location or the heap of a pointed memory state.

**Lemma 4.4 (Compressor)** Let \(q \geq 1\) and let \((s, h, l)\) be a pointed memory state. The two following statements hold:

...
1. there exists $l' \leq \maxval(s, h) + 1$ s.t. $(s, h, l) \approx_\alpha (s, h, l')$ holds for any $\alpha$;
2. for any $\alpha \geq 1$, any $h_1 \perp h$, there exists $h'_1 \perp h$ s.t. $(s, h_1, l) \approx_\alpha (s, h'_1, l)$ and $(s, h \uplus h'_1, l) \approx_\alpha (s, h \uplus h'_1, l)$ and $\maxval(h'_1) \leq \maxval(s, h, l) + 15q_0$.

Proof Let $m = (s, h, l)$ be a pointed memory state.

Let us start with Statement 1. Let us define $l'$ the following way: if $l \in \dom(h) \cup P \setminus (s, h)$ then $l' = l$; and $l' = \maxval(s, h) + 1$ if $l \notin \dom(h) \cup P \setminus (s, h)$. In the former case, we obviously have $l \models_{m,m} l'$ because $\mathcal{I}_{m,m}$ is reflexive. In the latter case, we get $l \models_{m,m} l'$ by Proposition 3.16 item 3. Hence $l/l'$ verify (T2–6) with respect to $m/m$. In both cases, the relation $l' \leq \maxval(s, h) + 1$ is obvious. Since $(s, h, l) \approx_b (s, h, l)$ holds by reflexivity and $l/l'$ verify (T2–6) then by Proposition 3.24, we get $(s, h, l) \approx_b (s, h, l')$. Let $\alpha \geq 0$. By Proposition 3.10, only the $\alpha$-equivalence constraints $\text{pred}_\alpha(s, h, i) \approx_\alpha \text{pred}_\alpha(s, h, i)$, $\text{loop}_\alpha(s, h, i) \approx_\alpha \text{loop}_\alpha(s, h, i)$ and $\text{rem}_\alpha(s, h, i) \approx_\alpha \text{rem}_\alpha(s, h, i)$ remain. By reflexivity, they hold trivially.

Now let us prove Statement 2. Let $\alpha \geq 1$ and $h_1 \perp h$. By reflexivity we have $(s, h, l) \approx_{q+\alpha} (s, h, l)$. Hence by Lemma 3.34, there exists a heap $h'_1$ such that $(s, h_1, l) \approx_\alpha (s, h'_1, l)$, $(s, h \uplus h_1, l) \approx_\alpha (s, h \uplus h'_1, l)$ and $\maxval(h'_1) \leq \maxval(s, h, l) + (2\alpha + 3)(q + 2) - 4$. As $\alpha \geq 1$ and $q \geq 1$, we get the relation $(2\alpha + 3)(q + 2) - 4 \leq 15q_0$. We deduce $\maxval(h'_1) \leq \maxval(s, h, l) + 15q_0$. □

We derive two corollaries that aim at replacing unbounded/infinite quantification with bounded/finite quantification in the respective definitions of $(s, h) \models_{l} \exists u \ A$ and $(s, h) \models_{l} A \rightarrow B$.

**Corollary 4.5** Let $q \geq 1$. For any pointed memory state $(s, h, l)$ and for any $1SL1$ formula $A$ with program variables in $x_1, \ldots, x_q$, we have:

$$(s, h) \models_{l} \exists u \ A \quad \text{iff} \quad \text{there is} \ l' \leq \maxval(s, h) + 1 \ \text{such that} \ (s, h) \models_{l'} \ A$$

**Proof** The if case is trivial. For the only if case, let us assume $(s, h) \models_{l} \exists u \ A$. Then there exists $l_0 \in \mathbb{N}$ such that $(s, h) \models_{l_0} \ A$. By Lemma 4.4, there exists $l' \leq \maxval(s, h) + 1$ such that $(s, h, l_0) \approx_\alpha (s, h, l')$ holds for any $\alpha$. Choosing e.g. $\alpha = \text{th}(q, A)$, we deduce $(s, h) \models_{l'} \ A$ by Theorem 4.3. □

**Corollary 4.6** Let $q \geq 1$. Let $(s, h, l)$ be pointed memory state and $A, B$ be two $1SL1$ formulae with program variables in $x_1, \ldots, x_q$. Let us define $m = \maxval(s, h, l) + 15|A| \rightarrow B | q^2$. Then $(s, h) \models_{l} A \rightarrow B$ if and only if

for any $h_1 \perp h$ with $\maxval(h_1) \leq m$, $(s, h_1) \models_{l} A$ implies $(s, h \uplus h_1) \models_{l} B$

**Proof** The only if case is trivial. For the if case, let us assume that $(s, h_1) \models_{l} A$ implies $(s, h \uplus h_1) \models_{l} B$ for any $h_1 \perp h$ such that $\maxval(h_1) \leq m$ and show that $(s, h) \models_{l} A \rightarrow B$ holds. Let us consider a heap $h'$ such that $h' \perp h$ and $(s, h') \models_{l} A$ and prove $(s, h \uplus h') \models_{l} B$. By Lemma 4.4, for $\alpha = q|A| \rightarrow B$ there exists a heap $h''$ such that $h'' \perp h$, $(s, h, l) \approx_\alpha (s, h'', l)$, $(s, h \uplus h'', l) \approx_\alpha (s, h \uplus h'', l)$ and $\maxval(h'') \leq \maxval(s, h, l) + 15q_0$. We deduce $\maxval(h'') \leq m$.

But we have $\text{th}(q, A) \leq q|A| \leq \alpha$. Hence by Theorem 4.3, from $(s, h) \models_{l} A$ and $(s, h', l) \approx_\alpha (s, h'', l)$ we deduce $(s, h'') \models_{l} A$. Then by hypothesis with $h_1$
as \(h^n\), we deduce \((s, h \uplus h^n) \models_1 B\). By Theorem 4.3, from \(\text{th}(q, B) \leq q|B| \leq \alpha\) and \((s, h \uplus h', l) \models_\alpha (s, h \uplus h^n, l)\) we conclude \((s, h \uplus h') \models_1 B\). \(\square\)

We also deduce a model compression result under \(\alpha\)-equivalence and then we derive a small model property as a consequence of Theorem 4.3.

**Corollary 4.7** Let \(q, \alpha \geq 1\). Every pointed memory state is \(\alpha\)-equivalent to a pointed memory state \((s, h, l)\) such that \(\text{maxval}(s, h, l) \leq 16\alpha q\).

**Proof** Let \((s_0, h_0, l_0)\) be a pointed memory state. The set \(s_0(V) \cup \{l_0\}\) is a subset of \(\mathbb{N}\) of cardinal less than \(q + 1\). Hence there exists a bijection \(\varphi: \mathbb{N} \rightarrow \mathbb{N}\) such that \(\varphi(s_0(V) \cup \{l_0\}) \subseteq [0, q]\). We define \(s = \varphi \circ s_0, h_1 = \varphi \circ h_0 \circ \varphi^{-1}\) and \(l = \varphi(l_0)\). By Proposition 2.3, the pointed memory states \((s_0, h_0, l_0)\) and \((s, h_1, l)\) do not distinguish any formula of 1SL1, hence we have \((s_0, h_0, l_0) \models_\alpha (s, h_1, l)\).

Now we consider the pointed memory state \((s, \Box, l)\). As \(\varphi(s_0(V) \cup \{l_0\}) \subseteq [0, q]\) we deduce \(s(V) \cup \{l\} \subseteq [0, q]\) hence \(\text{maxval}(s, \Box, l) \leq q\). We use Lemma 4.4 item 2 with \(\alpha \geq 1\) and \(h_1 \perp \Box\), hence there exists \(h\) such that \((s, h_1, l) \models_\alpha (s, h, l)\) and \((s, \Box \uplus h_1, l) \models_\alpha (s, \Box \uplus h, l)\) and \(\text{maxval}(h) \leq \text{maxval}(s, \Box, l) + 15\alpha q\). We deduce that \((s_0, h_0, l_0)\) and \((s, h, l)\) are \(\alpha\)-equivalent. Moreover, \(\text{maxval}(s, h, l) \leq \text{max}(q, q + 15\alpha q) \leq 16\alpha q\). \(\square\)

**Corollary 4.8 (Small Model Property)** Let \(A\) be an 1SL1 formula with program variables in \(x_1, \ldots, x_q\). If \(A\) is satisfiable then there exists a pointed memory state \((s, h, l)\) such that \((s, h) \models_1 A\) and \(\text{maxval}(s, h, l) \leq 16q \cdot \text{th}(q, A)\).

**Proof** Let \(\alpha = \text{th}(q, A)\). Let \((s', h', l')\) be a pointed memory state that satisfies \(A\), i.e. \((s', h') \models_\alpha A\). By Corollary 4.7, there exists a pointed memory state such that \((s', h', l') \models_\alpha (s, h, l)\) and \(\text{maxval}(s, h, l) \leq 16q \cdot \text{th}(q, A)\). By Theorem 4.3, from \(\alpha \leq \text{th}(q, A)\) we deduce \((s, h) \models_1 A\). \(\square\)

Now we can give proofs of high-level decidability results as easy consequences of the previous Compressor Lemma 4.4 and its corollaries.

**Theorem 4.9 (Decidability of Model-Checking)** The problem of checking whether an 1SL1 formula \(A\) and a pointed memory state \((s, h, l)\) verify \((s, h) \models_1 A\) is decidable.

**Proof** For \(q \geq 1\), \(V = \{x_1, \ldots, x_q\}\), a formula \(A\) with program variables in \(V\) and a pointed memory state \((s, h, l)\), we define a function \(\text{mc}(q, V, A, (s, h, l))\) which returns a value in \(\{\text{ff}, \text{tt}\}\) by structural induction on the formula \(A\):

if \(A\) is either \(e = e'\) or \(e \rightarrow e'\) or \(\text{emp}\), then we define \(\text{mc}(q, V, A, (s, h, l))\) such that \(\text{mc}(q, V, A, (s, h, l)) = \text{tt}\) iff \((s, h) \models_1 A\) (the details are left to the reader, same as upcoming function \(\text{amc}\) in Figure 4.1 page 40);

if the principal connective of \(A\) is Boolean, for instance if \(A = A_1 \land A_2\) then

\[
\text{mc}(q, V, A_1, (s, h, l)) = \text{tt}\text{ and } \text{mc}(q, V, A_2, (s, h, l)) = \text{tt}
\]

hold; otherwise \(\text{mc}(q, V, A, (s, h, l)) = \text{ff}\);
if \( \mathcal{A} \) is \( \exists u \mathcal{A}_1 \) then \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{tt} \) if \( \text{mc}(q, V, \mathcal{A}_1, (s, h, l')) = \text{tt} \) holds for some \( l' \leq \text{maxval}(s, h) + 1 \); otherwise \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{ff} \);

if \( \mathcal{A} \) is \( \mathcal{A}_1 \ast \mathcal{A}_2 \) then \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{tt} \) if both

\[
\text{mc}(q, V, \mathcal{A}_1, (s, h, 1, l)) = \text{tt} \quad \text{and} \quad \text{mc}(q, V, \mathcal{A}_2, (s, h, 2, l)) = \text{tt}
\]

hold for some heaps \( h_1 \) and \( h_2 \) such that \( h = h_1 \uplus h_2 \) (there are only finitely many such splits); otherwise \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{ff} \);

if \( \mathcal{A} \) is \( \mathcal{A}_1 \rightarrow \mathcal{A}_2 \) then \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{ff} \) if both

\[
\text{mc}(q, V, \mathcal{A}_1, (s, h, 1, l)) = \text{tt} \quad \text{and} \quad \text{mc}(q, V, \mathcal{A}_2, (s, h \uplus h_1, l)) = \text{ff}
\]

hold for some heap \( h_1 \) such that \( h_1 \perp h \) and

\[
\text{maxval}(h_1) \leq \text{maxval}(s, h, l) + 2|\mathcal{A}_1 \rightarrow \mathcal{A}_2|q^2
\]

(there are only finitely many such \( h_1 \)); otherwise \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{tt} \).

The termination of \( \text{mc} \) is by induction on \( \mathcal{A} \): any call to \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) \) only generates finitely many recursive sub-calls on the (strict) subformulae of \( \mathcal{A} \). Let us prove the correctness of the \( \text{mc} \) function. Let us fix \( q \) and \( V \). We show by induction on \( \mathcal{A} \) that if \( \mathcal{A} \) has its program variables in \( V \) then for any pointed memory state \( (s, h, l) \), the equivalence

\[
(s, h) \models_1 \mathcal{A} \quad \text{if and only if} \quad \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{tt}
\]

holds. The proof uses Corollaries 4.5 and 4.6 in an obvious way. For instance, let us consider the case where \( \mathcal{A} = \mathcal{A}_1 \rightarrow \mathcal{A}_2 \):

1. let us assume \((s, h) \models_1 \mathcal{A} \) and let us show \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{tt} \). We show that \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{ff} \) leads to a contradiction. Indeed, in that case, there exists a heap \( h_1 \) such that \( h_1 \perp h \) and \( \text{mc}(q, V, \mathcal{A}_1, (s, h, 1, l)) = \text{tt} \) and \( \text{mc}(q, V, \mathcal{A}_2, (s, h \uplus h_1, l)) = \text{ff} \). By the induction hypothesis, we deduce \((s, h_1) \models_1 \mathcal{A}_1 \) and \((s, h \uplus h_1) \not\models_1 \mathcal{A}_2 \). As a consequence we get \((s, h) \not\models_1 \mathcal{A}_1 \rightarrow \mathcal{A}_2 \) which contradicts the hypothesis. Hence \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{ff} \) is impossible so we must have \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{tt} \);

2. let us assume \( \text{mc}(q, V, \mathcal{A}_1, (s, h, l)) = \text{tt} \) and let us show \((s, h) \models_1 \mathcal{A}_1 \rightarrow \mathcal{A}_2 \). We use Corollary 4.6. Let us consider a heap \( h_1 \) such that \( h_1 \perp h \), \( \text{maxval}(h_1) \leq \text{maxval}(s, h, l) + 2|\mathcal{A}_1 \rightarrow \mathcal{A}_2|q^2 \) and \((s, h_1) \models_1 \mathcal{A}_1 \). Let us show \((s, h \uplus h_1) \models_1 \mathcal{A}_2 \). By contradiction, if \((s, h \uplus h_1) \not\models_1 \mathcal{A}_2 \) then by induction, we have \( \text{mc}(q, V, \mathcal{A}_1, (s, h, 1, l)) = \text{tt} \) and \( \text{mc}(q, V, \mathcal{A}_2, (s, h \uplus h_1, l)) = \text{ff} \). From the definition of \( \text{mc} \), we derive \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) = \text{ff} \) which leads to a contradiction. Hence we must have \((s, h \uplus h_1) \models_1 \mathcal{A}_2 \).

To finish, deciding the model checking problem \((\mathcal{A}, (s, h, l)) \) can be done this way: compute the program variables of \( \mathcal{A} \) in \( V = \{x_1, \ldots, x_q\} \) and call \( \text{mc}(q, V, \mathcal{A}, (s, h, l)) \) (remark that if \( \mathcal{A} \) contains no program variable, we can harmlessly add an arbitrary one to \( V \)).
Later we show how to transform the decidability proof into a bounded model checking algorithm that runs in $\text{pspace}$.

**Theorem 4.10 (Decidability of Satisfiability)** The problem of checking whether an $1\text{SL}1$ formula $\mathcal{A}$ admits a pointed memory state $(s,h,l)$ such that $(s,h) \models_1 \mathcal{A}$, is decidable.

This is a direct consequence of Corollary 4.8 and Theorem 4.9. Here is our main result characterizing the expressive power of $1\text{SL}1$ in terms of Boolean combination of test formulæ.

**Theorem 4.11 (Quantifier Admissibility)** Let $q \geq 1$. Every formula $\mathcal{A}$ in $1\text{SL}1$ with program variables in $x_1, \ldots, x_q$ is logically equivalent to a Boolean combination of test formulæ in $\text{Test}^n_{\alpha}$ with $\alpha = \text{th}(q, \mathcal{A})$.

**Proof** Let $\alpha = \text{th}(q, \mathcal{A})$ and consider the (saturated) set of literals

$$S_{\alpha}(s,h,l) \overset{\text{def}}{=} \left\{ B \mid B \in \text{Test}^n_{\alpha} \text{ and } (s,h) \models_1 B \right\} \cup \left\{ \neg B \mid B \in \text{Test}^n_{\alpha} \text{ and } (s,h) \not\models_1 B \right\}$$

As $\text{Test}^n_{\alpha}$ is finite, the set $S_{\alpha}(s,h,l)$ is finite and let us consider the well-defined atom $\bigwedge S_{\alpha}(s,h,l)$. It is obvious to check the equivalence

$$(s', h') \models_1 \bigwedge S_{\alpha}(s,h,l) \text{ iff } (s,h,l) \approx_{\alpha} (s', h', l')$$

The disjunction $T_\mathcal{A} \overset{\text{def}}{=} \bigvee \left\{ \bigwedge S_{\alpha}(s,h,l) \mid (s,h) \models_1 \mathcal{A} \right\}$ is a Boolean combination (with, as is usual, the empty disjunction understood as $\bot$) of test formulæ in $\text{Test}^n_{\alpha}$ because $\bigwedge S_{\alpha}(s,h,l)$ ranges over the finite set of atoms built from $\text{Test}^n_{\alpha}$. By Theorem 4.3, we deduce that $\mathcal{A}$ is logically equivalent to $T_\mathcal{A}$ which finishes the proof. \hfill $\Box$

We note that the proof of Theorem 4.11 can lead to an algorithmic way to eliminate quantifiers thanks to the decidability of model-checking (Theorem 4.9). Indeed, by Corollary 4.7, for every pointed memory state $(s,h,l)$, there is a pointed memory state $(s', h', l')$ such that $(s,h,l)$ and $(s', h', l')$ are $\alpha$-equivalent and $\maxval(s', h', l') \leq 16q\alpha$. So, in the construction of $T_\mathcal{A}$, we can restrict ourselves to pointed memory states whose maximal value is bounded by $16q\alpha$. However, the procedure described in the proof is not really suited for an effective elimination of quantifiers. Theorem 4.11 provides a characterization of the expressive power of $1\text{SL}1$, which is now easy to differentiate from $1\text{SL}2$. No big deal here since undecidability of $1\text{SL}2$ is established in (Demri and Deters 2014) (but strictly speaking, this does not entail that $1\text{SL}2$ is strictly more expressive than $1\text{SL}1$ in case there would be a non-computable translation from $1\text{SL}2$ into $1\text{SL}1$).

**Corollary 4.12** $1\text{SL}2$ is strictly more expressive than $1\text{SL}1$.

The proof can be found in Appendix C starting at page 65.
4.3 Discussion

The Boolean combination equivalent to any formula in 1SL1 is made of test formulæ that depend on the memory threshold of the formula and on the set of program variables occurring in it, which seems fair enough.

When $A$ in 1SL1 has no free occurrence of $u$, one can show that $A$ is equivalent to a Boolean combination of formulæ in $Test_{th}(q,A)$. Similarly, when $A$ in 1SL1 has no occurrence of $u$ at all, $A$ is equivalent to a Boolean combination of formulæ of the form $x_i = x_j$, $x_i \leftarrow x_j$, $\text{alloc}(x_i)$ and $\# \text{rem} \geq k$ with the alternative definition $\land(s,h) = s(V) \cap \text{dom}(h)$; see also (Lozes 2004a,b; Brochenin et al 2009).

Theorem 4.11 witnesses that the test formulæ we introduced properly abstract memory states when 1SL1 formulæ are involved. Test formulæ from Definition 3.1 were not given to us and we had to design such formulæ to conclude Theorem 4.11. All the test formulæ can be expressed in 1SL1, see developments in Section 2.4 and Lemma 2.14.

Last but not least, we need to prove that the set of test formulæ is expressively complete to get Theorem 4.11. Lemmas 3.33, 3.34 and 3.35 are helpful to obtain the Correctness Theorem 4.3, taking care of the different first- or second-order quantifiers. It is in their proofs that the completeness of the set $Test_u$ is best illustrated. Nevertheless, to apply these lemmas in the proof of Theorem 4.3, we designed the adequate definition for the function $th(\cdot,\cdot)$ and we arranged different thresholds in their statements. Then, there is a real interplay between the definition of $th(\cdot,\cdot)$ and how Lemmas 3.33, 3.34 and 3.35 are used in the proof of Theorem 4.3.

4.4 A pSPACE Upper Bound for Model Checking and Satisfiability

In this section, we consider 1SL1 formulæ with program variables in $x_1, \ldots, x_q$. But $q \geq 1$ remains a parameter that is instantiated in our proof of the pSPACE bound for satisfiability (resp. model checking) in 1SL1; see Theorems 4.17 and 4.18.

Proposition 4.13 We consider the function $\varphi$ that maps formulæ of 1SL1 defined by $\varphi(A) \overset{\text{def}}{=} 15q^2|A|^2$. Then the following relations hold for any $A, B$:

1. $\varphi(\neg A) \geq \varphi(A)$;
2. $\varphi(\exists u.A) \geq 1 + \varphi(A)$;
3. $\varphi(A \land B) \geq \max(\varphi(A), \varphi(B))$;
4. $\varphi(A \ast B) \geq \max(\varphi(A), \varphi(B))$;
5. $\varphi(A \rightarrow B) \geq 15|A \rightarrow B|q^2 + \max(\varphi(A), \varphi(B))$.

The proof is left to the reader. Remark that the actual value of $\varphi(A) = 15q^2|A|^2$ does not matter much; other choices could be made. It is only essential that $\varphi$ verifies the relations 1–5 of Proposition 4.13 because they are essential in the proof of correctness of the algorithm $\text{bmc}$ that follows. Notice however
that $\varphi$ should also be chosen to be smoothly computable, i.e. in our case, with a low space complexity bound like $O(\log q + \log |A|)$.

In Figure 4.2, we describe the recursive function $bmc(q, m, V, A, (s, h, l))$ with the following inputs:
- $q \geq 1$ and $m \in \mathbb{N}$ are two natural numbers;
- $V = \{x_1, \ldots, x_q\}$ is a set of $q$ program variables;
- $A$ is an 1SL1 formula with program variables in $x_1, \ldots, x_q$;
- $(s, h, l)$ is a pointed memory state where $s : \{x_1, \ldots, x_q\} \rightarrow \mathbb{N}$.

Such an input is called legitimate for $bmc$. The function $bmc$ is defined by structural induction on the formula $A$ and (eventually) returns a Boolean value in $\{\text{ff}, \text{tt}\}$. There is a special treatment for atomic formulæ $B$ of 1SL1 described by the (non-recursive) function $amc(q, V, B, (s, h, l)) \in \{\text{ff}, \text{tt}\}$; see Figure 4.1.
Proposition 4.14 On a legitimate input, \( \text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) \) always terminates and returns either \( \text{ff} \) or \( \text{tt} \). If \((q, m, V)\) can be stored in memory space \( \mathcal{O}(qm \log m) \) and the relation \( \text{maxval}(s, h, l) \leq m \) holds then the call \( \text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) \) runs in space \( \mathcal{O}((q + |\mathcal{A}|)m \log m) \).

Proof \( \text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) \) terminates because it is defined by structural induction on \( \mathcal{A} \). The depth of recursion is thus bounded by the depth of \( \mathcal{A} \), which is itself bounded by \(|\mathcal{A}|\). The parameters \( q, m, V \), and \( s \) remain unchanged during a run of \( \text{bmc} \) and can be stored in space \( \mathcal{O}(qm \log m) \). Each recursive call involves storing new data in the stack:

- when \( \text{bmc}(q, m, V, \exists u \mathcal{A}, (s, h, l)) \) makes the recursive call, the new data is \( l_0 \) (size bounded by \( \log m \));
- when \( \text{bmc}(q, m, V, \mathcal{A} \ast \mathcal{B}, (s, h, l)) \) makes recursive calls, the new data is either \( h_1 \) or \( h_2 = h - h_1 \) (both of size bounded by \( m \log m \));
- when \( \text{bmc}(q, m, V, \mathcal{A} \rightarrow \mathcal{B}, (s, h, l)) \) makes recursive sub-calls, the new data is either \( h_1 \) or \( h \gg h_1 \) (both of size bounded by \( m \log m \)).

We now prove following correctness lemma for \( \text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) \): on a legitimate input, it returns the correct value of the predicate \( (s, h) \models_1 \mathcal{A} \) provided \( m \) is chosen large enough.

Lemma 4.15 Let \( q \geq 1 \) and \( m \in \mathbb{N} \). Let \( \mathcal{A} \) be an 1SL1 formula with program variables in \( V = \{x_1, \ldots, x_q\} \) and \( (s, h, l) \) be a pointed memory state. If we assume \( \text{maxval}(s, h, l) + \varphi(\mathcal{A}) \leq m \) then

\[
\text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \text{tt} \quad \text{iff} \quad (s, h) \models_1 \mathcal{A}
\]

The proof can be found in Appendix C starting at page 65.

Theorem 4.16 Let \( q \geq 1 \). Given a 1SL1 formula \( \mathcal{A} \) with program variables in \( x_1, \ldots, x_q \), a pointed memory state \((s, h, l)\) and \( m \in \mathbb{N} \) such that \( \max(q, \text{maxval}(s, h, l), |\mathcal{A}|) \leq m \), it is possible to check the predicate \( (s, h) \models_1 \mathcal{A} \) in space \( \mathcal{O}(m^5 \log m) \).

Proof Let \( V = \{x_1, \ldots, x_q\} \) and \( m' = 16m^4 \). We have \( \text{maxval}(s, h, l) \leq m \leq m' \) and

\[
\text{maxval}(s, h, l) + \varphi(\mathcal{A}) = \text{maxval}(s, h, l) + 15q^2|\mathcal{A}|^2 \leq m + 15m^4 \leq m'
\]

Up to compression of the names \( x_1, \ldots, x_q \), we can store the data \((q, 16m^4, V)\) in space \( \mathcal{O}(qm' \log m') \). Hence \( \text{bmc}(q, m', V, \mathcal{A}, (s, h, l)) \) terminates after a run in space \( \mathcal{O}((q + |\mathcal{A}|)m' \log m') \) (see Proposition 4.14) and returns the correct value of the predicate \( (s, h) \models_1 \mathcal{A} \) (see Lemma 4.15). From \( q \leq m \) and \(|\mathcal{A}| \leq m\), we deduce the \( \mathcal{O}(m'^3 \log m) \) space upper bound.

\[\square\]

Theorem 4.17 The satisfiability of an 1SL1 formula \( \mathcal{A} \) can be solved in space \( \mathcal{O}(|\mathcal{A}|^{15} \log |\mathcal{A}|) \).
Proof The first step is to collect the program variables of $\mathcal{A}$. Since each atomic formula contains at most two different program variables, $\mathcal{A}$ contains at most $|\mathcal{A}|+1$ different program variables. Let $q = |\mathcal{A}|+1$. Hence there exists a set $\mathcal{V} = \{x_1, \ldots, x_q\}$ such that $\mathcal{A}$ has program variables in $x_1, \ldots, x_q$. Computing $q$ and $\mathcal{V}$ can be done in deterministic time $O(|\mathcal{A}| \log |\mathcal{A}|)$ with a sorting algorithm. Hence it can also be done in space $O(|\mathcal{A}| \log |\mathcal{A}|)$.

We now use the particular values $q$ and $\mathcal{V}$ obtained from $\mathcal{A}$. Corollary 4.8 states that if $\mathcal{A}$ is satisfiable, then there is a pointed memory state $(s, h, l)$ such that $(s, h) \models_1 \mathcal{A}$ and $\maxval(s, h, l) \leq m$ with $m = 16q \text{th}(q, \mathcal{A})$. From $q = |\mathcal{A}| + 1$ and $\text{th}(q, \mathcal{A}) \leq q|\mathcal{A}|$, we deduce $m \leq 16(|\mathcal{A}| + 1)^3$.

To check for the satisfiability of $\mathcal{A}$, it is thus sufficient to test the predicate $(s, h) \models_1 \mathcal{A}$ for all pointed memory states $(s, h, l)$ such that $\maxval(s, h, l) \leq m$. For such a pointed memory state we have $\maxval(q, \maxval(s, h, l), |\mathcal{A}|) \leq \maxval(q, m, |\mathcal{A}|) \leq 16(|\mathcal{A}| + 1)^3$. Hence by Theorem 4.16, checking for satisfiability of $\mathcal{A}$ can be done in space $O(|\mathcal{A}|^{13} \log |\mathcal{A}|)$. \(\square\)

Before we prove the PSPACE upper bound for the model-checking problem, we have to describe the composition of the input of this problem (and how its size is measured) because it does not only contain a formula but also a model. Hence the input of the model-checking problem is of the form $(s, h, l, \mathcal{A})$ where $(s, h, l)$ is a pointed memory state and $\mathcal{A}$ is an 1SL1 formula; and the problem is to determine whether $(s, h) \models_1 \mathcal{A}$ holds. Let $\mathcal{V} = \{x_1, \ldots, x_q\}$ be the $q$ different program variables that occur in $\mathcal{A}$. The store $s$ must be defined on at least those $q$ program variables that occur in $\mathcal{A}$. Let $n$ be the cardinality of $\text{dom}(h)$. The heap $h$ which is defined on exactly $n$ locations. Let $m$ be the maximal value of the numerical inputs, i.e. $s(\mathcal{V}) \cup \text{dom}(h) \cup \text{ran}(h) \cup \{\}$.

The value $(q + n + 1) \log m + |\mathcal{A}|$ measures the size of the input of the model-checking problem. We rather choose the lower bound $k = q + n + 1 + |\mathcal{A}|$, i.e. we assume that each numerical input is at least one bit long.

Theorem 4.18 Model-checking for 1SL1 can be solved in polynomial space.

Proof From the input $((s, h, l), \mathcal{A})$, we first compute a pointed memory state $(s', h', l')$ of small size which is equivalent to $(s, h, l)$ in polynomial space: we construct $(s', h', l')$ such that $\maxval(s', h', l') \leq 2k$ and $((s, h) \models_1 \mathcal{A}$ iff $(s', h') \models_{_P} \mathcal{A}$). Let us define the set $D = s(\mathcal{V}) \cup \text{dom}(h) \cup \text{ran}(h) \cup \{\}$. We know that $d = |\text{card}(D)| \leq q+2n+1 \leq 2k$. In deterministic polynomial-time (and thus also polynomial space), we compute a bijection $\varphi : D \rightarrow [0, d - 1]$, for instance using a sorting algorithm. Then we compute $s' = \varphi \circ s$, $h' = \varphi \circ h \circ \varphi^{-1}$ and $l' = \varphi(l)$ in polynomial time. We obviously have $\maxval(s', h', l') \leq d - 1 \leq 2k$ and by Proposition 2.3, we have the equivalence $(s, h) \models_1 \mathcal{A}$ iff $(s', h') \models_{_P} \mathcal{A}$.

We have $\maxval(q, \maxval(s', h', l'), |\mathcal{A}|) \leq 2k$, hence by Theorem 4.16, we can check the predicate $(s', h') \models_{_P} \mathcal{A}$ in space $O(k^5 \log k)$ which is thus polynomial in the size of the input. \(\square\)

Theorem 4.19 The model-checking and satisfiability problems for 1SL1 are PSPACE-complete.
Proof PSpace-hardness for both problems is a consequence of (Calcagno et al 2001) since the problems for propositional separation logic SL0 are already PSpace-hard. Note that the PSpace-hardness proof can be adapted with a single record field and that nil can be simulated by a dedicated program variable, see (Calcagno et al 2001). The PSpace upper bound for model-checking (resp. satisfiability) is stated in Theorem 4.18 (resp. Theorem 4.17).

**Corollary 4.20** Let \( q \geq 1 \). Let \( \mathcal{A} \) be an ISL1 formula with program variables in \( x_1, \ldots, x_q \) and let \( \alpha = \text{th}(q, \mathcal{A}) \). Computing a Boolean combination of test formul\ae\ in \( \text{Test}_\alpha \) logically equivalent to \( \mathcal{A} \) can be done in polynomial space (even though the outcome formula can be of exponential size).

Proof We analyze the proof of Theorem 4.11. Let \( \mathcal{A} \) be an ISL1 formula with \( \alpha = \text{th}(q, \mathcal{A}) \). By the proof of Theorem 4.11 and Corollary 4.7, \( \mathcal{A} \) is logically equivalent to the formula below

\[
\bigvee \left\{ \land S_\alpha(s, h, l) \mid (s, h) \models_l \mathcal{A} \text{ and } \text{maxval}(s, h, l) \leq 16q\alpha \right\}
\]

The non-isomorphic copies of pointed memory states \( (s, h, l) \) such that

\[
\text{maxval}(s, h, l) \leq 16q\alpha
\]

can be enumerated in polynomial space. Moreover, the model-checking problem for ISL1 can be solved in polynomial space, whence the above formula can be built in polynomial space (even though its size may be exponential in the size of \( \mathcal{A} \)). Note also that \( S_\alpha(s, h, l) \) can be computed in polynomial space too. \( \square \)

### 5 Complexity of Satisfiability of Test Formulae

The goal of this section is to complete the proof of Theorem 3.5. We decide the conjunctions of literals in deterministic polynomial-time using a saturation algorithm based in a set of deduction rules for basic test formul\ae\ . We then describe the NP procedure for Boolean combinations of test formul\ae\ .

Remember that \( \text{Basic}^\alpha \) (resp. \( \text{Test}^\alpha \)) denotes the set of basic formul\ae\ (resp. test formul\ae\ ) built from the program variables \( x_1, \ldots, x_q \).

**Proposition 5.1 (Soundness of the saturation rules)** Let \( \mathcal{P} \) be a subset of \( \text{Basic}^\alpha \) and let \( (s, h, l) \) be a pointed memory state such that \( (s, h) \models_l \mathcal{B} \) holds for any \( \mathcal{B} \in \mathcal{P} \). If \( \mathcal{C} \in \text{Basic}^\alpha \) is derivable from the formul\ae\ of \( \mathcal{P} \) using the rules of Figure 5.1 then \( (s, h) \models_l \mathcal{C} \) holds.

The proof is by induction on the derivation height and it is left to the reader.

If a set of basic formul\ae\ is satisfied in a model, then all the basic logical consequences of those formul\ae\ are also satisfied in that model. Hence, in general it is not possible to satisfy exactly a given set of basic formul\ae\ . But if a set \( \mathcal{P} \) of basic formul\ae\ is closed under the rules of Figure 5.1, then it is possible to satisfy exactly the formul\ae\ of \( \mathcal{P} \) in the canonical pre-model defined below.
**Definition 5.2** (Canonical pre-model) Let \( q \geq 1 \). The canonical pre-model of a finite set \( P \) of formulae of Basic\(^a\) is built the following way: we define two partial functions \( s : V \to [1, q] \) and \( h : [1, q] \to [q + 1, 2q] \), a finite graph \( H \subseteq [0, 2q] \times [0, 2q + 1] \) and a finite subset \( L \subseteq [0, 2q] \) by:

- \( s(x_i) \overset{\text{def}}{=} \min\{j \mid x_i = x_j \in P\} \) and \( h_i \overset{\text{def}}{=} \min\{q + j \mid \text{conv}(x_i, x_j) \in P\} \);
- \( F_m \overset{\text{def}}{=} \{x_i \mapsto x_i, \ldots, x_i \mapsto x_q\} \) and \( B_w \overset{\text{def}}{=} \{\text{btwn}(x_i, x_1), \ldots, \text{btwn}(x_i, x_q)\} \);
- \( F_u \overset{\text{def}}{=} \{u \mapsto x_1, \ldots, u \mapsto x_q\} \) and \( T_o \overset{\text{def}}{=} \{x_i \mapsto u, \ldots, x_q \mapsto u\}; \)
- \( E_q \overset{\text{def}}{=} \{x_1 = u, \ldots, x_q = u\} \);
- for \( B \subseteq \text{Basic}\(^a\) the notation \( B \downarrow P \) is a shortcut for \( B \cap P = \emptyset \);
- \( H \) is defined to be the least set such that:
  - \( H_1. (s(x_i), s(x_j)) \in H \) if \( x_i \mapsto x_j \in P \);
  - \( H_2. (s(x_i), h_j) \in H \) if \( \text{conv}(x_i, x_j) \in P \) and \( F_m \downarrow P \);
  - \( H_3. (h_i, s(x_j)) \in H \) if \( \text{btwn}(x_i, x_j) \in P \) and \( F_m \downarrow P \);
  - \( H_4. (h_i, h_j) \in H \) if \( \{\text{conv}(x_i, x_j), \text{toloop}(x_j)\} \subseteq P \) and \( F_m \downarrow P \);
  - \( H_5. (h_i, 0) \in H \) if \( \text{toloop}(x_i) \in P \) and \( (F_m \cup B_w) \cup \{\text{toloop}(x_i)\} \subseteq P \);
  - \( H_6. (0, s(x_i)) \in H \) if \( u \mapsto x_i \in P \) and \( (E_q \cup T_o) \downarrow P \);
  - \( H_7. (0, 0) \in H \) if \( u \mapsto u \in P \) and \( (E_q \cup T_o) \cup F_m \downarrow P \);
  - \( H_8. (0, 2q + 1) \in H \) if \( (E_q \cup T_o) \cup F_m \cup \{u \mapsto u\} \downarrow P \) and \( \text{alloc}(u) \in P \);
- \( L \) is defined to be the least set such that:
  - \( L_1. s(x_i) \in L \) if \( x_i = u \in P \);
L2. \( b_i \in L \) if \( x_i \leftrightarrow u \in P \) and \( \text{Eq}_u \perp P \);
L3. \( 0 \in L \) if \((\text{Eq}_u \cup \text{To}_u) \perp P \).

Remark that when we write e.g. \((s(x_i), h_j) \in H\), we assume that \( s(x_i) \) and \( h_j \) are both defined otherwise the pair is not added to \( H \). The same remark holds for the definition of \( L \).

Remark that \( \text{Fm} \) abbreviates “from,” \( \text{Bw} \) abbreviates “between,” \( \text{To} \) abbreviates “equal.” We prove that the canonical pre-model, which might not even be a memory state in general, becomes a model of exactly those formulæ of \( P \) when \( P \) is saturated under logical consequence.

**Proposition 5.3 (Completeness of the saturation rules)** If the (finite) subset \( P \subseteq \text{Basic}^a \) is closed under the rules of Figure 5.1 and \((s, H, L) \) is the canonical pre-model of \( P \) then:
- \( s \) is a total function \( s : \mathcal{V} \to [1, q] \), hence \( s \) is a store;
- \( H \) is a finite and functional graph, hence \( H \) is the graph of some heap \( h \);
- \( L \) is a singleton subset of \( \mathbb{N} \), i.e. \( L = \{l\} \) for a location \( l \);
- the inclusion \( \text{dom}(h) \subseteq \bigcap(s, h) \cup \{l\} \) holds;
- for any formula \( B \in \text{Basic}^a \) we have \((s, h) \models \_1 B \) iff \( B \in P \).

The proof can be found in Appendix D starting at page 66.

Hence if \( P \) is closed under the rules of Figure 5.1 then the canonical pre-model of \( P \) is a model of exactly those formulæ in \( P \). As a consequence, any conjunction of basic test formulæ is satisfiable. Let us see what happens when we add negations of basic test formulæ and cardinality constraints like \( \# \text{pred}(x_i) \geq k \) or \( \# \text{loop}_{x_i} \geq k \)...

Let us write \( \text{cl}(P) \) to denote the closure of a (finite) set \( P \) of basic formulæ of \( \text{Basic}^a \) under the rules of Figure 5.1, i.e. all the formulæ that can be deduced from those of \( P \). We can compute \( \text{cl}(P) \) by an obvious saturation algorithm. Since \( \text{Basic}^a \) is closed under the rules of Figure 5.1, the cardinal of \( \text{cl}(P) \) is at most the cardinal of \( \text{Basic}^a \), i.e. \( 4q^2 + 6q + 3 \). Hence, the saturation algorithm runs in polynomial time in \( q \).

The expression \( \sim B^- \) is defined as usual by \( \sim B^- \overset{\text{df}}{=} \{ \sim B \mid B \in B^- \} \).

Given \( B^+, B^- \subseteq \text{Basic}^a \), checking for the satisfiability of the conjunction of the formulæ of \( B^+ \cup \sim B^- \) is easy. It is sufficient to compute the closure \( \text{cl}(B^+) \):
- if \( B^- \cap \text{cl}(B^+) = \emptyset \) then the conjunction of \( B^+ \cup \sim B^- \) is satisfied in the canonical model of \( \text{cl}(B^+) \) by Proposition 5.3;
- if otherwise \( B^- \cap \text{cl}(B^+) \neq \emptyset \) then \( B^+ \cup \sim B^- \) is unsatisfiable. Indeed, let \( B \in B^- \cap \text{cl}(B^+) \). Then \( B \) should be both satisfied (by Proposition 5.1) and unsatisfied in any model of \( B^+ \cup \sim B^- \) which leads to a contradiction.

For the general problem of the satisfiability of conjunctions of literals in \( \text{Basic}^a \cup \text{Size}_a \), there is a subtlety related to the interpretation of the quantified variable \( u \). Indeed, using only literals from \( \text{Basic}^a \) it is possible to require that the interpretation of \( u \) belongs to \( \text{dom}(h) \setminus \bigcup(s, h) \). Indeed, any model of

\[ B \cup \{ \neg(x_1 = u), \ldots, \neg(x_q = u), \neg(x_1 \rightarrow u), \ldots, \neg(x_q \rightarrow u), \text{alloc}(u) \} \]
would meet such a requirement. Hence, one of the test formulae

\[
\# \text{pred}_\tau(x_1) \geq 1, \ldots, \# \text{pred}_\tau(x_q) \geq 1, \# \text{loop}_\tau \geq 1, \# \text{rem}_\tau \geq 1
\]

has to be satisfied in such a model. That is why we develop a more involved argument to check the satisfiability of conjunctions of literals in \(\text{Test}^n\) in polynomial time.

**Definition 5.4** A cardinality assignment is a tuple \((p_1, \ldots, p_q, l, r)\) of \(q + 2\) elements of \(\mathbb{N}\).

**Proposition 5.5** Let \(q \geq 1\). Let \(s : \mathcal{V} \to \mathbb{N}\) be a store, \(h : \mathbb{N} \to \mathbb{N}\) be a heap and \(l \in \mathbb{N}\) be a location. Let \((p_1, \ldots, p_q, l, r)\) be a cardinality assignment such that:

1. \(s(x_i) = s(x_j)\) implies \(p_i = p_j\) for all \(i, j \in [1, q]\);
2. \(\text{card}(\text{pred}_\tau(s, h, i)) \leq p_i\) for any \(i \in [1, q]\);
3. \(\text{card}(\text{loop}_\tau(s, h)) \leq 1\);
4. \(\text{card}(\text{rem}_\tau(s, h)) \leq r\).

There exists a heap \(h'\) such that:

- \((s, h, l) \sim_h (s, h', l)\);
- \(\text{card}(\text{pred}_\tau(s, h', i)) = p_i\) for any \(i \in [1, q]\);
- \(\text{card}(\text{loop}_\tau(s, h')) = 1\);
- \(\text{card}(\text{rem}_\tau(s, h')) = r\).

The proof can be found in Appendix D starting at page 76.

It is always possible to add elements outside the core of a heap as much as we wish while preserving basic equivalence.

**Definition 5.6** (1-, 2- and 3-consistency) Let \(q, a \geq 1\). Let \(\mathcal{B}^+, \mathcal{B}^-\) be two (finite) sets of formulae of Basic\(^a\) and \(S\) be a (finite) set of literals from \(\text{Size}_a\). We say that the triple \((\mathcal{B}^+, \mathcal{B}^-, S)\) is 1-consistent when for all \(i, j \in [1, q]\) and for all \(a, b \in \mathbb{N}\) the following conditions hold:

\[C1.1 \quad \mathcal{B}^- \cap \text{cl}(\mathcal{B}^+) = \emptyset;\]

\[C1.2 \quad \text{if } x_i = x_j \in \text{cl}(\mathcal{B}^+) \text{ and } \{ \# \text{pred}_\tau(x_i) \geq a, -\# \text{pred}_\tau(x_j) \geq b \} \subseteq S \text{ then } a < b;\]

\[C1.3 \quad \text{if } \{ \# \text{loop}_\tau \geq a, -\# \text{loop}_\tau \geq b \} \subseteq S \text{ then } a < b;\]

\[C1.4 \quad \text{if } \{ \# \text{rem}_\tau \geq a, -\# \text{rem}_\tau \geq b \} \subseteq S \text{ then } a < b.\]

We say that the triple \((\mathcal{B}^+, \mathcal{B}^-, S)\) is 2-consistent when it is 1-consistent and for all \(i, j \in [1, q]\) the following conditions hold:

\[C2.1 \quad \text{if } \{ x_i = x_j, u \leftarrow u \} \subseteq \text{cl}(\mathcal{B}^+) \text{ and } -\# \text{pred}_\tau(x_j) \geq 1 \in S \text{ then } (\mathcal{B}^+ \cup \{ B \}, \mathcal{B}^-, S) \text{ is 1-consistent for some } B \in \text{Eq}_a \cup \text{To}_a;\]

\[C2.2 \quad \text{if } u \leftarrow u \in \text{cl}(\mathcal{B}^+) \text{ and } -\# \text{loop}_\tau \geq 1 \in S \text{ then } (\mathcal{B}^+ \cup \{ B \}, \mathcal{B}^-, S) \text{ is 1-consistent for some } B \in \text{Eq}_a \cup \text{To}_a;\]
We say that the triple \((B^+, B^-, S)\) is \(3\)-consistent when it is \(2\)-consistent and the following conditions hold:

\[ \text{C3.1 if } \text{alloc}(u) \in \text{cl}(B^+) \text{ and } \neg \# \text{rem} \geq 1 \in S \text{ then } (B^+ \cup \{B\}, B^-, S) \text{ is } \] 2-consistent for some \(B \in Eq_u \cup To_u \cup Fm_u \cup \{u \mapsto u\}. \]

where \(Eq_u\), \(To_u\) and \(Fm_u\) are from Definition 5.2.

**Proposition 5.7** If the conjunction of the formul\ae in \(B^+ \cup \neg B^- \cup S\) is satisfiable then the triple \((B^+, B^-, S)\) is \(n\)-consistent for any \(n \in \{1, 2, 3\}\).

The proof can be found in Appendix D starting at page 77.

**Proposition 5.8** If the triple \((B^+, B^-, S)\) is \(3\)-consistent then the conjunction of the formul\ae in \(B^+ \cup \neg B^- \cup S\) is satisfiable.

The proof can be found in Appendix D starting at page 78.

**Proposition 5.9** For any \(n \in \{1, 2, 3\}\), the \(n\)-consistency of \((B^+, B^-, S)\) can be checked in polynomial time in the size of \((B^+, B^-, S)\).

The proof is left to the reader.

Let us conclude this section by explaining why the satisfiability problem for Boolean combinations of test formul\ae (see Definition 3.4) can be solved in \(NP\), establishing the upper bound that was postponed from the proof of Theorem 3.5. Let \(A\) be a Boolean combination of test formul\ae over \(x_1, \ldots, x_q\). An \(NP\) procedure for checking the satisfiability of \(A\) goes as follows:

1. non-deterministically guess a conjunction of literals contained in \(A\) that makes true \(A\) propositionally, say \(A_1 \land \cdots \land A_n\). Then check for the satisfiability of \(A_1 \land \cdots \land A_n\) starting at step 2. If it is satisfiable return “yes.” If no such conjunction is satisfiable return “false”;

2. none of the \(A_i\) can be \(\bot\); if \(A_i\) is \(\bot\) then remove it; hence all the \(A_i\) are literals from \(\text{Basic}^a \cup \text{Size}_n\);

3. sort \(A_1, \ldots, A_n\) and find \((B^+, B^-, S)\) so that \(\{A_1, \ldots, A_n\} = B^+ \cup \neg B^- \cup S\);

4. Return the \(3\)-consistency of \((B^+, B^-, S)\).

From Propositions 5.7 and 5.8, the \(3\)-consistency of \((B^+, B^-, S)\) is equivalent to satisfiability of the conjunction of \(B^+ \cup \neg B^- \cup S\), hence this is equivalent to the satisfiability of \(A_1 \land \cdots \land A_n\). Step 1 is the nondeterministic polynomial-time step, all the other steps can be performed in deterministic polynomial time; use Proposition 5.9 for step 4.

6 Conclusion

In (Brochenin et al 2012), the undecidability of first-order separation logic 1SL with a unique record field is shown. Propositional separation logic 1SL0 is also known to be \(PSPACE\)-complete (Calcagno et al 2001). In this paper,
we provided an extension with a unique quantified variable (and with both separating connectives) and we show that the satisfiability problem for 1SL1 is \textsc{pspace}-complete by presenting an original and fine-tuned abstraction of memory states. We proved that in 1SL1 separating connectives can be eliminated in a controlled way as well as first-order quantification over the single variable. In that way, we show a quantifier elimination property similar to what is known with Presburger arithmetic. Last but not least, we have established that satisfiability problem for Boolean combinations of test formulæ is \textsc{np}-complete thanks to a saturation algorithm to deal with conjunctions. This is reminiscent of decision procedures used in SMT solvers and it is a challenging question to take advantage of these features to decide 1SL1 with an SMT solver. Finally, the design of decidable fragments between 1SL1 and undecidable 1SL2 that admit decision procedures by adapting our method would be worth being further investigated. Indeed, even though the extension with strictly more than one record field might preserve decidability (which remains to be formally proved), it is open whether the addition of the reachability predicate remains decidable.

References


Lozes E (2004b) Separation Logic preserves the expressive power of classical logic. In: 2nd Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE’04)
A Proofs of Section 2

Proposition 2.9 Let $s, h, h_1, h_2$ be such that $h = h_1 \equiv h_2$ and let $i \in [1, q]$. The following identities hold:

1. $\text{pred}(s, h_1, i) = (\text{pred}(s, h, i) \cap \text{dom}(h_1)) \cup (\text{pred}(s, h, i) \cap \Delta(s, h_1, h_2))$;
2. $\text{loop}(s, h_1) = (\text{loop}(s, h) \cap \text{dom}(h_1)) \cup (\text{loop}(s, h) \cap \Delta(s, h_1, h_2))$;
3. $\text{rem}(s, h_1) = (\text{rem}(s, h) \cap \text{dom}(h_1)) \cup (\text{rem}(s, h) \cap \Delta(s, h_1, h_2))$.

Proof First, observe that we have the following identities:

$$\text{pred}(s, h_1, i) = \text{pred}(s, h, i) \cap \text{dom}(h_1)$$
$$\text{loop}(s, h_1) = \text{loop}(s, h) \cap \text{dom}(h_1)$$
$$\text{rem}(s, h_1) = \text{rem}(s, h) \cap \text{dom}(h_1)$$

By definition, we have

$$\text{pred}(s, h_1, i) = \text{pred}(s, h, i) \cup (\text{dom}(h_1) \setminus \text{dom}(h))$$

Hence,

$$\text{pred}(s, h_1, i) = (\text{pred}(s, h, i) \cap \text{dom}(h_1)) \cup (\text{pred}(s, h, i) \cap \Delta(s, h_1, h_2))$$

Consequently,

$$\text{pred}(s, h_1, i) = (\text{pred}(s, h, i) \cap \text{dom}(h_1)) \cup (\text{pred}(s, h, i) \cap \Delta(s, h_1, h_2))$$

since $\Delta(s, h_1, h_2) \subseteq \text{dom}(h_1)$. The other identities are established in a similar fashion. □

Proposition 2.10 Let $(s, h)$ be a memory state, $l_1 \in \mathbb{N}\setminus \text{dom}(h)$ and $l_2 \in \mathbb{N}$. Let us write $h_{1 \rightarrow 2}$ for $h \equiv [l_1 \rightarrow l_2]$ and let $i$ be in $[1, q]$. The following identities hold:

$$\text{dom}(h_{1 \rightarrow 2}) = \text{dom}(h) \cup \{l_1\}$$
$$\text{pred}(s, h_{1 \rightarrow 2}, i) = \text{pred}(s, h, i) \cup \{l_1\} \text{ if } l_2 = s(x_i)$$
$$\text{loop}(s, h_{1 \rightarrow 2}) = \text{loop}(s, h)$$
$$\text{rem}(s, h_{1 \rightarrow 2}) = \text{rem}(s, h) \cup \{l_1\} \text{ if } l_2 \not\in \text{dom}(V) \cup \{l_1\}$$
$$\n(s, h_{1 \rightarrow 2}) = \n(s, h) \cup \{l_1\} \text{ if } l_2 \in \text{dom}(V)$$

Proof The proof of the first four identities is left to the reader. For the identity that describes $\n(s, h_{1 \rightarrow 2})$, we notice that

$$\text{ref}(s, h_{1 \rightarrow 2}) = \text{ref}(s, h) \cup \{l_1\}$$

holds. For $\text{acc}(s, h_{1 \rightarrow 2})$, it is a bit more complicated. We have

$$\text{acc}(s, h_{1 \rightarrow 2}) = h(s(V)) \cup \{l_2\} \text{ if } l_1 \in s(V) \text{ and } l_2 \not\in \n(s, h) \cup \{l_1\} \text{ or } l_2 \not\in \text{dom}(h) \cup \{l_1\}$$

Hence we deduce the properties:

(P1) $\text{acc}(s, h) \subseteq \text{acc}(s, h_{1 \rightarrow 2}) \subseteq \text{acc}(s, h) \cup \{l_1, l_2\}$;
(P2) $l_1 \in \text{acc}(s, h_{1 \rightarrow 2}) \iff l_1 \in h(s(V))$ or $l_1 = l_2 \in s(V)$;
Proposition 2.11 Let \((s, h)\) be a memory state, \(l_1 \in \mathbb{N}\)\(\text{dom}(h)\) and \(l_2 \in \mathbb{N}\). Let us write \(h_{l_2} = h]\([l_1 \rightarrow l_2]\) and let \(i\) be in \([1, q]\). The following identities hold:

\[
\begin{align*}
\text{pred}_{(s, h_{l_2})}(i) &= \text{pred}_{(s, h_{l_2})}(i)\quad \text{if} \quad l_1 \neq \text{pred}_{(s, h_{l_2})}(i) \\
&= \text{pred}_{(s, h_{l_2})}(i)\quad \text{otherwise}
\end{align*}
\]

\[
\begin{align*}
\text{loop}_{(s, h_{l_2})} &= \text{loop}_{(s, h_{l_2})}\quad \text{if} \quad l_1 \neq \text{loop}_{(s, h_{l_2})} \\
&= \text{loop}_{(s, h_{l_2})}\quad \text{otherwise}
\end{align*}
\]

\[
\begin{align*}
\text{rem}_{(s, h_{l_2})} &= \text{rem}_{(s, h_{l_2})}\quad \text{if} \quad l_1 \neq \text{rem}_{(s, h_{l_2})} \\
&= \text{rem}_{(s, h_{l_2})}\quad \text{otherwise}
\end{align*}
\]

where \(X - \{l_2\}\) means that the location \(l_2\) already belongs to the set \(X\) and is (strictly) removed from it.

Proof Let us first establish the two following properties:
(P1) \( l_1 \in \nabla(s, h_{1\to 2}) \) iff \( l_1 \in \nabla(s, h) \);

(P2) if \( l_2 \in \text{dom}(h) \setminus \nabla(s, h) \) then \( (l_2 \in \nabla(s, h_{1\to 2}) \) iff \( l_1 \in s(V) \)).

Property (P1) is a direct consequence of the last equation of Proposition 2.10 and the fact that \( l_1 \notin \nabla(s, h) \) (remember \( l_1 \notin \text{dom}(h) \)). Let us prove Property (P2):

- for the only if part, we assume \( l_2 \notin \nabla(s, h_{1\to 2}) \) and prove \( l_1 \notin s(V) \) by contradiction.
  Indeed, if \( l_1 \notin s(V) \), then we have the inclusion \( l_2 \in \nabla(s, h_{1\to 2}) \subseteq \nabla(s, h) \setminus \{l_1\} \) according to Proposition 2.10. Hence either \( l_2 \notin \nabla(s, h) \) which contradicts \( l_2 \in \text{dom}(h) \setminus \nabla(s, h) \) or \( l_2 = l_1 \) which implies \( l_2 \notin \text{dom}(h) \) and contradicts \( l_2 \in \text{dom}(h) \setminus \nabla(s, h) \);

- for the if part, if \( l_1 \in s(V) \) then \( l_2 = h_{1\to 2}(l_1) \in h_{1\to 2}(s(V)) \). Since \( l_2 \in \text{dom}(h) \subseteq \text{dom}(h_{1\to 2}) \) we deduce \( l_2 \in \nabla(s, h_{1\to 2}) \).

By Proposition 2.10, there are only three possible values for \( \nabla(s, h_{1\to 2}) \):

\[
\nabla(s, h_{1\to 2}) = \{ \nabla(s, h), \nabla(s, h) \cup \{l_1\}, \nabla(s, h) \cup \{l_1, l_2\} \}
\]

(A.1)

Let us now consider the case of \( \text{pred}(s, h_{1\to 2}, i) \). According to Proposition 2.10 and \( l_1 \notin \text{dom}(h) \), we know that

\[
\text{pred}(s, h_{1\to 2}, i) \in \{ \text{pred}(s, h, i), \text{pred}(s, h, i) \cup \{l_1\} \}
\]

Hence there are only three possible values for \( \text{pred}(s, h_{1\to 2}, i) \) which are \( \text{pred}(s, h, i) \cup \{l_1\} \), \( \text{pred}(s, h, i) \), and \( \text{pred}(s, h, i) \setminus \{l_2\} \) and we study those three cases:

- \( l_1 \in \text{pred}(s, h_{1\to 2}, i) \setminus \text{pred}(s, h, i) \) iff \( l_1 \in \text{pred}(s, h_{1\to 2}, i) \) iff \( l_1 \in \text{pred}(s, h_{1\to 2}, i) \) and \( l_1 \notin \nabla(s, h_{1\to 2}) \) iff \( l_2 = s(x_1) \) and \( l_1 \notin \nabla(s, h) \);

- if \( l_2 \in \text{pred}(s, h, i) \setminus \text{pred}(s, h, i) \) then \( l_1 \in \text{pred}(s, h, i) \) and \( l_2 \notin \text{pred}(s, h, i) \) or \( l_2 \in \nabla(s, h_{1\to 2}) \). But \( \text{pred}(s, h, i) \subseteq \text{pred}(s, h, i) \subseteq \text{pred}(s, h_{1\to 2}, i) \). Hence we know that \( l_2 \in \text{pred}(s, h_{1\to 2}, i) \) and thus we must have \( l_2 \in \nabla(s, h_{1\to 2}) \). Since \( l_2 \in \text{pred}(s, h, i) \subseteq \text{dom}(h) \setminus \nabla(s, h) \) we deduce \( l_1 \in s(V) \) by Property (P2);

- if \( l_1 \in s(V) \) and \( l_2 \in \text{pred}(s, h, i) \) then \( l_2 \in h_{1\to 2}(s(V)) \) \( \cap \) \( \text{dom}(h) \subseteq \nabla(s, h_{1\to 2}) \). Hence \( l_2 \notin \text{pred}(s, h_{1\to 2}, i) \).

Let us now consider the case of \( \text{loop}(s, h_{1\to 2}) \). According to Proposition 2.10, we know that

\[
\text{loop}(s, h_{1\to 2}) \in \{ \text{loop}(s, h), \text{loop}(s, h) \cup \{l_1\} \}
\]

and by inclusion (A.1), we deduce that there are only three possible values for \( \text{loop}(s, h_{1\to 2}) \) which are \( \text{loop}(s, h) \cup \{l_1\} \), \( \text{loop}(s, h) \), and \( \text{loop}(s, h) \setminus \{l_2\} \).

- \( l_1 \in \text{loop}(s, h_{1\to 2}) \setminus \text{loop}(s, h) \) iff \( l_1 \in \text{loop}(s, h_{1\to 2}) \) iff \( l_1 \in \text{loop}(s, h_{1\to 2}) \) and \( l_1 \notin \nabla(s, h_{1\to 2}) \) iff \( l_2 = s(x_1) \) and \( l_1 \notin \nabla(s, h) \);

- if \( l_2 \in \text{loop}(s, h) \setminus \text{loop}(s, h) \) then \( l_2 \in \text{loop}(s, h) \) and either \( l_2 \notin \text{loop}(s, h_{1\to 2}) \) or \( l_2 \in \nabla(s, h_{1\to 2}) \). But \( \text{loop}(s, h) \subseteq \text{loop}(s, h_{1\to 2}) \) and thus we must have \( l_2 \in \nabla(s, h_{1\to 2}) \). Since \( l_2 \in \text{loop}(s, h) \subseteq \text{dom}(h_{1\to 2}) \setminus \nabla(s, h) \) we deduce \( l_1 \in s(V) \) by Property (P2);

- if \( l_1 \in s(V) \) and \( l_2 \in \text{loop}(s, h) \) then \( l_2 \in h_{1\to 2}(s(V)) \) \( \cap \) \( \text{dom}(h) \subseteq \nabla(s, h_{1\to 2}) \). Hence \( l_2 \notin \text{loop}(s, h_{1\to 2}) \).

Let us now consider the case of \( \text{rem}(s, h_{1\to 2}, i) \). According to Proposition 2.10, we know that

\[
\text{rem}(s, h_{1\to 2}) \in \{ \text{rem}(s, h), \text{rem}(s, h) \cup \{l_1\} \}
\]

and by inclusion (A.1), we deduce that there are only three possible values for \( \text{rem}(s, h_{1\to 2}) \) which are \( \text{rem}(s, h) \cup \{l_1\} \), \( \text{rem}(s, h) \), and \( \text{rem}(s, h) \setminus \{l_2\} \).

- \( l_1 \in \text{rem}(s, h_{1\to 2}) \setminus \text{rem}(s, h) \) iff \( l_1 \in \text{rem}(s, h_{1\to 2}) \) iff \( l_1 \in \text{rem}(s, h_{1\to 2}) \) and \( l_1 \notin \nabla(s, h_{1\to 2}) \) iff \( l_2 = s(x_1) \) and \( l_1 \notin \nabla(s, h) \);

- if \( l_2 \in \text{rem}(s, h) \setminus \text{rem}(s, h) \) then \( l_2 \in \text{rem}(s, h) \) and either \( l_2 \notin \text{rem}(s, h_{1\to 2}) \) or \( l_2 \in \nabla(s, h_{1\to 2}) \). Because of the inclusions \( \text{rem}(s, h) \subseteq \text{rem}(s, h) \subseteq \text{rem}(s, h_{1\to 2}) \), we have \( l_2 \notin \text{rem}(s, h_{1\to 2}) \) and thus we deduce \( l_2 \notin \nabla(s, h_{1\to 2}) \). Since \( l_2 \in \text{rem}(s, h) \subseteq \text{dom}(h_{1\to 2}) \setminus \nabla(s, h) \) we deduce \( l_1 \in s(V) \) by Property (P2);
- if $l_1 \in s(V)$ and $l_2 \in \text{rem}(s,h)$ then $l_2 \not\in h_{1\to 2}(s(V)) \cap \text{dom}(h) \subseteq \bigcup(s,h_{1\to 2})$. Hence $l_2 \not\in \text{rem}(s,h_{1\to 2})$.

\[ \square \]

**Lemma 2.14** For any $k \geq 1$ and for any $i \in [1,q]$, there exist ISL1 formulae denoted $\# \text{pred}_{\bigcirc}(x_i) \geq k$, $\# \text{loop}_{\bigcirc} \geq k$ and $\# \text{rem}_{\bigcirc} \geq k$ respectively such that, for any memory state $(s,h)$ and for any location $l \in \text{N}$ the following equivalences hold:

1. $(s,h) \models \# \text{pred}_{\bigcirc}(x_i) \geq k$ iff $\text{card}(\text{pred}_{\bigcirc}(s,h,i)) \geq k$;
2. $(s,h) \models \# \text{loop}_{\bigcirc} \geq k$ iff $\text{card}(\text{loop}_{\bigcirc}(s,h)) \geq k$;
3. $(s,h) \models \# \text{rem}_{\bigcirc} \geq k$ iff $\text{card}(\text{rem}_{\bigcirc}(s,h)) \geq k$.

**Proof** Let us first establish the equivalence $(s,h) \models \# \text{pred}_{\bigcirc}(x_i) \geq k$ iff $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \geq k$.

For the if part, let us assume $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \geq k$. Then, then let us define

\[ R = \{ i \in [1,q] \mid \text{h}(s(x_i)) = s(x_i) \text{ and } \forall r \in [1,q], \text{h}(s(x_r)) = s(x_r) \Rightarrow i \leq r \} \]

\[ A = \{ i \in [1,q] \mid \text{h}(s(x_i)) \not= s(V) \text{ and } \text{h}^2(s(x_i)) = s(x_i) \text{ and } \forall r \in [1,q], \text{h}(s(x_r)) = s(x_r) \Rightarrow i \leq r \} \]

We also recall the notations $sR = \{ s(x_i) \mid i \in R \}$ and $sA = \{ s(x_i) \mid i \in A \}$ from Proposition 2.13. We check the identities $\text{pred}_{\bigcirc}(s,h,j) = sR \cup s(hA)$, $\text{card}(sR) = \text{card}(R)$ and $\text{card}(h(sA)) = \text{card}(A)$ hold. We deduce $\text{card}(R) + \text{card}(A) \geq k$. By Proposition 2.13, we have $(s,h) \models \text{ref}_{\bigcirc}$ and $(s,h) \models \text{acc}_{\bigcirc}$. For any $r \in R$ we have $s(h(x_r)) = s(x_r)$ hence $(s,h) \models \bigwedge_{r \in R} \text{ref}_{\bigcirc}(x_r)$. For any $a \in A$ we have $h^2(s(x_a)) = s(x_a)$ hence $(s,h) \models \bigwedge_{a \in A} \text{acc}_{\bigcirc}(x_a)$. We deduce $sR \subseteq \text{ref}_{\bigcirc}(s,h,j) \cap \text{acc}_{\bigcirc}(s,h,j)$ and $\text{card}(sR) = \text{card}(R)$ and $\text{card}(h(sA)) = \text{card}(A)$. Hence $sR \cup s(hA) \subseteq \text{pred}_{\bigcirc}(s,h,j)$ and $\text{card}(sR \cup s(hA)) = \text{card}(R) + \text{card}(A) \geq k$.

As a consequence, $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \geq k$ holds.

Let us now establish the equivalence $(s,h) \models \# \text{pred}_{\bigcirc}(x_i) \geq k$ iff $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \geq k$.

For the if part, let us assume $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \geq k$. Let us define $p = \text{card}(\text{pred}_{\bigcirc}(s,h,j))$. From $\text{pred}_{\bigcirc}(s,h,j) \subseteq \bigcup(s,h)$ we deduce $p \leq 2q$. We have $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) = p < p + 1$ and as a consequence, we deduce $(s,h) \not\models \# \text{pred}_{\bigcirc}(x_i) \geq p + 1$. From $\text{pred}_{\bigcirc}(s,h,j) = \text{pred}_{\bigcirc}(s,h,j) \cup \text{pred}_{\bigcirc}(s,h,j)$ we get $\text{pred}_{\bigcirc}(s,h,j) \geq k + p$ and thus the relation $(s,h) \models \# \text{pred}_{\bigcirc}(x_i) \geq k$.

For the only if part, let us assume $(s,h) \models \# \text{pred}_{\bigcirc}(x_i) \geq k$. There exists $p \leq 2q$ such that $(s,h) \not\models \# \text{pred}_{\bigcirc}(x_i) \geq p + 1$ and $(s,h) \models \# \text{pred}_{\bigcirc}(x_i) \geq p + 1$. We deduce the upper bound $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \leq p$ and the lower bound $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \geq k + p$. Using the partition $\text{pred}_{\bigcirc}(s,h,j) = \text{pred}_{\bigcirc}(s,h,j) \cup \text{pred}_{\bigcirc}(s,h,j)$, we derive the lower bound $\text{card}(\text{pred}_{\bigcirc}(s,h,j)) \geq k$.

\[ \square \]

**Lemma 2.21** Let $\alpha_1, \alpha_2 \in \text{N}$ and $X, X', Y_0$ be finite sets such that $X \subseteq X' \sim \alpha_1 + \alpha_2$, $Y_0$ holds. Then there are two finite sets $Y, Y'$ such that $Y_0 = Y \cup Y'$, $X \sim \alpha_1$, $Y$ and $X' \sim \alpha_2$ $Y'$ hold.
Proof By Proposition 2.18 item 3, we have two cases: either \( \text{card}(X \uplus X') = \text{card}(Y_0) < \alpha_1 + \alpha_2 \) or \( \text{card}(X \uplus X') \geq \alpha_1 + \alpha_2 \) and \( \text{card}(Y_0) \geq \alpha_1 + \alpha_2 \).

The case \( \text{card}(X \uplus X') = \text{card}(Y_0) < \alpha_1 + \alpha_2 \) is easy: we have \( \text{card}(X) \leq \text{card}(X) + \text{card}(X') \leq \text{card}(Y_0) \), then for \( Y \), we choose any subset of \( Y_0 \) such that \( \text{card}(Y) = \text{card}(X) \). Then we define \( X'' = Y_0 \backslash Y \) and we get \( \text{card}(X') = \text{card}(Y_0) - \text{card}(X') = \text{card}(X \uplus X') - \text{card}(X) = \text{card}(X') \). Then we have both \( X \sim_{\alpha_1} Y \) and \( X' \sim_{\alpha_2} Y'' \).

Let us consider the case \( \text{card}(X \uplus X') \geq \alpha_1 + \alpha_2 \) and \( \text{card}(Y_0) \geq \alpha_1 + \alpha_2 \). We have four sub-cases:

- the case \( \text{card}(X) < \alpha_1 \) and \( \text{card}(X') < \alpha_2 \) is impossible because it contradicts \( \text{card}(X \uplus X') \geq \alpha_1 + \alpha_2 \);
- in the case \( \text{card}(X) \geq \alpha_1 \) and \( \text{card}(X') < \alpha_2 \), let \( Y'' \) be any subset of \( Y_0 \) such that \( \text{card}(Y') = \text{card}(X') \) and \( Y = Y_0 \backslash Y'' \). We have \( \text{card}(X') = \text{card}(Y') \) hence \( X' \sim_{\alpha_2} Y'' \). We have \( \text{card}(X) \geq \alpha_1 \) and \( \text{card}(Y) = \text{card}(Y_0) - \text{card}(Y') \geq (\alpha_1 + \alpha_2) - \alpha_2 = \alpha_1 \) hence \( X \sim_{\alpha_1} Y \);
- the case \( \text{card}(X) < \alpha_1 \) and \( \text{card}(X') \geq \alpha_2 \) is obtained by symmetry from the previous case;
- in the case \( \text{card}(X) \geq \alpha_1 \) and \( \text{card}(X') \geq \alpha_2 \), let \( Y \) be any subset of \( Y_0 \) s.t. \( \text{card}(Y) = \alpha_1 \) and \( Y' = Y_0 \backslash Y \). We have \( \text{card}(X) \geq \alpha_1 \) and \( \text{card}(Y) = \alpha_1 \) hence \( X \sim_{\alpha_1} Y \). We have \( \text{card}(X') \geq \alpha_2 \) and \( \text{card}(Y') = \text{card}(Y_0) - \text{card}(Y) \geq (\alpha_1 + \alpha_2) - \alpha_1 = \alpha_2 \) hence \( X' \sim_{\alpha_2} Y'' \).

\[ \square \]

### B Proofs of Section 3

**Proposition 3.13** Let \( u, v \in \mathbb{N} \). For \((\text{T10})-\text{(T20)}\) defined as

\[(\text{T10}) \quad u \in s(V) \iff v \in s'(V);\]
\[(\text{T11}) \quad u \in h(s(V)) \iff v \in h'(s'(V));\]
\[(\text{T12}) \quad u \in p\backslash s(h) \iff v \in p\backslash s'(h');\]
\[(\text{T13}) \quad u \in \overline{s}(h) \iff v \in \overline{s}'(h');\]
\[(\text{T14}) \quad u \in \text{pred}(s, h, i) \iff v \in \text{pred}(s', h', i) \text{ for any } i \in [1, q];\]
\[(\text{T15}) \quad u \in \text{pred}(s, h) \iff v \in \text{pred}(s', h');\]
\[(\text{T16}) \quad u \in \text{loop}(s, h) \iff v \in \text{loop}(s', h');\]
\[(\text{T17}) \quad u \in \text{rem}(s, h) \iff v \in \text{rem}(s', h');\]
\[(\text{T18}) \quad u \in \text{pred}(s, h, i) \iff v \in \text{pred}(s', h', i) \text{ for any } i \in [1, q];\]
\[(\text{T19}) \quad u \in \text{loop}(s, h) \iff v \in \text{loop}(s', h');\]
\[(\text{T20}) \quad u \in \text{rem}(s, h) \iff v \in \text{rem}(s', h');\]
\[(\text{T21}) \quad u \in p\backslash m \iff v \in p\backslash m';\]

the following propositions hold:

1. \((\text{T2})\) implies \((\text{T10})\);
2. \((\text{T3})\) implies \((\text{T11})\);
3. \((\text{T2})-\text{T3})\) imply \((\text{T12})\);
4. \((\text{T2})-\text{T4})\) imply \((\text{T14})\);
5. \((\text{T2})-\text{T6})\) imply \((\text{T10})-\text{T20})\);
6. \((\text{T1})-\text{T3})\) imply \((\text{T21})\).

**Proof** The proofs are easy. Here is a summary of the arguments:

- \((\text{T10})\) is a direct consequence of \((\text{T2})\);
- \((\text{T11})\) is a direct consequence of \((\text{T3})\);
- \((\text{T12})\) we use the identity \( p\backslash s(h) = s(V) \cup h(s(V)) \), \((\text{T10})\) and \((\text{T11})\);
- \((\text{T13})\) we use the identity \( s(h) = p\backslash s(h) \cap \text{dom}(h) \), \((\text{T12})\) and \((\text{T4})\);
- \((\text{T14})\) just another way to write \((\text{T5})\);
- \((\text{T15})\) we use the identity \( \text{pred}(s, h) = \bigcup_i \text{pred}(s, h, i) \) and \((\text{T14})\);
- \((\text{T16})\) just another way to write \((\text{T6})\);
- \((\text{T17})\) with \( \text{rem}(s, h) = \text{dom}(h) \backslash \text{pred}(s, h, i) \cup \text{loop}(s, h) \), \((\text{T4})\), \((\text{T15})\), \((\text{T16})\);
- \((\text{T18})\) we use \( \text{pred}(s, h, i) = \text{rem}(s, h, i) \cup p\backslash s(h) \) and \((\text{T14})\), \((\text{T12})\);
- \((\text{T19})\) we use \( \text{loop}(s, h) = \text{loop}(s, h) \cup p\backslash s(h) \) and \((\text{T16})\), \((\text{T12})\);
- \((\text{T20})\) we use \( \text{rem}(s, h) = \text{rem}(s, h) \cup p\backslash s(h) \) and \((\text{T17})\), \((\text{T12})\);
- \((\text{T21})\) we use the identity \( p\backslash m = p\backslash s(h) \cup \{l\} \), \((\text{T12})\) and \((\text{T1})\).

\[ \square \]
Proposition 3.15 The following inclusions hold:
1. \(\mathfrak{N} \subseteq \mathfrak{P}(s, h) \times \mathfrak{P}(s', h')\)
2. \(\mathfrak{T} \cap \mathfrak{P}(s, h) \times N \subseteq \mathfrak{N}\)
3. \(\mathfrak{T} \cap N \times \mathfrak{P}(s', h') \subseteq \mathfrak{N}\)
4. \(\mathfrak{N} \subseteq \mathfrak{P}(m) \times \mathfrak{P}(m')\)

Proof Inclusions 1 and 4 are trivial. Let us consider Inclusion 2, hence let \(u, v\) be such that \(u \in \mathfrak{N}\) and \(v \in \mathfrak{P}(s, h)\). Since \(\mathfrak{P}(s, h) = s(\mathfrak{P}) \cup h(\mathfrak{P})\) we have two cases:
- either \(u = s(x)\) for some \(x \in [1, q]\), and \(v = s'(x)\) by (T2). Hence \(u\) and \(v\) satisfy (92) and thus we derive \(u \in N\); or
- \(u = h(s(x))\) for some \(x \in [1, q]\), and \(v = h'(s'(x))\) by (T3). Hence \(u\) and \(v\) satisfy (93) and thus we deduce \(u \in \mathfrak{N}\).

Inclusions 3 and 6 are proved in a similar way. □

Proposition 3.16 The following properties hold:
1. The relation \(\mathfrak{T}\) restricted to \(\mathfrak{P}(s, h) \times \mathfrak{P}(s', h')\) is functional and injective;
2. The relation \(\mathfrak{T}\) restricted to \(\mathfrak{P}(m) \times \mathfrak{P}(m')\) is functional and injective;
3. For any \(u \not\in \text{dom}(h) \cup \mathfrak{P}(s, h), v \not\in \text{dom}(h') \cup \mathfrak{P}(s', h')\), we have \(u \not\in \mathfrak{T} \lor v\);\n4. For any \(u \not\in \text{dom}(h) \cup \mathfrak{P}(m), v \not\in \text{dom}(h') \cup \mathfrak{P}(m')\), we have \(u \not\in \mathfrak{T} \lor v\).

Proof To show that \(\mathfrak{T}\) is functional, we prove that for all \(u \in \mathfrak{P}(s, h)\), for all \(v, w \in \mathfrak{P}(s', h')\), \(u \in \mathfrak{T} \lor v\) and \(u \not\in \mathfrak{T} \lor w\). If \(u = s(x)\), then we must have \(v = s'(x) = u\), and if \(u = h(s(x))\), then we must have \(v = h'(s'(x)) = w\). By symmetric arguments, \(\mathfrak{T}\) is injective.

The proof that \(\mathfrak{T}\) restricted to \(\mathfrak{P}(m) \times \mathfrak{P}(m')\) is functional and injective (Property 2) is similar.

Let us now establish Property 3. We consider \(u, v \in \mathfrak{N}\) such that \(u \not\in \text{dom}(h) \cup \mathfrak{P}(s, h)\) and \(v \not\in \text{dom}(h') \cup \mathfrak{P}(s', h')\). Conditions (T2)–(T3) hold because \(u \not\in \mathfrak{P}(\mathfrak{V}) \cup h(\mathfrak{P}(\mathfrak{V}))\) and \(v \not\in \mathfrak{P}(\mathfrak{V}) \cup h'(\mathfrak{P}(\mathfrak{V}))\). Conditions (T4)–(T6) hold because \(u \not\in \text{dom}(h) \land v \not\in \text{dom}(h')\). The proof of Property 4 follows a similar pattern. □

Theorem 3.17 \(m \equiv m'\) if and only if \(\mathfrak{N} \subseteq \mathfrak{T}\).

Proof Let us show the only if part first. We assume \(m \equiv m'\) and we prove \(\mathfrak{N} \subseteq \mathfrak{T}\) by case analysis on \(\mathfrak{N} \mathfrak{N}\):\n
(911) let us check \(I \mathfrak{T} I'\): Condition (T1) is trivial; for Condition (T2), \(l = s(x)\) iff \(s(h) \equiv l\); \(x = x\) if \(s'(x) = x\); \(u \equiv l' = s'(x)\); for Condition (T3), \(l = h(s(x))\) iff \(s(h) \equiv l\); \(x = x\) if \(s'(x) = x\); \(u \equiv l' = h'(s'(x))\); for Condition (T4), \(l \in \text{dom}(h)\) iff \(s(h) \equiv \text{alloc}(x)\) iff \(s'(x) = x\); \(u \equiv u\) if \(l' \in \text{dom}(h)\); for Condition (T5), \(h(l) = s(x)\) iff \(s(h) \equiv l\); \(u \equiv u\) if \(s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); for Condition (T6), \(h(l) = t\) iff \(s(h) \equiv l\); \(u \equiv u\) if \(s'(x) = x\); \(u \equiv l' \equiv t\).

(912) let us check \(s(x) \mathfrak{T} s'(x)\); for Condition (T1), \(s(x) = l\) iff \(s'(x) = l'\) (as previously); for Condition (T2), \(s(x) = s(x)\) iff \(s(h) \equiv x = x\); \(s'(x) = x\); \(u \equiv u\) if \(s'(x) = x\); \(u \equiv u\) if \(s'(x) = s'(x)\); for Condition (T3), \(s(x) = h(s(x))\) iff \(s(h) \equiv x = x\); \(s'(x) = x\); \(u \equiv u\) if \(s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); for Condition (T4), \(s(x) \in \text{dom}(h)\) iff \(s(h) \equiv \text{conv}(x)\) iff \(s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); \(u \equiv x\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\); for Condition (T5), \(h(s(x)) = s(x)\) iff \(s(h) \equiv x = x\); \(x = x\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\); for Condition (T6), \(h(s(x)) = s(x)\) iff \(s(h) \equiv x = x\); \(x = x\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\); for Condition (T4), \(h(s(x)) \in \text{dom}(h)\) iff \(s(h) \equiv \text{alloc}(x)\) iff \(s'(x) = x\); \(l' \equiv x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\); for Condition (T5), \(h(s(x)) = s(x)\) iff \(s(h) \equiv x = x\); \(x = x\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\); for Condition (T6), \(h(s(x)) = s(x)\) iff \(s(h) \equiv x = x\); \(x = x\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\); for Condition (T4), \(h(s(x)) \in \text{dom}(h)\) iff \(s(h) \equiv \text{conv}(x)\) iff \(h'(s'(x)) = h'(s'(x))\); for Condition (T5), \(h(s(x)) = s(x)\) iff \(s(h) \equiv x = x\); \(x = x\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\); for Condition (T6), \(h(s(x)) = s(x)\) iff \(s(h) \equiv x = x\); \(x = x\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(l' \equiv s'(x) = x\); \(u \equiv u\) if \(h'(s'(x)) = s'(x)\).
Let us now tackle the if part. We assume $\mathcal{R} \subseteq \mathcal{T}$. Hence we have $l \mathcal{T} l', s(x_i) \mathcal{T} s'(x_i)$ for any $i \in [1,q]$, and $h(s(x_i)) \mathcal{T} h'(s'(x_i))$ for any $i \in [1,q]$ such that $s(x_i) \in \text{dom}(h)$ (and $s'(x_i) \in \text{dom}(h')$). To establish $m \approx_m m'$, we consider a formula $\mathcal{B} \in \text{Basic}$ and we show that $(s,h) \models \mathcal{B}$ implies $(s',h') \models_{\mathcal{T}} \mathcal{B}$. The reverse implication can be proved by symmetric arguments. We proceed by a case analysis on $\mathcal{B}$:

- $\mathcal{B}$ is $s_i = x_j$: if $(s,h) \models s_i = x_j$, then $s(x_i) = s(x_j)$. Using the instance of $(\mathcal{L}_2)$ for $s(x_i) \mathcal{T} s'(x_i)$ with parameter $j$, we get $s'(x_i) = s(x_j)$. We deduce $(s',h') \models_{\mathcal{T}} s_i = x_j$.

- $\mathcal{B}$ is $s_i \xrightarrow{\text{toalloc}} x_j$: if $(s,h) \models s_i \xrightarrow{\text{toalloc}} x_j$ then $h(s(x_i)) = s(x_j)$. Using the instance of $(\mathcal{L}_2)$ for $h(s(x_i)) \mathcal{T} h'(s'(x_i))$ with parameter $j$, we get $h'(s'(x_i)) = s'(x_j)$. We deduce $(s',h') \models_{\mathcal{T}} s_i \xrightarrow{\text{toalloc}} x_j$.

- $\mathcal{B}$ is $\text{toalloc}(s_i; x_j)$: if $(s,h) \models \text{toalloc}(s_i; x_j)$ then $h(s(x_i))) = h(s(x_i)))$. Using the instance of $(\mathcal{L}_3)$ for $h(s(x_i)) \mathcal{T} h'(s'(x_i))$ with parameter $j$, we get $h'(s'(x_i)) = h'(s'(x_i))$. We deduce $(s',h') \models_{\mathcal{T}} \text{toalloc}(s_i; x_j)$.

- $\mathcal{B}$ is $\text{toloop}(x_i)$: if $(s,h) \models \text{toloop}(x_i)$ then $h'(s'(x_i)) = h(s(x_i)))$. By $(\mathcal{L}_6)$ for $h(s(x_i)) \mathcal{T} h'(s'(x_i))$ with parameter $j$, we get $h'(s'(x_i)) = h'(s'(x_i))$. We deduce $(s',h') \models_{\mathcal{T}} \text{toloop}(x_i)$.

- $\mathcal{B}$ is $\text{alloc}(u)$: if $(s,h) \models \text{alloc}(u)$ then $l \in \text{dom}(h)$. Using $(\mathcal{L}_4)$ for $h(s(x_i)) \mathcal{T} h'(s'(x_i))$, we get $h'(s'(x_i)) \in \text{dom}(h')$. We deduce $(s',h') \models_{\mathcal{T}} \text{alloc}(u)$; $B$ is $s_i = u$: if $(s,h) \models s_i = u$ then $s(x_i) = l$. Using $(\mathcal{L}_1)$ for $s(x_i) \mathcal{T} s'(x_i)$, we get $s'(x_i) = l'$.

- $\mathcal{B}$ is $s_i \xrightarrow{u} u$: if $(s,h) \models s_i \xrightarrow{u} u$ then $h(s(x_i)) = s(x_i)$. Using $(\mathcal{L}_1)$ for $h(s(x_i)) \mathcal{T} h'(s'(x_i))$, we get $h'(s'(x_i)) = l'$. We deduce $(s',h') \models_{\mathcal{T}} s_i \xrightarrow{u} u$.

Proposition 3.18 If $m \approx_m m'$ then the following properties hold:

1. The relation $\mathcal{R}$ is total and surjective between $\mathcal{P}(s,h)$ and $\mathcal{P}(s',h')$;
2. The relation $\mathcal{R}'$ is total and surjective between $\mathcal{P}(m)$ and $\mathcal{P}(m')$.

Proof Let us consider Property 1. To show that $\mathcal{R}$ is total, we prove that for all $u \in \mathcal{P}(s,h)$, there is $v \in \mathcal{P}(s',h')$ such that $u \mathcal{R} v$. If $u = s(x_i)$ then choose $v = s'(x_i)$; and if $u = h(s(x_i))$ then $s(x_i) \in \text{dom}(h)$. But we have $s(x_i) \mathcal{R} s'(x_i)$ by definition of $\mathcal{R}$, hence by Theorem 3.17 we deduce $s(x_i) \mathcal{T} s'(x_i)$. As a consequence $(s(x_i),s'(x_i))$ verify $(\mathcal{L}_4)$ and thus $s'(x_i) \in \text{dom}(h')$. We choose $v = h'(s'(x_i))$ and we get $u \mathcal{R} v$ and $v \in \mathcal{P}(s',h')$.

By symmetric arguments, $\mathcal{R}$ is surjective. The proof that $\mathcal{R}'$ is total and surjective (Property 2) is similar.

Proposition 3.21 If $m$ and $m'$ satisfy $m \approx m'$, then $\mathcal{I}$ is a total relation on $N$: for any $u \in N$, there exists $v \leq \text{maxval}(s',h') + 1$ such that $u \mathcal{I} v$.

Proof Since $\approx \subseteq \approx_\geq$, we have $\mathcal{R} \subseteq \mathcal{I}$ by Lemma 3.19. Let us consider $u \in N$. We have to show that there exists $v \in N$ such that $u \mathcal{I} v$ holds. We determine the value of $v$ according to the first condition that holds in the following list:

- if $u \in \mathcal{P}(s,h)$ then let us define $v$ to be the unique location in $\mathcal{P}(s',h')$ such that $u \mathcal{R} v$, see Lemma 3.19. We deduce $u \mathcal{I} v$. The relation $v \leq \text{maxval}(s',h') + 1$ holds because $v \in \mathcal{P}(s',h')$;
if \( u \in \text{pred}_{\sigma}(s,h,j) \) for some \( j \in [1,q] \) then we know that \( \text{pred}_{\sigma}(s,h,j) \) is not empty. We have \( \text{pred}_{\sigma}(s,h,j) = \text{pred}_{\sigma}(s',h',j) \) by Proposition 3.10 hence \( \text{pred}_{\sigma}(s',h',j) \) is not empty either. We choose any \( v \in \text{pred}_{\sigma}(s',h',j) \).

The relation \( v \leq \maxval(s',h') + 1 \) holds because \( v \in \text{dom}(h') \). Let us check that \( u \not\in \text{v} \) holds by establishing Properties (32–6) for \( u/v \). We have \( u \in \text{pred}_{\sigma}(s,h,j) \) and \( v \in \text{pred}_{\sigma}(s',h',j) \). As a consequence we deduce \( u \not= p\overline{\sigma}(s,h) \) and \( v \not= p\overline{\sigma}(s',h') \). Hence Properties (32–3) hold. We also have \( u \in \text{dom}(h) \) and \( v \in \text{dom}(h') \) hence Property (34) holds. We have \( h(u) = s(x_j) \) and \( h'(v) = s'(x_j) \). We deduce \( h(u) \not\in \text{dom}(h') \) and thus \( h(u) \not\in \text{v} \). Since (32) holds for \( h(u)/h'(v) \), we deduce that Property (35) holds for \( u/v \). Let us prove Property (36) for \( u/v \): the identity \( u = h(u) \) implies \( u = s(x_j) \) which contradicts \( u \not= p\overline{\sigma}(m) \). Hence \( u \not= h(u) \) and for the similar reasons, \( v \not= h'(v) \);

if \( u \in \text{loop}_{\sigma}(s,h) \) then we proceed in a way similar to the previous case;

in the remaining cases we have \( u \not= (\text{dom}(h) \cup p\overline{\sigma}(s,h)) \). Let us define \( v = \maxval(s',h') + 1 \).

We have \( v \not= (\text{dom}(h') \cup p\overline{\sigma}(s',h')) \) and by Proposition 3.16 item 3, we deduce \( u \not\in \text{v} \). \( \Box \)

**Proposition 3.22** Let us assume \( \mathcal{N} \subseteq \mathcal{T}^1 \) (or equivalently \( m \geq m' \)). Then the following statements are equivalent:

1. \( \mathcal{N} \subseteq \mathcal{D}_1 \land \mathcal{D}_2 \);
2. \( \mathcal{N} \subseteq \mathcal{D}_1 \land \mathcal{D}_2 \);
3. \( \mathcal{N}_1 \subseteq \mathcal{T}_1 \land \mathcal{T}_2 \);
4. \( m_1 \geq m'_1 \land m_2 \geq m'_2 \).

**Proof** Let us review the easy implications first. Obviously, statement 3 and 4 are equivalent by Theorem 3.17. Then statement 2 implies statement 3 by the two following deductions:

\( \mathcal{N}_1 \subseteq \mathcal{N}_2 \land \mathcal{N}_3 \subseteq \mathcal{N}_4 \land \mathcal{N}_2 \subseteq \mathcal{D}_1 \land \mathcal{D}_2 \).

Let us now show that statement 1 implies statement 2. So we assume \( \mathcal{N} \subseteq \mathcal{D}_1 \land \mathcal{D}_2 \).

We show that \( \mathcal{N} \subseteq \mathcal{T}_1 \), the case \( \mathcal{N} \subseteq \mathcal{T}_2 \) being treated in a similar way. So let us assume \( u,v \) such that \( u \not\in \mathcal{T}^1 \) and let us show that \( u \not\in \mathcal{T}_1 \). Because \( \mathcal{N} \subseteq \mathcal{T}^1 \) we have \( u \not\in \mathcal{T}_1 \). Because \( \mathcal{N} \subseteq \mathcal{D}_1 \land \mathcal{D}_2 \) we have \( \mathcal{N} \subseteq \mathcal{D}_1 \). Let us show that \( u/v \) verify \((11–6)\) with respect to \( m_1/m'_1 \); Properties (1,2) hold because \( u \not\in \mathcal{T}^1 \); Property (33): \( u = h_1(s(x_j)) \) if \( s(x_j) \in \text{dom}(h_1) \) and \( u = h(s(x_j)) \) if \( s'(x_j) \in \text{dom}(h') \) and \( v = h'(s'(x_j)) \) if \( v = h_1'(s'(x_j)) \) because \( s(x_j) \in \mathcal{D}_1 \) \( s'(x_j) \) (which comes from \( s(x_j) \)) and \( u \not\in \mathcal{T}^1 \);

Property (34): \( u \not\in \text{dom}(h_1) \) if \( \text{v} \not\in \text{dom}(h'_1) \) because \( u \not\in \mathcal{T}_1 \);

Property (35): \( h_1(u) = s(x_j) \) if \( h(u) = s(x_j) \) and \( h'(v) = s'(x_j) \) if \( h_1'(v) = s'(x_j) \) because \( u \not\in \mathcal{T}_1 \);

Property (36): \( h_1(u) = u \) if \( u \not\in \text{dom}(h_1) \) and \( h(u) = u \) if \( v \not\in \text{dom}(h_1) \) and \( h'(v) = v \) if \( h_1'(v) = v \) because \( u \not\in \mathcal{T}_1 \).

Let us finish by showing that statement 3 implies statement 1. So we assume \( \mathcal{N}_1 \subseteq \mathcal{T}_1 \) and \( \mathcal{N}_2 \subseteq \mathcal{T}_2 \).

Let us show the inclusion \( \mathcal{N}_1 \subseteq \mathcal{D}_1 \land \mathcal{D}_2 \). We assume \( u,v \) such that \( u \not\in \mathcal{N}_1 \). Let us show that \( u \not\in \mathcal{D}_1 \), the case of \( u \not\in \mathcal{D}_2 \) being treated in a similar way. Notice the inclusion \( \mathcal{T}_1 \subseteq \mathcal{D}_1 \) that always holds by Definition 3.14. For \( u \not\in \mathcal{N}_1 \) there are three cases:

- either \( u = l \) and \( v = l' \). We have \( l \not\in \mathcal{N}_1 \land \mathcal{N}_2 \subseteq \mathcal{T}_1 \land \mathcal{T}_2 \subseteq \mathcal{D}_1 \), thus we get \( l \land \mathcal{D}_1 \) hence \( u \not\in \mathcal{D}_1 \);

- or \( u = s(x_i) \) and \( v = s'(x_i) \) for some \( i \in [1,q] \). We have \( s(x_i) \subseteq \mathcal{N}_1 \land s'(x_i) \subseteq \mathcal{N}_2 \subseteq \mathcal{D}_1 \), thus we get \( u \not\in \mathcal{D}_1 \);

- or \( u = h(s(x_i)) \) and \( v = h'(s'(x_i)) \) for some \( i \in [1,q] \). Let us assume \( u \not\in \text{dom}(h_1) \) and let us prove \( v \not\in \text{dom}(h_1') \). From \( h(s(x_i)) \in \text{dom}(h_1) \) deduce \( s(x_i) \in \text{dom}(h) = \text{dom}(h_1) \). Hence we have two cases:

- either \( s(x_i) \not\in \text{dom}(h_1) \). In this case we have \( h(s(x_i)) = h(s(x_i)) = u \not\in \text{dom}(h_1) \) and \( s(x_i) \not\in \text{dom}(h_1) \). From \( \mathcal{N}_2 \subseteq \mathcal{T}_2 \) we deduce \( s(x_i) \in \mathcal{T}_1 \land s'(x_i) \not\in \text{dom}(h_1') \). By (34) (twice) we get \( h_1'(s'(x_i)) \in \text{dom}(h_1') \) and \( s'(x_i) \not\in \text{dom}(h_1') \). We deduce \( v = h'(s'(x_i)) = h_1'(s'(x_i)) \not\in \text{dom}(h_1') \);
or \( s(x_t) \in \text{dom}(h_2) \). In this case we have \( h_2(s(x_t)) = h(s(x_t)) = u \in \text{dom}(h_1) \) and \( s(x_t) \in \text{dom}(h_2) \). Since \( \text{dom}(h) = \text{dom}(h_1) \cup \text{dom}(h_2) \) we deduce \( h(s(x_t)) \in \text{dom}(h) \) and \( h_2(s(x_t)) \notin \text{dom}(h_2) \). Because \( M(h) \subseteq M(h_2) \) and \( M(h_1) \subseteq M(h_2) \) we have \( s(x_t) \not\in \text{dom}(h'_2) \), \( h(s(x_t)) \not\in \text{dom}(h'_2) \) and \( h_2(s(x_t)) \not\in \text{dom}(h'_2) \). Hence by (T4) (three times) we deduce \( s'(x_t) \in \text{dom}(h'_2) \), \( h'(s(x_t)) \in \text{dom}(h'_2) \) and \( h'_2(s(x_t)) \notin \text{dom}(h'_2) \). As a consequence \( v = h'(s(x_t)) = h'_2(s'(x_t)) \notin \text{dom}(h'_2) \) and \( v = h'(s(x_t)) \in \text{dom}(h') \). We conclude \( v \in \text{dom}(h'_1) \).

**Lemma 3.25** Let \( \alpha \geq 1 \) and let \((s, h, l)\) and \((s', h', l')\) be two pointed memory states. Let \( l_1, l_2, l'_1, l'_2 \in \mathbb{N} \) be such that \( l_1 \notin \text{dom}(h) \) and \( l'_1 \notin \text{dom}(h') \). We assume that one of the conditions below holds:

(C1) \( l_1/l'_1 \) verify \((T1-3), l_2/l'_2 \) verify \((T1-6)\), and \( l_2 = l_1 \) iff \( l'_2 = l'_1 \);

(C2) \( l_1 \notin s(V) \), \( l_1/l'_1 \) verify \((T1-3), l_2/l'_2 \) verify \((T2)\), and \( l_2 = l_1 \) iff \( l'_2 = l'_1 \).

If \((s, h, l) \geq_\alpha (s', h', l')\) then \((s, h) \equiv [l_1 \mapsto l_2, l) \geq_\beta (s', h' \equiv [l'_1 \mapsto l'_2, l')\) where \( \beta = \alpha - 1 \) if \( l_1 \in s(V) \), and \( \beta = \alpha \) otherwise.

**Proof** We denote \( h \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \) by \( h'_1 \equiv [l'_1 \mapsto l'_2] \). According to Proposition 3.10, we have to establish

\[
(s, h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \) (B.1)

together with \( \beta \)-equipoherence constraints:

\[
\text{pred}(s, h, l) \equiv_\beta \text{pred}(s', h', l') \) (B.2)

\[
\text{loop}(s, h, l) \equiv_\beta \text{loop}(s', h', l') \) (B.3)

\[
\text{rem}(s, h, l) \equiv_\beta \text{rem}(s', h', l') \) (B.4)

We start with basic Equivalence (B.1). We have to show that for any \( B \in \text{Basic}^a \), \((s, h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \equiv_\nu B \) iff \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu B \). We prove that \((s, h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \equiv_\nu B \) implies \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu B \). The reverse implication can be established in a symmetric way. Note that all hypotheses are symmetric: when \( l_1/l'_1 \) verify \((T2)\), we have \( l_1 \notin s(V) \) iff \( l'_1 \notin s'(V) \). We proceed by a case analysis on \( B \):

- **B is \( x_i \equiv x_j \):** using \((s, h, l) \geq_\alpha (s', h', l')\), we derive \((s, h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \equiv_\nu x_i \equiv x_j \) iff \((s, h) \equiv [l_1 \mapsto l_2, x_i \equiv x_j \) iff \( s(x_i) = x_j \) iff \( h(s(x_i)) = s(x_j) \). We derive \( h'(s'(x_i)) = s'(x_j) \) (because \((s, h, l) \geq_\alpha (s', h', l')\) and we thus also have \( h'_1 \equiv [l'_1 \mapsto l'_2] (s'(x_i)) = s'(x_j) \)). Hence \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu x_i \equiv x_j \).

- **B is \( x_i \equiv x_i \):** let us suppose \((s, h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \equiv_\nu x_i \equiv x_j \) and let us show \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu x_i \equiv x_j \).

For \((s, h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \equiv_\nu x_i \equiv x_j \) we have two cases:

- \( h(s(x_i)) = s(x_j) \) (because \((s, h, l) \geq_\alpha (s', h', l')\) and thus also have \( h'_1 \equiv [l'_1 \mapsto l'_2] (s'(x_i)) = s'(x_j) \).

- \( l_1 = s(x_i) \) and \( l_2 = s(x_j) \). Since \( l_1/l'_1 \) and \( l_1'/l'_1 \) verify \((T2)\) in both \( (C1) \) and \( (C2) \), we get \( l'_2 = s'(x_j) \) and \( l'_2 = s'(x_j) \) and finally \( s'(h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu x_i \equiv x_j \).

In both cases we obtain \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu x_i \equiv x_j \).

- **B is \( \text{conv}(x_i, x_j) \):** let us suppose \((s, h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \equiv_\nu \text{conv}(x_i, x_j) \) and prove that \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu \text{conv}(x_i, x_j) \).

From \( h_1 \equiv [l_1 \mapsto l_2, h \equiv [l'_1 \mapsto l'_2] \equiv_\nu \text{conv}(x_i, x_j) \) we get four cases:

- \( s(x_i) \in \text{dom}(h) \) and \( h(s(x_i)) = h(s(x_i)) \). Then \((s, h) \equiv_\nu \text{conv}(x_i, x_j) \) from which we get \( s'(h') \equiv_\nu \text{conv}(x_i, x_j) \) and thus also \( (s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu \text{conv}(x_i, x_j) \);

- \( h(s(x_i)) = l_2 \) and \( s(x_j) = l_1 \). If \((C1) \) holds then \( l_2/l'_2 \) verify \((T3)\) and \( l_1/l'_1 \) verify \((T2)\) and we get \( h'(s'(x_i)) = l'_2 \) and \( s'(x_j) = l'_1 \). Hence \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu \text{conv}(x_i, x_j) \).

If \((C2) \) holds then \( l(s(x_i)) = l_1 \) contradicts \( l \notin s(V) \);

- the case \( s(x_i) = x_i \) and \( h(s(x_i)) = l_2 \) is symmetric to the previous one;

- \( s(x_i) = x_i \) and \( h(s(x_i)) = l_1 \). In case \((C1) \), \( l_1/l'_1 \) verify \((T2)\) and thus we get \( s'(x_i) = s'(x_j) = l'_1 \) and then \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu \text{conv}(x_i, x_j) \).

(C2) implies \( l \notin s(V) \) which contradicts \( s(x_i) = l_1 \).

In all four cases we have \((s', h'_1 \equiv [l'_1 \mapsto l'_2] \equiv_\nu \text{conv}(x_i, x_j) \).
\(B\) is \(\text{btwn}(x_i, x_j)\): let us assume \((s, h_{1\to 2}) \models I\ \text{btwn}(x_i, x_j)\) and let us prove that \((s', h'_{1\to 2}) \models I'\ \text{btwn}(x_i, x_j)\). We have four cases:

- \(h(\text{post}(s(x_i))) = s(x_j)\). Then \((s, h) \models I\ \text{btwn}(x_i, x_j)\) from which we get \((s', h') \models I'\ \text{btwn}(x_i, x_j)\);
- \(h(x_i) = l_1\) and \(h(x_j) = s(x_j)\). In both (C1) and (C2), \(l_1/\ell_1\) verify (T3) and \(l_2/\ell_2\) verify (T2). Hence we get \(h'(s'(x_i)) = \ell_1'\) and \(l_2' = s'(x_j)\). Thus \((s', h'_{1\to 2}) \models I'\ \text{btwn}(x_i, x_j)\);
- \(l_1 = s(x_i)\) and \(h(l_2) = s(x_j)\). In (C1), \(l_1/\ell_1\) verify (T2) and \(l_2/\ell_2\) verify (T5), hence we get \(l_1' = s'(x_i)\) and \(h'(l_2') = s'(x_j)\). Thus \((s', h'_{1\to 2}) \models I'\ \text{btwn}(x_i, x_j)\). In case (C2), \(l_1 = l_2 = s(x_i)\) and let us prove \((s', h'_{1\to 2}) \models I'\ \text{btwn}(x_i, x_j)\).

In all four cases we have \((s', h'_{1\to 2}) \models I'\ \text{btwn}(x_i, x_j)\);

\(B\) is \(\text{toalloc}(x_i)\): let us assume \((s, h_{1\to 2}) \models I\ \text{toalloc}(x_i)\) and let us prove \((s', h'_{1\to 2}) \models I'\ \text{toalloc}(x_i)\). We get four cases:

- \(h(\text{post}(s(x_i))) \in \text{dom}(h)\). Then \((s, h) \models I\ \text{toalloc}(x_i)\) from which we get \((s', h') \models I'\ \text{toalloc}(x_i)\);
- \(h(\text{post}(s(x_i))) = l_1\). \(l_1/\ell_1\) verify (T3) in both (C1) and (C2), thus we get \(h'(s'(x_i)) = \ell_1'\) and thus \((s', h'_{1\to 2}) \models I'\ \text{toalloc}(x_i)\);
- \(l_1 = s(x_i)\) and let us prove \((s', h'_{1\to 2}) \models I'\ \text{toalloc}(x_i)\).

In all four cases we have \((s', h'_{1\to 2}) \models I'\ \text{toalloc}(x_i)\);

\(B\) is \(\text{toloop}(x_i)\): let us suppose \((s, h_{1\to 2}) \models I\ \text{toloop}(x_i)\) and let us prove \((s', h'_{1\to 2}) \models I'\ \text{toloop}(x_i)\). We get four cases:

- \(h(\text{post}(s(x_i))) = h(\text{post}(s(x_i)))\). Then \((s, h) \models I\ \text{toloop}(x_i)\) from which we get \((s', h') \models I'\ \text{toloop}(x_i)\);
- \(h(\text{post}(s(x_i))) = l_1\). \(l_1/\ell_1\) verify (T3) in both (C1) and (C2), thus we get \(h'(s'(x_i)) = \ell_1'\) and \(l_1 = l_2'\) hold in both (C1) and (C2). Hence we get \(l_1' = l_1\) and \(l_2' = \ell_2\) and thus \((s', h'_{1\to 2}) \models I'\ \text{toloop}(x_i)\).

In all four cases we have \((s', h'_{1\to 2}) \models I'\ \text{toloop}(x_i)\);

\(B\) is \(u \rightarrow u\): let us suppose \((s, h_{1\to 2}) \models I\ u \rightarrow u\) and let us prove that \((s', h'_{1\to 2}) \models I'\ u \rightarrow u\). We get two cases:

- \(h(l) = l\). We derive \((s, h) \models I\ u \rightarrow u\) then \((s', h') \models I'\ u \rightarrow u\) and hence \((s', h'_{1\to 2}) \models I'\ u \rightarrow u\);
- \(l_1 = l_2\). \(l_1/\ell_1\) verify (T1), and \(l_2 = l_1\) holds in both (C1) and (C2). Hence we get \(l_1' = l_1\) and \(l_2' = \ell_2\) and thus \((s', h'_{1\to 2}) \models I'\ u \rightarrow u\).

In both cases we obtain \((s', h'_{1\to 2}) \models I'\ u \rightarrow u\);

\(B\) is \(\text{alloc}(u)\): let us assume \((s, h_{1\to 2}) \models I\ \text{alloc}(u)\) and let us prove that \((s', h'_{1\to 2}) \models I'\ \text{alloc}(u)\). We get two cases:

- \(l \in \text{dom}(h)\). We derive \((s, h) \models I\ \text{alloc}(u)\) then \((s', h') \models I'\ \text{alloc}(u)\) and hence \((s', h'_{1\to 2}) \models I'\ \text{alloc}(u)\);
- \(l = l_1\). \(l_1/\ell_1\) verify (T1) in both (C1) and (C2), hence we get \(l_1' = l_1\). We deduce \((s', h'_{1\to 2}) \models I'\ \text{alloc}(u)\).

In both cases we obtain \((s', h'_{1\to 2}) \models I'\ \text{alloc}(u)\);

\(B\) is \(x_i \rightarrow u\): let us suppose \((s, h_{1\to 2}) \models I\ x_i \rightarrow u\) and let us prove that \((s', h'_{1\to 2}) \models I'\ x_i \rightarrow u\). We get two cases:

- \(h(\text{post}(s(x_i))) = l\). We derive \((s, h) \models I\ x_i \rightarrow u\) then \((s', h') \models I'\ x_i \rightarrow u\) and hence \((s', h'_{1\to 2}) \models I'\ x_i \rightarrow u\);
- \( l_1 = s(x_i) \) and \( l_2 = l \). In case (C1), \( l_1/l'_1 \) verify (T2) and \( l_2/l'_2 \) verify (T1), hence we get \( l_1' = s'(x_i) \) and \( l_2' = l' \) and thus \( (s', h'_{1\rightarrow 2}) \models_{P} x_i \mapsto u \). In case (C2), \( l_1 = s(x_i) \) contradicts \( l_1 \neq s(V) \).

In both cases we obtain \( (s', h'_{1\rightarrow 2}) \models_{P} x_i \mapsto u \).

\( B \) is \( u \mapsto x_j \): let us suppose \( (s, h_{1\rightarrow 2}) \models_{P} u \mapsto x_j \) and let us show \( (s', h'_{1\rightarrow 2}) \models_{P} u \mapsto x_j \). We get two cases.

- \( h(l) = s(x_j) \). We derive \( (s, h) \models_{P} u \mapsto x_j \) then \( (s', h') \models_{P} u \mapsto x_j \) and hence \( (s', h'_{1\rightarrow 2}) \models_{P} u \mapsto x_j \);
- \( l_1 = l \) and \( l_2 = s(x_i) \). \( l_1/l'_1 \) verify (T1) and \( l_2/l'_2 \) verify (T2) in both (C1) and (C2), hence we get \( l_1' = l' \) and \( l_2' = s'(x_i) \). We deduce \( (s', h'_{1\rightarrow 2}) \models_{P} u \mapsto x_j \).

In both cases we obtain \( (s', h'_{1\rightarrow 2}) \models_{P} u \mapsto x_j \).

This ends the proof of the basic equivalence \( (s, h_{1\rightarrow 2}, l) \simeq_{\gamma} (s', h'_{1\rightarrow 2}, l') \).

Let us consider \( \beta \)-Equipotence (B.2). By Proposition 2.11, there are 3 possible values for \( \text{pred}_{\beta}(s, h_{1\rightarrow 2}) \):

- if \( l_1 \notin s(V) \) and \( l_2 = s(x_i) \) then the identity
  \[
  \text{pred}_{\beta}(s, h_{1\rightarrow 2}, i) = \text{pred}_{\beta}(s, h, i) \wedge \{l_1\}
  \]
  holds. We can treat the case of (C1) and (C2) simultaneously.

  As \( l_1/l'_1 \) verify (T2–3), \( l_1/l'_1 \) also verify (T12) and we deduce \( l_1' \notin s(V) \). As \( l_2/l'_2 \) verify (T2), we get \( l_2' = s'(x_i) \).

  Thus by Proposition 2.11 again, we have
  \[
  \text{pred}_{\beta}(s', h'_{1\rightarrow 2}, i) = \text{pred}_{\beta}(s', h', i) \wedge \{l_1'\}
  \]

  From \( (s, h, l) \simeq_{\alpha} (s', h', l) \), we have \( \text{pred}_{\beta}(s, h, i) \simeq_{\alpha} \text{pred}_{\beta}(s', h', i) \) by Proposition 2.11 and we derive Equipotence (B.2) using Lemma 2.19 and \( \beta \leq \alpha + 1 \).

- if \( l_1 \in s(V) \) and \( l_2 \in \text{pred}_{\beta}(s, h, i) \) then the identity
  \[
  \text{pred}_{\beta}(s, h_{1\rightarrow 2}, i) = \text{pred}_{\beta}(s, h, i) \setminus \{l_2\}
  \]
  holds. We treat the case (C1) and (C2) separately.

  On the one hand, if (C1) holds then, \( l_1/l'_1 \) verify (T2) and thus also (T10). Thus we get \( l_1' \in s'(V) \). Moreover, \( l_2/l'_2 \) verify (T1–6) and thus also (T18). Thus we get \( l_2' \in \text{pred}_{\beta}(s', h', i) \). We deduce
  \[
  \text{pred}_{\beta}(s', h'_{1\rightarrow 2}, i) = \text{pred}_{\beta}(s', h', i) \setminus \{l_2'\}
  \]
  by Proposition 2.11. Since \( l_1 \in s(V) \), we have \( \beta + 1 = \alpha \). Thus by Proposition 2.20, we obtain Equipotence (B.2); On the other hand, (C2) contradicts \( l_1 \in s(V) \);

- in the **otherwise** case we have
  \[
  (l_1 \in s(V) \text{ or } l_2 \neq s(x_i)) \text{ and } (l_1 \notin s(V) \text{ or } l_2 \notin \text{pred}_{\beta}(s, h, i))
  \]
  and \( \text{pred}_{\beta}(s, h_{1\rightarrow 2}, i) = \text{pred}_{\beta}(s, h, i) \). By a combination of the arguments of two previous cases, in both (C1) and (C2), we have
  \[
  (l_1' \notin s'(V) \text{ or } l_2' \neq s'(x_i)) \text{ and } (l_1' \notin s'(V) \text{ or } l_2' \notin \text{pred}_{\beta}(s', h', i))
  \]
  and thus we get \( \text{pred}_{\beta}(s', h'_{1\rightarrow 2}, i) = \text{pred}_{\beta}(s', h', i) \) by Proposition 2.11. Equipotence (B.2) is immediate.

Let us consider \( \beta \)-Equipotence (B.3). We have \( \text{loop}_{\beta}(s, h) \simeq_{\alpha} \text{loop}_{\beta}(s', h') \). By Proposition 2.11, there are three cases for the value of \( \text{loop}_{\beta}(s, h_{1\rightarrow 2}) \):
Separation Logic with One Quantified Variable 

if \( l_1 \not\in p\circ\forall(s,h) \) and \( l_1 = l_2 \) then \( \text{loop}_{\mathcal{P}}(s,h_{1\rightarrow2}) = \text{loop}_{\mathcal{P}}(s,h) \cup \{l_1\} \). As \( l_1/l'_1 \) verify (T12), and \( l_2 = l_1 \) iff \( \alpha' = \alpha' \), we deduce \( l'_1 \not\in p\circ\forall(s',h')' \) and \( l'_1 = l'_2 \). Thus by Proposition 2.11, we have \( \text{loop}_{\mathcal{P}}(s',h'_{1\rightarrow2}) = \text{loop}_{\mathcal{P}}(s',h') \cup \{l'_1\} \). Since we have \( \text{loop}_{\mathcal{P}}(s,h) \sim_{\alpha} \text{loop}_{\mathcal{P}}(s',h') \), we deduce Equipotence (B.3) using Lemma 2.19 and \( \beta \leq \alpha + 1 \);

if \( l_1 \in s(V) \) and \( l_2 \not\in \text{loop}_{\mathcal{P}}(s,h) \) then \( \text{loop}_{\mathcal{P}}(s,h_{1\rightarrow2}) = \text{loop}_{\mathcal{P}}(s,h) \cup \{l_2\} \).

On the one hand, if (C1) holds then \( l_1/l'_1 \) verify (T10) and \( l_2/l'_2 \) verify (T19). Hence we get \( l'_2 \not\in s(V) \) and \( l'_2 \in \text{loop}_{\mathcal{P}}(s',h') \). We deduce \( \text{loop}_{\mathcal{P}}(s',h'_{1\rightarrow2}) = \text{loop}_{\mathcal{P}}(s',h') \cup \{l'_2\} \). Since \( l_1 \not\in s(V) \), we have \( \beta + 1 = \alpha \) and thus by Proposition 2.20 we get Equipotence (B.3).

On the other hand, (C2) contradicts \( l_1 \not\in s(V) \);

in the otherwise case we have

\[
(l_1 \not\in p\circ\forall(s,h) \text{ or } l_1 \not\in l_2) \text{ and } (l_2 \not\in s(V) \text{ or } l_2 \not\in \text{loop}_{\mathcal{P}}(s,h))
\]

and \( \text{loop}_{\mathcal{P}}(s,h_{1\rightarrow2}) = \text{loop}_{\mathcal{P}}(s,h) \). By a combination of the arguments of two previous cases, in both (C1) and (C2), we have

\[
(l'_1 \not\in p\circ\forall(s',h') \text{ or } l'_1 \not\in l'_2) \text{ and } (l'_2 \not\in s'(V) \text{ or } l'_2 \not\in \text{loop}_{\mathcal{P}}(s',h'))
\]

thus \( \text{loop}_{\mathcal{P}}(s',h'_{1\rightarrow2}) = \text{loop}_{\mathcal{P}}(s',h') \). Equipotence (B.3) is immediate.

Let us consider \( \beta \)-Equipotence (B.4). We have \( \text{rem}_{\mathcal{P}}(s,h) \sim_{\alpha} \text{rem}_{\mathcal{P}}(s',h') \). By Proposition 2.11, there are three cases for the value of \( \text{rem}_{\mathcal{P}}(s,h_{1\rightarrow2}) \):

if \( l_1 \not\in p\circ\forall(s,h) \cup \{l_2\} \) and \( l_2 \not\in s(V) \) then \( \text{rem}_{\mathcal{P}}(s,h_{1\rightarrow2}) = \text{rem}_{\mathcal{P}}(s,h) \cup \{l_1\} \). As \( l_1/l'_1 \) verify (T12), \( l_2/l'_2 \) verify (T10), we deduce \( l'_1 \not\in \text{pred}(s',h') \). Since \( l_2 \not\in s'(V) \), we have \( \beta + 1 = \alpha \) and thus by Proposition 2.20 we get Equipotence (B.4).

On the other hand, (C2) contradicts \( l_1 \not\in s(V) \);

in the otherwise case we have

\[
(l_1 \not\in p\circ\forall(s,h) \cup \{l_2\} \text{ or } l_2 \in s(V)) \text{ and } (l_1 \not\in s(V) \text{ or } l_2 \not\in \text{rem}_{\mathcal{P}}(s,h))
\]

and \( \text{rem}_{\mathcal{P}}(s,h_{1\rightarrow2}) = \text{rem}_{\mathcal{P}}(s,h) \). By a combination of the arguments of two previous cases, in both (C1) and (C2), we have

\[
(l'_1 \not\in p\circ\forall(s',h') \cup \{l'_2\} \text{ or } l'_2 \not\in s'(V)) \text{ and } (l'_1 \not\in s'(V) \text{ or } l'_2 \not\in \text{rem}_{\mathcal{P}}(s',h'))
\]

thus \( \text{rem}_{\mathcal{P}}(s',h'_{1\rightarrow2}) = \text{rem}_{\mathcal{P}}(s',h') \). Equipotence (B.4) is immediate.

\[
\square
\]

Proposition 3.26 Let \( m = (s,h,l) \) be a pointed memory state and \( l_1,l_2 \in \mathbb{N} \) be such that \( l_1 \not\in \text{dom}(h) \cup p\circ\forall(m) \). We have \( (s,h) \cup [l_1 \rightarrow l_2],l) \approx_{h} (s,h,l) \). Moreover, given \( \alpha \geq 0 \), if we assume that one of the following conditions hold

(C1) \( l_2 = s(x_{i}) \) and \( \text{card}(\text{pred}(s,h,i)) \geq \alpha \) for some \( i \in [1,q] \);
(C2) \( l_2 = l_1 \) and \( \text{card}(\text{loop}_{\mathcal{P}}(s,h)) \geq \alpha \);
(C3) \( l_2 \not\in s(V) \cup \{l_1\} \) and \( \text{card}(\text{rem}_{\mathcal{P}}(s,h)) \geq \alpha \).

then we have \( (s,h) \cup [l_1 \rightarrow l_2],l) \approx_{h} (s,h,l) \).
Proof We write \( h_{1 \rightarrow 2} \) to denote \( h \upharpoonright \{l_1 \mapsto l_2\} \). First, without assuming any of (C1–3), let us prove that \( (s, h_{1 \rightarrow 2}, I) \models h \). By Proposition 3.2 (monotonicity), we only need to prove that \( (s, h_{1 \rightarrow 2}, I) \models \mathcal{B} \) implies \( (s, h) \models \mathcal{B} \) for any formula \( \mathcal{B} \in \text{Basic}^* \). We proceed by a case analysis on \( \mathcal{B} \):

- \( \mathcal{B} \) is \( \alpha \) : This only depends on the value of \( s \) and therefore we are done;

- \( \mathcal{B} \) is \( \alpha \) : from \( (s, h_{1 \rightarrow 2}, I) \models \alpha \), we get \( h_{1 \rightarrow 2}(s(x_j)) = s(x_j) \). But since \( l_1 \not\in p^G(m) \), we deduce \( l_1 \neq s(x_j) \) and thus \( h(s(x_j)) = s(x_j) \). We get \( (s, h) \models \alpha \);

- \( \mathcal{B} \) is \( \text{conv}(x_i, x_j) \) : from \( (s, h_{1 \rightarrow 2}, I) \models \text{conv}(x_i, x_j) \), the identity \( h_{1 \rightarrow 2}(s(x_i)) = h_{1 \rightarrow 2}(s(x_j)) \) holds. But since \( l_1 \not\in \{s(x_i), s(x_j)\} \), we deduce \( h(s(x_i)) = h(s(x_j)) \) and thus \( (s, h) \models \text{conv}(x_i, x_j) \);

- \( \mathcal{B} \) is \( \text{btm}(x_i, x_j) \) : from \( (s, h_{1 \rightarrow 2}, I) \models \text{btm}(x_i, x_j) \), we get \( h_{1 \rightarrow 2}(h_{1 \rightarrow 2}(s(x_i))) = s(x_j) \). But since \( l_1 \not\in \{s(x_i), h(s(x_i))\} \) (remember \( h(s(x_i)) \in p^G(m) \)), we get \( h(h(s(x_i))) = s(x_j) \) and thus \( (s, h) \models \text{btm}(x_i, x_j) \);

- \( \mathcal{B} \) is \( \text{alloc}(x_i) \) : from \( (s, h_{1 \rightarrow 2}, I) \models \text{alloc}(x_i) \), we deduce \( h_{1 \rightarrow 2}(s(x_i)) \in \text{dom}(h_{1 \rightarrow 2}) \). Since \( l_1 \not\in \{s(x_i)\} \), we get \( h(s(x_i)) \in \text{dom}(h) \setminus \{l_1\} \). Since \( h(s(x_i)) \not\in l_1 \), we deduce \( h(s(x_i)) \in \text{dom}(h) \) and thus \( (s, h) \models \text{alloc}(x_i) \);

- \( \mathcal{B} \) is \( \text{toalloc}(x_i) \) : from \( (s, h_{1 \rightarrow 2}, I) \models \text{toalloc}(x_i) \), we get \( h_{1 \rightarrow 2}(h_{1 \rightarrow 2}(s(x_i))) = h_{1 \rightarrow 2}(s(x_i)) \). But since \( l_1 \not\in \{s(x_i), h(s(x_i))\} \), we get \( h(h(s(x_i))) = h(s(x_i)) \) and thus \( (s, h) \models \text{toalloc}(x_i) \);

- \( \mathcal{B} \) is \( u \in u \Rightarrow u \) : from \( (s, h_{1 \rightarrow 2}, I) \models u \Rightarrow u \), we get \( h_{1 \rightarrow 2}(h_{1 \rightarrow 2}(s(x_i))) = l_1 \). But since \( l_1 \not\in \{s(x_i)\} \), we deduce \( h(s(x_i)) = l_1 \) and thus \( (s, h) \models u \Rightarrow u \);

- \( \mathcal{B} \) is \( \text{alloc}(u) \) : from \( (s, h_{1 \rightarrow 2}, I) \models \text{alloc}(u) \), we get \( l \in \text{dom}(h_{1 \rightarrow 2}) \). But since \( l_1 \not\in \{l\} \) and \( \text{dom}(h_{1 \rightarrow 2}) = \text{dom}(h) \cup \{l_1\} \), we deduce \( l \in \text{dom}(h) \) and thus \( (s, h) \models \text{alloc}(u) \);

- \( \mathcal{B} \) is \( \text{toalloc}(u) \) : from \( (s, h_{1 \rightarrow 2}, I) \models \text{toalloc}(u) \), we get \( h_{1 \rightarrow 2}(h_{1 \rightarrow 2}(s(x_i))) = l \). But since \( l_1 \not\in \{s(x_i)\} \), we deduce \( h(s(x_i)) = l \) and thus \( (s, h) \models \text{toalloc}(u) \);

- \( \mathcal{B} \) is \( u \in u \Rightarrow u \) : from \( (s, h_{1 \rightarrow 2}, I) \models u \Rightarrow u \), we get \( h_{1 \rightarrow 2}(h_{1 \rightarrow 2}(s(x_i))) = l_1 \). But since \( l_1 \not\in \{s(x_i)\} \), we deduce \( h(s(x_i)) = l_1 \) and thus \( (s, h) \models u \Rightarrow u \).

Now we assume \( \alpha \geq 0 \) such that one of either (C1), (C2) or (C3) holds. Since we already have \( (s, h_{1 \rightarrow 2}, l) \models h \), according to Proposition 3.10, we have to establish three \( \alpha \)-equivalence constraints:

\[
\text{pred}_{\alpha}(s, h_{1 \rightarrow 2}, j) \sim_{\alpha} \text{pred}_{\alpha}(s, h, j) \quad \text{for any } j \in [1, q]
\]

\[
\text{loop}_{\alpha}(s, h_{1 \rightarrow 2}) \sim_{\alpha} \text{loop}_{\alpha}(s, h)
\]

\[
\text{rem}_{\alpha}(s, h_{1 \rightarrow 2}) \sim_{\alpha} \text{rem}_{\alpha}(s, h)
\]

If (C1) holds then by Proposition 2.11, we have \( \text{pred}_{\alpha}(s, h_{1 \rightarrow 2}, j) = \text{pred}_{\alpha}(s, h, j) \cup \{l_1\} \) if \( s(x_i) = s(x_j) \), \( \text{pred}_{\alpha}(s, h_{1 \rightarrow 2}, j) = \text{pred}_{\alpha}(s, h, j) \) if \( s(x_i) \neq s(x_j) \). Then we already have \( \text{pred}_{\alpha}(s, h_{1 \rightarrow 2}, j) \sim_{\alpha} \text{pred}_{\alpha}(s, h, j) \) when \( s(x_i) \neq s(x_j) \), \( \text{loop}_{\alpha}(s, h_{1 \rightarrow 2}) \sim_{\alpha} \text{loop}_{\alpha}(s, h) \) and \( \text{rem}_{\alpha}(s, h_{1 \rightarrow 2}) \sim_{\alpha} \text{rem}_{\alpha}(s, h) \). If \( s(x_i) = s(x_j) \) holds then we have \( \text{pred}_{\alpha}(s, h, j) = \text{pred}_{\alpha}(s, h, i) \). As a consequence, we get \( \text{card}(\text{pred}_{\alpha}(s, h_{1 \rightarrow 2}, j)) \geq \alpha + 1 \) and \( \text{card}(\text{pred}_{\alpha}(s, h, j)) \geq \alpha \). Hence \( \text{pred}_{\alpha}(s, h_{1 \rightarrow 2}, j) \sim_{\alpha} \text{pred}_{\alpha}(s, h, j) \) holds as well.

If (C2) holds then by Proposition 2.11, we have \( \text{loop}_{\alpha}(s, h_{1 \rightarrow 2}) = \text{loop}_{\alpha}(s, h, j) \cup \{l_1\} \). Indeed, \( l_2 = s(l_2) \) implies \( l_1 = l_2 \in \{V\} \) which contradicts \( l_1 \not\in \text{dom}(h) \cup p^G(m) \). We also get \( \text{rem}_{\alpha}(s, h_{1 \rightarrow 2}) = \text{rem}_{\alpha}(s, h, j) \cup \{l_1\} \). The three \( \alpha \)-equivalence constraints follow.

If (C3) holds then by Proposition 2.11, we have \( \text{loop}_{\alpha}(s, h_{1 \rightarrow 2}) = \text{loop}_{\alpha}(s, h, j) \cup \{l_1\} \). \( \text{rem}_{\alpha}(s, h_{1 \rightarrow 2}) = \text{rem}_{\alpha}(s, h) \). The \( \alpha \)-equivalence constraints follow. 

\[ \square \]

Corollary 3.27 Let \( \alpha \geq 0 \). Let \( m = (s, h, l) \) be a pointed memory state and \( h' \) be a heap such that \( \text{dom}(h') \cap (\text{dom}(h) \cup p^G(m)) = \emptyset \). If for any \( u \in \text{dom}(h') \) one of the following conditions holds
(C1) $h'(u) = s(x_i)$ and $\text{card}(\text{pred}_q(s(h, i))) \geq \alpha$ for some $i \in [1, q]$;
(C2) $h'(u) = u$ and $\text{card}(\text{loop}_w(s(h))) \geq \alpha$;
(C3) $h'(u) \notin s(V) \cup \{u\}$ and $\text{card}(\text{rem}_w(s(h))) \geq \alpha$.

then we have $(s, h \uplus h', l) \succeq_\alpha (s, h, l)$.

Proof We prove the result by induction on (the size of the domain of) $h'$. If $h' = \emptyset$ then the result it trivial by reflexivity of $\succeq_\alpha$. Otherwise, we can write $h' = [l_1 \mapsto l_2] \cup h''$. From $\text{dom}(h') \cap \text{(dom}(h) \cup \text{p}^\sigma(s(h, l))) = \emptyset$, we deduce $l_1 \notin \text{dom}(h')$ and $l_1 \notin s(V)$.

We then use the induction hypothesis on $h''$ (with $h = [l_1 \mapsto l_2]$ replacing $h$). Let us verify the requirements:

1. We have $\text{dom}(h'') \cap (\text{dom}(h \cup [l_1 \mapsto l_2]) \cup \text{p}^\sigma(s, h \cup [l_1 \mapsto l_2], l)) = \text{dom}(h'') \cap (\text{dom}(h) \cup \{l_1\} \cup \text{p}^\sigma(s, h, l)) = \emptyset$ because $l_1 \notin \text{dom}(h'')$ and $l_1 \notin s(V)$.

2. Let $u \in \text{dom}(h'')$. Let us show that either (C1), (C2) or (C3) holds for $u$. We have $u \notin \text{dom}(h'')$ and $h''(u) = h''(u)$. By hypothesis, one of the following conditions holds:

   - $h''(u) = s(x_i)$ and $\text{card}(\text{pred}_w(s(h, i))) \geq \alpha$ for some $i \in [1, q]$.
   - From $s(V) \subseteq \text{p}^\sigma(s, h, l)$, we deduce $l_1 \notin s(V)$. Thus by Proposition 2.11 we have $\text{pred}_q(s, h \cup [l_1 \mapsto l_2], i) \in \{\text{pred}_q(s, h, i), \text{pred}_q(s, h, i) \cup \{l_1\}\}$ and hence $\text{card}(\text{pred}_w(s, h \cup [l_1 \mapsto l_2], i)) \geq \alpha$. We also have $h''(u) = h''(u) = s(x_i)$ hence Condition (C1) holds.
   - $h''(u) = u$ and $\text{card}(\text{loop}_w(s(h))) \geq \alpha$. By Proposition 2.11 again, from $l_1 \notin s(V)$ we deduce $\text{card}(\text{loop}_w(s, h \cup [l_1 \mapsto l_2])) \geq \alpha$. As $h''(u) = h''(u) = u$, Condition (C2) holds.
   - $h''(u) \notin s(V) \cup \{u\}$ and $\text{card}(\text{rem}_w(s(h))) \geq \alpha$. By Proposition 2.11 again, from $l_1 \notin s(V)$ we deduce $\text{card}(\text{rem}_w(s, h \cup [l_1 \mapsto l_2])) \geq \alpha$. As $h''(u) = h''(u) \notin s(V) \cup \{u\}$, Condition (C3) holds.

As a consequence, we obtain $(s, h \uplus [l_1 \mapsto l_2] \cup h'', l) \succeq_\alpha (s, h \uplus [l_1 \mapsto l_2], l)$ by induction and thus $(s, h \uplus [l_1 \mapsto l_2] \cup h'' \cup u, l) \succeq_\alpha (s, h \uplus [l_1 \mapsto l_2], l)$ by transitivity of $\succeq_\alpha$. \qed

Proposition 3.29 Let $\alpha \geq 1$. We assume that the following conditions hold:

(a) $m \geq \alpha + 1$.
(b) $m_0 \geq \alpha + 1$.
(c) $\text{dom}(h) \subseteq \text{p}^\sigma(m)$.
(d) $\text{dom}(h') \subseteq \text{p}^\sigma(m')$.

Let $l_1 \in \alpha(s(V)) \cap \text{dom}(h_0 \cup h)$ and $l_2 \in \alpha$. There exist $l'_{1}, l'_{2} \in \alpha$ such that

1. $l'_{1} \in \alpha(s'(V)) \cap \text{dom}(h_0 \cup h')$;
2. $l'_{1}, l'_{2} \leq \text{maxval}(m'_0) + 1$;
3. $(s, h \uplus h' \uplus [l_1 \mapsto l_2], l) \succeq_\alpha (s', h' \uplus [l'_1 \mapsto l'_2], l')$;
4. $(s, h_0 \uplus h \uplus [l_1 \mapsto l_2], l) \succeq_\alpha (s', h'_0 \uplus h' \uplus [l'_1 \mapsto l'_2], l')$.

Proof According to Lemma 3.19, we have both $\alpha(l'_{1}, l'_{2})$ and $\alpha(l_{1}, l_{2})$.

Let use define $l'_{1}$ to be the unique value such that $l_1 \alpha(l'_{1})$. We also have $l_1 \alpha(l'_{2})$. Hence we get both $l_1 \alpha(l'_{1})$ and $l_2 \alpha(l'_{2})$. From $l_1 \in \alpha(s(V)) \cap \text{dom}(h'_0 \cup h')$ and $l_2 \in \alpha(s(V)) \cap \text{dom}(h'_0 \cup h')$. Since $l_1 \in \alpha(s(V))$ and $l_2 \notin \text{dom}(h_0 \cup h)$ from $l_1 \alpha(l'_{1})$ we deduce $l'_{1} \in \alpha(s'(V)) \cap \text{dom}(h'_0 \cup h')$ by (T10) and (T4). Hence Property 1 holds.

Let us establish $l'_{2}$. By Proposition 3.20: since $\alpha \geq 1$, we have $m_0 \geq 2$ and thus there exists $l'_{2} \leq \text{maxval}(m'_0) + 1$ such that $l_2 \alpha(l'_{2})$. Property 2 holds because $l'_{2} \in \alpha(s'(V)) \cap \text{dom}(h'_0 \cup h')$. Let us establish Property 4. i.e. $(s', h_0 \uplus h \uplus [l_1 \mapsto l_2], l) \succeq_\alpha (s', h'_0 \uplus h' \uplus [l'_1 \mapsto l'_2], l')$. We use Lemma 3.25 (C1) with $m_0 \alpha(m'_0)$. We have both $l_1 \notin \text{dom}(h_0 \cup h)$ and $l'_{1} \notin \text{dom}(h'_0 \cup h')$.

We check $\alpha(l'_{1} \mapsto l'_{2})$ verify (T1–3) because $l_1 \alpha(l'_{1})$ holds. $l_2/l'_2$ verify (T1–6) because $l_2 \alpha(l'_{2})$ holds. Let us check $l_1 = l'_1$ if $l'_{1} = l'_2$.?
We apply Lemma 3.25 (C2) with Hypothesis (a) and (b) and we get Property 3 and 4.

Proof

Let us show Property 3, i.e. \((s, h = [l_1 \rightarrow l_2], l) \approx_\alpha (s', h' = [l'_1 \rightarrow l'_2], l')\). We use Lemma 3.25 (C1) with \(m/m'\). Since \(l_1 \not\in \text{dom}(h_0 = h)\) and \(l'_2 \not\in \text{dom}(h'_0 = h')\) then \(l_1 \not\in \text{dom}(h)\) and \(l'_2 \not\in \text{dom}(h')\). \(l_1 / l'_1\) verify (T1–3) because \(l_1 \not\in l_2\) holds. We already verified that \(l_1 = l_2\) iff \(l'_1 = l'_2\) holds. Let us check that \(l_2/l'_2\) verify (T1–6), i.e. \(l_2 \not\in l_1\)

- if \(l_2 \not\in \text{pred}(m)\) then, as \(l_2 / l'_2\) holds, by Proposition 3.28, we get \(l_2 \not\in l'_2\) and thus \(l'_2 / l_2\);

- if \(l_2 \not\in \text{pre}(m)\) then we must have \(l'_2 \not\in \text{pre}(m')\); otherwise if \(l'_2 \in \text{pre}(m')\) then we would have \(l_2 \not\in l'_2\) by Proposition 3.28, which contradicts \(l_2 \not\in \text{pre}(m)\). Hence by Hypotheses (c) and (d) we deduce \(l_2 \not\in \text{dom}(h)\) and \(l'_2 \not\in \text{dom}(h')\). By Proposition 3.16 item 4, we deduce \(l_2 \not\in l'_1\).

We apply Lemma 3.25 (C1) with Hypothesis (a) and we get Property 3.

\(\Box\)

**Proposition 3.31**  Let \(\alpha \geq 1\). We assume that the following conditions hold:

(a) \(m \simeq_\alpha m'\);

(b) \(m_0 \simeq_\alpha m'_0\);

Let \(l_1 \not\in \text{dom}(h_0 = h) \cup s(V)\) and \(l_2 \in \mathbb{N}\). There exist \(l'_1, l'_2 \in \mathbb{N}\) such that

1. \(l'_1 \not\in \text{dom}(h'_0 = h') \cup s'(V)\);

2. \(l'_1, l'_2 \leq \text{maxval}(m'_0) + 2\);

3. \((s, h = [l_1 \rightarrow l_2], l) \simeq_\alpha (s', h' = [l'_1 \rightarrow l'_2], l')\);

4. \((s, h = [l_1 \rightarrow l_2], l) \simeq_\alpha (s', h' = h' = [l'_1 \rightarrow l'_2], l')\).

**Proof**  According to Lemma 3.19, we have both \(m_1 \in \mathcal{T}_1\) and \(m_0 \in \mathcal{T}_0\).

Let us define \(l'_1\) and simultaneously check Property 1 and prove that \(l'_1 \leq \text{maxval}(m_0) + 1\), \(l_1 \not\in l'_1\) hold:

- if \(l_1 \not\in \text{pred}(m_0)\) then let us define \(l'_1 \in \text{pred}(m_0)\) as the unique value such that \(l_1 / l'_1\). We immediately deduce \(l_1 \not\in l'_1\). As a consequence, \(l'_1 \not\in \text{dom}(h_0 = h)\) by (T4) and \(l'_1 \not\in s'(V)\) by (T10). Hence Property 1 holds. As \(l'_1 \in \text{pred}(m_0)\), the relation \(l'_1 \leq \text{maxval}(m_0) + 1\) holds trivially. Only \(l_1 \not\in l'_1\) remains. We use Proposition 3.28: if \(l_1 \not\in \text{pred}(m)\) or \(l'_1 \not\in \text{pre}(m)\) then \(l_1 / l'_1\), hence \(l_1 \not\in l'_1\); otherwise both \(l_1 \not\in \text{dom}(h) \cup \text{pred}(m)\) and \(l'_1 \not\in \text{dom}(h') \cup \text{pre}(m')\) hold and we deduce \(l_1 \not\in l'_1\) by Proposition 3.16 item 4;

- if \(l_1 \not\in \text{pre}(m_0)\) then we define \(l'_1 = \text{maxval}(m_0) + 1\) and Property 1 holds in an obvious way. We also have \(l_1 \not\in \text{dom}(h_0 = h) \cup \text{pred}(m_0)\) and \(l'_1 \not\in \text{dom}(h'_0 = h') \cup \text{pre}(m'_0)\) and we deduce \(l_1 \not\in l'_1\), since \(l_1 \not\in \text{dom}(h) \cup \text{pred}(m)\) and \(l'_1 \not\in \text{dom}(h') \cup \text{pre}(m')\) and we deduce \(l_1 \not\in l'_1\) by Proposition 3.16 item 4.

From \(l'_1 \leq \text{maxval}(m_0) + 1\), we obviously derive Property 2 for \(l'_1\).

Let us define \(l'_2\) by choosing the first possible choice in the following list. We simultaneously check Property 2 for \(l'_2\) and prove that \(l_2/l'_2\) verify (T2), and that \(l_2 = l_1\) iff \(l'_2 = l'_2\) holds:

- if \(l_2 = l_1\) then we define \(l'_2 = l'_2\). In case, Property 2 obviously holds for \(l'_2\) since it holds for \(l'_2\). Since \(l_1 \not\in l'_2\) holds then \(l_1 / l'_2\) verify (T2), and thus \(l_2/l'_2\) verify (T2). Since \(l_2 = l_1\) and \(l'_2 = l'_2\), the property \(l_2 = l_1\) iff \(l'_2 = l'_2\) holds;

- if \(l_2 \not\in s(V)\) then we define \(l'_2\) to be the unique location such that \(l_2 \not\in l'_2\). Then \(l'_2 \in \text{pred}(m') \leq \text{maxval}(m'_0)\) and as a consequence, Property 2 holds for \(l'_2\). From \(l'_2 \leq \text{maxval}(m'_0) + 1\) we deduce \(l_2 \not\in l'_2\) and as a consequence, \(l_2/l'_2\) verify (T2). We have \(l_2 \not\in s(V)\) hence we deduce \(l'_2 \not\in s'(V)\) using \(l_2 \not\in l'_2\) and (T10). We have \(l_2 \not\in s(V)\) hence we deduce \(l'_2 \not\in s'(V)\) (because \(l'_2 \geq \text{maxval}(\text{pred}(m'_0))\)). Hence \(l_2/l'_2\) verify (T2).

We apply Lemma 3.25 (C2) with Hypothesis (a) and (b) and we get Property 3 and 4.  \(\Box\)
C Proofs of Section 4

Corollary 4.12 1SL2 is strictly more expressive than 1SL1.

Proof Let $\mathcal{A}$ be the sentence in 1SL2 that states that there is a path of length 3 between $x_1$ and $x_2$ in the memory state and nothing else, for instance

$$\mathcal{A} \overset{\text{def}}{=} x_1 \neq x_2 \land C \land \neg(C \land \neg \text{emp})$$

with

$$C \overset{\text{def}}{=} \exists u_1, u_2 \left( u_1 \neq u_2 \land u_1 \neq x_1 \land u_1 \neq x_2 \land u_2 \neq x_1 \land u_2 \neq x_2 \land x_1 \mapsto u_1 \land u_1 \mapsto u_2 \land u_2 \mapsto x_2 \right)$$

Suppose that there is a sentence $\mathcal{A}'$ in 1SL1 whose models are precisely the memory states defined by $\mathcal{A}$. Let us show that this leads to a contradiction.

By Theorem 4.11, there is a Boolean combination $\mathcal{A}''$ of test formulae from $\text{Test}_{bmc}\{q, \mathcal{A}'\}$ for some $q \geq 1$ such that $\mathcal{A}'$ and $\mathcal{A}''$ are equivalent. Let $s$ be the store with $s(x_1) = 0$ and $s(x_2) = 3$. Let $h_1$ be the heap such that $h_1(0) = 1$, $h_1(1) = 2$ and $h_1(2) = 3$. Similarly, let $h_2$ be the heap such that $h_2(0) = 1$, $h_2(1) = 2$ and $h_2(4) = 3$. And let $i = 0$ for instance (any other value would fit). We note that $(s, h_1) \models A$ and therefore $(s, h_1) \models A''$ by assumption. Similarly, $(s, h_2) \not\models A$ and therefore $(s, h_2) \not\models A''$ by assumption. However, it is worth noting that for every test formula $B$ from $\bigcup_{i \geq 1} \text{Test}_i$, we have $(s, h_1) \models B$ iff $(s, h_2) \not\models B$, which leads to a contradiction because $A'$ is a Boolean combination of formulae from $\bigcup_{i \geq 1} \text{Test}_i$.

Lemma 4.15 Let $q \geq 1$ and $m \in \mathbb{N}$. Let $\mathcal{A}$ be an 1SL1 formula with program variables in $V = \{x_1, \ldots, x_q\}$ and $(s, h, l)$ be a pointed memory state. If we assume $\text{maxval}(s, h, l) + \varphi(\mathcal{A}) \leq m$ then

$$\text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \text{tt} \iff (s, h) \models A$$

Proof We proceed by induction on $\mathcal{A}$ and we prove the double implication, assuming that $\text{maxval}(s, h, l) + \varphi(\mathcal{A}) \leq m$ holds:

if $\mathcal{A}$ is atomic then $\text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \text{bmc}(q, m, V, \mathcal{A}, (s, h, l))$. The correctness of $\text{bmc}$ is obvious and left to the reader;

if $\mathcal{A}$ is $\neg A_1$ then

$$\text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \neg \text{bmc}(q, m, V, A_1, (s, h, l))$$

We deduce the equivalences $\text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \text{tt} \iff \text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) \neq \text{tt} \iff (s, h) \not\models A_1 \iff (s, h) \models A$ using the induction hypothesis;

if $\mathcal{A} = A_1 \lor A_2$ then

$$\text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \text{bmc}(q, m, V, A_1, (s, h, l)) \lor \text{bmc}(q, m, V, A_2, (s, h, l))$$

We deduce the equivalences $\text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \text{tt} \iff \text{bmc}(q, m, V, \mathcal{A}, (s, h, l)) = \text{tt} \land \text{bmc}(q, m, V, A_2, (s, h, l)) = \text{tt} \iff (s, h) \models A_1 \land (s, h) \models A_2 \iff (s, h) \models A$ using the induction hypotheses;

if $\mathcal{A}$ is $\exists u A_1$, let us assume $\text{bmc}(q, m, V, \exists u A_1, (s, h, l)) = \text{tt}$ and prove $(s, h) \models \exists u A_1$. By definition, there exists $l_0 \leq m$ such that $l_0 + \varphi(A_1) \leq m$ and $\text{bmc}(q, m, V, A_1, (s, h, l_0)) = \text{tt}$. We have $\text{maxval}(s, h) + \varphi(A_1) \leq \text{maxval}(s, h, l_0) + \varphi(A_1) \leq m$ by Proposition 4.13 item 1. Hence we deduce $\text{maxval}(s, h, l_0) + \varphi(A_1) \leq m$. By induction hypothesis we get $(s, h) \models l_0 A_1$ and thus $(s, h) \models \exists u A_1$.

Now let us assume $(s, h) \models \exists u A_1$ and prove $\text{bmc}(q, m, V, \exists u A_1, (s, h, l)) = \text{tt}$. By Corollary 4.5, there exists $l_0 \leq \text{maxval}(s, h) + 1$ such that $(s, h) \models l_0 A_1$. We have $\text{maxval}(s, h) + 1 + \varphi(A_1) \leq \text{maxval}(s, h, l_0) + \varphi(A_1) \leq m$ by Proposition 4.13 item 1. Hence we get both $l_0 + \varphi(A_1) \leq m$ and $\text{maxval}(s, h) + \varphi(A_1) \leq m$ and we deduce $\text{maxval}(s, h, l_0) + \varphi(A_1) \leq m$. By induction we derive $\text{bmc}(q, m, V, A_1, (s, h, l_0)) = \text{tt}$. By definition of $\text{bmc}$, we conclude $\text{bmc}(q, m, V, \exists u A_1, (s, h, l)) = \text{tt}$.
if $A$ is $A_1 * A_2$, let us assume $\text{bmc}(q, m, V, A_1 * A_2, (s, h, l)) = \text{tt}$ and let us prove $(s, h) \models_I A_1 * A_2$. By definition of bmc, there exists a heap $h_1 : [0, m] \rightarrow [0, m]$ such that $\text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) \leq m$ and $h_1 \subseteq h$. Then $\text{bmc}(q, m, V, A_1, (s, h_1, l)) = \text{tt}$ and $\text{bmc}(q, m, V, A_2, (s, h_2, l)) = \text{tt}$ with $h_2 = h - h_1$. For each $c \in \{1, 2\}$, we have $\text{maxval}(s, h_c, l) + \varphi(A_c) \leq \text{maxval}(s, h_c, l) + \varphi(A) \leq m$ by Proposition 4.13 item 4. Hence by induction hypotheses, we deduce $(s, h_1) \models_I A_1$ and $(s, h_2) \models_I A_2$. Given that the identity $h = h_1 \cup h_2$ holds, we get $(s, h) \models_I A_1 * A_2$.

Now let us assume $(s, h) \models_I A_1 * A_2$ and prove $\text{bmc}(q, m, V, A_1 * A_2, (s, h, l)) = \text{tt}$. There exists $h_1$ and $h_2$ such that $h = h_1 \cup h_2$, $(s, h_1) \models_I A_1$ and $(s, h_2) \models_I A_2$. For each $c \in \{1, 2\}$, from $h_1 \subseteq h$ we deduce $\text{maxval}(s, h_1, l) + \max(\varphi(A_1), \varphi(A_2)) \leq \text{maxval}(s, h, l) + \varphi(A) \leq m$ by Proposition 4.13 item 4. Hence we have the identities $\text{bmc}(q, m, V, A_1, (s, h_1, l)) = \text{tt}$ and $\text{bmc}(q, m, V, A_2, (s, h_2, l)) = \text{tt}$ by induction hypothesis. Moreover, we have $\text{subheap}(h_1, h) = \text{tt}$ and $\text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) \leq m$ holds. As $h_2 = h - h_1$, by definition of bmc, we get $\text{bmc}(q, m, V, A_1 * A_2, (s, h, l)) = \text{tt}$.

If $A$ is $A_1 \rightarrow A_2$, let us assume $\text{bmc}(q, m, V, A_1 \rightarrow A_2, (s, h, l)) = \text{tt}$ and prove $(s, h) \models_I A_1 \rightarrow A_2$. For this we use Corollary 4.6. Let us consider $h_1 \perp h$ such that $\text{maxval}(h_1) \leq \text{maxval}(s, h_1, l) + 15|A_1 \rightarrow A_2|^2$ such that $\text{maxval}(h_1) \leq m$ and $(s, h_1) \models_I A_1$ and $(s, h) \models_I A_2$. We have

$$\text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) \leq \text{maxval}(s, h_1, l) + 15|A_1 \rightarrow A_2|^2 + \max(\varphi(A_1), \varphi(A_2)) \leq \text{maxval}(s, h, l) + \varphi(A) \leq m$$

by Proposition 4.13 item 5. Let us prove the identity $\text{bmc}(q, m, V, A_1, (s, h_1, l)) = \text{tt}$. We have $\text{maxval}(s, h_1, l) + \varphi(A_1) \leq \text{maxval}(s, h_1, l) + \varphi(A) \leq m$ by Proposition 4.13 item 5. We also have $\text{maxval}(h_1) + \varphi(A_1) \leq \text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) \leq m$. Hence we get $\text{maxval}(s, h_1, l) + \varphi(A_1) \leq m$ and by induction hypothesis, from $(s, h_1) \models_I A_1$, we get $\text{bmc}(q, m, V, A_1, (s, h_1, l)) = \text{tt}$. Since $\text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) \leq m$ holds, by definition of bmc we have $h_1 \perp h$ and $\text{bmc}(q, m, V, A_1, (s, h_1, l)) = \text{tt}$. Hence, to satisfy $\text{bmc}(q, m, V, A_1 \rightarrow A_2, (s, h, l)) = \text{tt}$, we must have $\text{bmc}(q, m, V, A_2, (s, h \cup h_1, l)) = \text{tt}$. Then $\text{maxval}(h_1, l) + \varphi(A_2) \leq \text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) \leq m$ holds and $\text{maxval}(s, h, l) + \varphi(A_2) \leq \text{maxval}(s, h, l) + \varphi(A) \leq m$ by Proposition 4.13 item 5. We deduce $\text{maxval}(s, h \cup h_1, l) + \varphi(A_2) \leq m$ and thus we have $(s, h \cup h_1) \models_I A_2$ by induction hypothesis.

Now let us assume $(s, h) \models_I A_1 \rightarrow A_2$ and prove $\text{bmc}(q, m, V, A_1 \rightarrow A_2, (s, h, l)) = \text{tt}$. By definition of bmc, we pick $h_1 : [0, m] \rightarrow [0, m]$ and we verify that either $\text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) > m$ or $h_1 \perp h$ does not hold or $\text{bmc}(q, m, V, A_1, (s, h_1, l)) = \text{tt}$ or $\text{bmc}(q, m, V, A_2, (s, h \cup h_1, l)) = \text{tt}$. So we assume $h_1 \perp h$ and $\text{maxval}(h_1) + \max(\varphi(A_1), \varphi(A_2)) \leq m$ and $\text{bmc}(q, m, V, A_1, (s, h_1, l)) = \text{tt}$ and we prove the identity $\text{bmc}(q, m, V, A_2, (s, h \cup h_1, l)) = \text{tt}$. We have $\text{maxval}(h_1) + \varphi(A_1) \leq m$ and

$$\text{maxval}(s, h, l) + \varphi(A_1) \leq \text{maxval}(s, h, l) + \varphi(A) \leq m$$

by Proposition 4.13 item 5. Hence we derive $\text{maxval}(s, h_1, l) + \varphi(A_1) \leq m$ thus by induction hypothesis we get $(s, h_1) \models_I A_1$. As $h_1 \perp h$, we deduce $(s, h \cup h_1) \models_I A_2$. We have $\text{maxval}(h_1) + \varphi(A_2) \leq m$ and $\text{maxval}(s, h, l) + \varphi(A) \leq m$ by Proposition 4.13 item 5. Hence we derive $\text{bmc}(q, m, V, A_2, (s, h \cup h_1, l)) = \text{tt}$ and thus by induction hypothesis we get $\text{bmc}(q, m, V, A_2, (s, h \cup h_1, l)) = \text{tt}$.

\section*{D Proofs of Section 5}

**Proposition 5.3** (Completeness of the saturation rules) If the (finite) subset $P \subseteq \text{Basic}^*$ is closed under the rules of Figure 5.1 and $(s, H, L)$ is the canonical pre-model of $P$ then:

- $s$ is a total function $s : V \rightarrow [1, q]$, hence $s$ is a store;
- $H$ is a finite and functional graph, hence $H$ is the graph of some heap $h$;
L is a singleton subset of \( \mathbb{N} \), i.e. \( L = \{ l \} \) for a location \( l \);
- the inclusion \( \text{dom}(h) \subseteq \bigvee (s, h) \cup \{ l \} \) holds;
- for any formula \( B \in \text{Basic} \) we have \( (s, h) \models_B \iff B \in P \).

Proof Since \( P \) is closed under the three rules

\[
\begin{array}{ccc}
x = x & x = y & y = x \\
x = z & x = y & y = z
\end{array}
\]

the relation \( \{(x, y) \mid x = y \in P\} \) is an equivalence relation. Hence the function \( s \) is total: indeed \( x_i \in P \) and the set \( \{ j \mid x_i = x_j \in P \} \) contains at least \( i \). Hence \( s(x_i) \) is always defined and we have \( x_i = x_{s(x_i)} \in P \). Moreover we have

\[ s(x_i) = s(x_j) \] iff \( x_i = x_j \in P \) for all \( i, j \in [1, q] \) \hspace{1cm} (D.1)

Since \( P \) is closed under the two rules

\[
\begin{array}{ccc}
\text{conv}(x_i, x_j) & \text{conv}(x_i, x_j) & \text{conv}(x_i, x_k) \\
\text{conv}(x_j, x_i) & \text{conv}(x_j, x_i) & \text{conv}(x_j, x_k)
\end{array}
\]

the relation \( \{(i, j) \mid \text{conv}(x_i, x_j) \in P\} \) is a partial equivalence relation and \( h_i, h_j \) are both defined and \( h_i = h_j \) iff \( \text{conv}(x_i, x_j) \in P \) for all \( i, j \in [1, q] \) \hspace{1cm} (D.2)

It is obvious that \( H \) is a finite graph. An important remark for the rest of the proof is the following: by construction we have

\[ \{0\} = \{s(x_i) \mid i \in [1, q]\} \cup \{h_i \mid i \in [1, q] \text{ and } h_i \text{ is defined}\} = \{2q + 1\} \leq N \hspace{1cm} (D.3) \]

i.e. these sets are mutually disjoint. Let \( u, w \in N \) be such that \( (u, w) \in H \) and let us check the following characteristic properties of the graph \( H \):

P1 one of the three following properties holds:
- either \( u = s(x_i) \) for some \( i \in [1, q] \);
- or \( u = h_i \) for some \( i \) such that \( \text{conv}(x_i, x_i) \in P \);
- or \( u = 0 \);

P2 if \( u = s(x_i) \) then
- either \( w = s(x_j) \) and \( x_i \leftarrow x_j \in P \) for some \( j \in [1, q] \);
- or \( w = h_i \) and \( \text{conv}(x_j, x_i) \in P \) and \( \{ x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q \} \cap P = \emptyset \);

P3 if \( h_i \) is defined and \( u = h_i \) then \( \text{conv}(x_i, x_i) \in P \), \( \{ x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q \} \cap P = \emptyset \) and:
- either \( w = s(x_j) \) and \( \text{btm}(x_i, x_j) \in P \) for some \( j \in [1, q] \);
- or \( w = h_i \) and \( \text{toloop}(x_i) \in P \);
- or \( w = 0 \) and \( \text{toalloc}(x_i) \in P \) and
\[
\{ \text{btm}(x_i, x_1), \ldots, \text{btm}(x_i, x_q), \text{toloop}(x_i) \} \cap P = \emptyset .
\]

P4 if \( u = 0 \) then \( \{ x_1 = u, \ldots, x_i = u, x_i \leftarrow u, \ldots, x_q \leftarrow u \} \cap P = \emptyset \) and:
- either \( w = s(x_j) \) and \( u \leftarrow x_i \in P \) for some \( i \in [1, q] \);
- or \( w = 0 \) and \( u \leftarrow u \in P \) and \( \{ u \leftarrow x_1, \ldots, u \leftarrow x_q \} \cap P = \emptyset \);
- or \( w = 2q + 1 \) and \( \text{alloc}(u) \in P \) and \( \{ u \leftarrow x_1, \ldots, u \leftarrow x_q, u \leftarrow u \} \cap P = \emptyset \).

We prove Properties P1 to P4 in that order:

- Property P1 holds by definition of \( H \). We just have to check that when \( u = h_i \) then \( \text{conv}(x_i, x_i) \in P \) but this is a consequence of Equivalence (D.2);
- let us check Property P2. By definition of \( H \) and Property (D.3), there are two possibilities for \( (s(x_i), w) \in H \):
  - either \( (s(x_i), w) = (s(x_k), s(x_j)) \) with \( x_k \leftarrow x_j \in P \). But from \( s(x_k) = s(x_i) \) we deduce \( x_k = x_i \in P \) by Equivalence (D.1). As \( P \) is closed under the rule
\[
\begin{array}{c}
x_k = x_i \\
x_i \leftarrow x_j
\end{array}
\]
we deduce \( x_i \leftarrow x_j \in P \) and \( v = s(x_j) \);
– or \((s(x_j), w) = (s(x_k), h_j)\) with \(\text{conv}(x_k, x_j) \in P\) and \(\{x_k \leftarrow x_1, \ldots, x_k \leftarrow x_q\} \cap P = \emptyset\).

From \(s(x_j) = s(x_k)\) we deduce \(\{x_k = x_i, x_i = x_k\} \subseteq P\) by Equivalence (D.1). But \(P\) is closed under the rule

\[
x_k = x_i \quad \text{conv}(x_k, x_i) \quad \text{conv}(x_i, x_j) \quad \text{conv}(x_j, x_i) \quad \text{conv}(x_i, x_{i+1}) \quad x_i \leftarrow x_{i+1}
\]

hence we deduce \(\{\text{conv}(x_i, x_j), \text{conv}(x_i, x_{i+1})\} \subseteq P\) and \(\{x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q\} \cap P = \emptyset\).

We conclude \(w = h_j = h_i\) using Equivalence (D.2);

– let us check Property P3. By definition of \(H\) and Property (D.3), there are three possibilities for \((h_i, w) \in H\):

  – either \((h_i, w) = (h_k, s(x_j))\) with \(\text{btwn}(x_k, x_j) \in P\) and \(\{x_k \leftarrow x_1, \ldots, x_k \leftarrow x_q\} \cap P = \emptyset\).

  We deduce \(h_i = h_k\) and \(w = s(x_j)\). Thus \(\{\text{conv}(x_i, x_j), \text{conv}(x_i, x_{i+1}), \text{conv}(x_i, x_{i+2})\} \subseteq P\) by Equivalence (D.2). Since \(P\) is closed under the rules

\[
\begin{align*}
\text{conv}(x_i, x_j) & \quad \text{btwn}(x_i, x_j) & x_k \leftarrow x_j \\
\text{btwn}(x_i, x_j) & \quad \text{btwn}(x_i, x_j) & x_k \leftarrow x_j
\end{align*}
\]

we deduce \(\text{btwn}(x_i, x_j) \in P\) and \(\{x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q\} \cap P = \emptyset\);

  – \((h_i, w) = (h_k, h_j)\) with \(\text{toalloc}(x_k) \in P\) and \(\{x_k \leftarrow x_1, \ldots, x_k \leftarrow x_q, \text{btwn}(x_k, x_j)\} \cap P = \emptyset\).

  We deduce \(h_i = h_k\) and \(w = h_j\). By Equivalence (D.2), we deduce the inclusion \(\{\text{conv}(x_i, x_j), \text{conv}(x_i, x_{i+1}), \text{conv}(x_i, x_{i+2})\} \subseteq P\). Since \(P\) is closed under the rules

\[
\begin{align*}
\text{conv}(x_i, x_j) & \quad \text{toalloc}(x_k) & \text{toalloc}(x_k) \\
\text{toalloc}(x_i) & \quad \text{toalloc}(x_k) & \text{toalloc}(x_k)
\end{align*}
\]

we deduce \(\text{toalloc}(x_k) \in P\) and \(\{x_k \leftarrow x_1, \ldots, x_k \leftarrow x_q, \text{btwn}(x_k, x_j)\} \cap P = \emptyset\);

  – let us finally check Property P4. By definition of \(H\) and Property (D.3), there are three possibilities for \((0, w) \in H\):

    – \((0, w) = (0, s(x_j))\) with \(u \leftarrow x_j \in P\) and \(\{x_1 = u, \ldots, x_q = u, x_j \leftarrow u\} \cap P = \emptyset\).

    Hence \(w = s(x_j)\) and all the other properties hold;

    – \((0, w) = (0, u)\) with \(u \leftarrow u \in P\) and \(\{x_1 = u, \ldots, x_q = u, x_j \leftarrow u\} \cap P = \emptyset\).

    Hence \(w = 0\) and all the other properties hold;

    – \((0, w) = (0, 2q + 1)\) with \(\text{alloc}(u) \in P\) and \(\{x_1 = u, \ldots, x_q = u, x_j \leftarrow u\} \cap P = \emptyset\).

    Hence \(w = 2q + 1\) and all the other properties hold.

We can now check that \(H\) is a functional graph. Assume that \(\{(u, v), (u, w)\} \subseteq H\). Let us show \(v = w\). We have three cases:

– either \(u = s(x_j)\)

  – \(v = s(x_j)\) and \(w = s(x_k)\) with \(\{x_i \leftarrow x_j, x_i \leftarrow x_k\} \subseteq P\). But \(P\) is closed under the rule

\[
x_i \leftarrow x_j \quad x_i \leftarrow x_k \\
x_j = x_k
\]

hence \(x_j = x_k \in P\) and thus \(v = s(x_j) = s(x_k) = w\) by Equivalence (D.1);
\[ - v = s(x_i) \text{ and } w = h_i \text{ is impossible because } \{x_i \mapsto x_1, \ldots, x_i \mapsto x_q \} \cap P = \emptyset. \\
- v = h_j \text{ and } w = h_j, \text{ imply } v = w; \\
- v = h_j \text{ with } \text{conv}(x_i, x_j) \in P \text{ and } \{x_i \mapsto x_1, \ldots, x_i \mapsto x_q \} \cap P = \emptyset. \\
\]

Thus \(x_j = x_k \in P\) and thus \(v = s(x_j) = s(x_k) = w\) by Equivalence (D.1);

- \(v = s(x_j)\) and \(w = h_i\) with \(\text{tloop}(x_j, x_k) \subseteq P\). But \(P\) is closed under the rule

\[
\text{tloop}(x_i) \text{ tloop}(x_i, x_j) \\
x_i \mapsto x_j
\]

hence we deduce \(x_i \mapsto x_j\), which contradicts \(\{x_i \mapsto x_1, \ldots, x_i \mapsto x_q \} \cap P = \emptyset\);

- \(v = s(x_i)\) and \(w = 0\) is impossible because \(\text{btvm}(x_i, x_j) \in P\) and \(\{x_i \mapsto x_1, \ldots, x_i \mapsto x_q \} \cap P = \emptyset\);

- \(v = h_i \text{ and } w = h_i, \text{ implies } v = w; \\
- v = h_i \text{ and } w = 0 \text{ is impossible because } \text{tloop}(x_i) \in P \text{ and } \{x_i \mapsto x_1, \ldots, x_i \mapsto x_q \} \cap P = \emptyset; \\
- v = 0 \text{ and } w = 0 \text{ implies } v = w; \\
- \text{or } u = 0 \text{ with } \{x_i \mapsto u, \ldots, x_q \mapsto u\} \cap P = \emptyset.
\]

\[
- v = s(x_i) \text{ and } w = s(x_j) \text{ with } \{u \mapsto x_i, u \mapsto x_j \} \subseteq P. \text{ But } P \text{ is closed under the rule}
\]

\[
\text{tloop}(x_i) \text{ tloop}(x_i, x_j) \\
x_i \mapsto x_j
\]

hence \(x_i = x_j \in P\) and thus \(v = s(x_i) = s(x_j) = w\) by Equivalence (D.1);

- \(v = s(x_i)\) and \(w = 0\) with \(\{u \mapsto x_i, u \mapsto u\} \subseteq P\). But \(P\) is closed under the rule

\[
\text{tloop}(x_i) \text{ tloop}(x_i, x_j) \\
x_i \mapsto x_j
\]

hence \(x_i = x_j \in P\) which contradicts \(\{x_i \mapsto x_1, \ldots, x_i \mapsto x_q \} \cap P = \emptyset\);

- \(v = s(x_i)\) and \(w = 2q + 1\) is impossible because \(u \mapsto x_i \in P\) contradicts \(\{u \mapsto x_1, \ldots, u \mapsto x_q, u \mapsto u\} \cap P = \emptyset\)

- \(v = 0 \text{ and } w = 0 \text{ implies } v = w; \\
- v = 0 \text{ and } w = 2q + 1 \text{ is impossible because } u \mapsto u \in P \text{ contradicts } \{u \mapsto x_1, \ldots, u \mapsto x_q, u \mapsto u\} \cap P = \emptyset; \\
- v = 2q + 1 \text{ and } w = 2q + 1 \text{ implies } v = w.
\]

Let us now show that \(L\) is a singleton set. For that, we first show that \(L\) contains no more than one location:

- if \(s(x_i) \in L\) and \(s(x_j) \in L\) then \(\{x_i = u, x_j = u\} \subseteq P\). But \(P\) is closed under the rules

\[
\text{tloop}(x_i) \text{ tloop}(x_i, x_j) \\
x_i \mapsto x_j
\]

thus \(x_i = x_j \in P\) and \(s(x_i) = s(x_j)\);

- \(s(x_i) \in L\) and \(h_j \in L\) is impossible because \(x_i = u \in P\) contradicts \(\{x_1 = u, \ldots, x_q = u\} \cap P = \emptyset\);

- the case when \(s(x_i) \in L\) and \(0 \in L\) is impossible because \(x_i = u \in P\) contradicts \(\{x_1 = u, \ldots, x_q = u\} \cap P = \emptyset\);

- if \(h_i \in L\) and \(h_j \in L\) then we have \(\{x_i, u, x_j \mapsto u\} \subseteq P\). But \(P\) is closed under the rule

\[
\text{tloop}(x_i) \text{ tloop}(x_i, x_j) \\
x_j \mapsto u
\]

hence \(\text{conv}(x_i, x_j) \in P\) and thus \(h_i = h_j\);
Otherwise we have $t_L$ is impossible because $x_i \mapsto u \in P$ contradicts $\{x_1 \mapsto u, \ldots, x_q \mapsto u\} \cap P = \emptyset$.

Then we show that $L$ is not empty. If there exists $i$ such that $x_i = u \in P$ then $s(x_i) \in L$. Otherwise we have $\{x_1 = u, \ldots, x_q = u\} \cap P = \emptyset$. If there exists $j$ such that $x_j \mapsto u \in P$ then, because $P$ is closed under the rule

$$x_i \mapsto u \quad x_j \mapsto u \quad \frac{}{\text{conv}(x_j, x_j)}$$

we have $\text{conv}(x_j, x_j) \in P$ and thus $h_j$ is defined (see Equivalence (D.2)) and we deduce $h_j \in L$. Otherwise $\{x_1 = u, \ldots, x_q = u\} \cap P = \emptyset$. From the three previous properties we deduce $\text{dom}(h) \subseteq \eta(s, h) \cup \{l\}$ holds. For this we show that the three following properties hold:

- $\text{dom}(h) \subseteq \{s(x_i) \mid i \in [1, q]\} \cup \{h_i \mid i \in [1, q] \text{ and } h_i \text{ is defined}\} \cup \{0\}$;
- $\{s(x_i) \mid i \in [1, q]\} \cup \{h_i \mid i \in [1, q] \text{ and } h_i \text{ is defined}\} \subseteq \text{p}(\eta(s, h))$;
- if $0 \in \text{dom}(h)$ then $l = 0$.

The first property is trivial by definition of $H$. For the second property, we first notice that $\{s(x_i) \mid i \in [1, q]\} \subseteq \text{p}(\eta(s, h))$. Then if $h_i$ is defined then $\text{conv}(x_i, x_i) \in P$ and $\{x_i \mapsto x_1, \ldots, x_q \mapsto x_q\} \cap P = \emptyset$ holds. By characteristic Property P3 of $H$. Hence we have $s(x_i, h_i) \in H$ and we deduce $h_j = h(s(x_1, \ldots, x_q))$. Finally, if $0 \in \text{dom}(h)$ then by characteristic Property P4 of $H$ we have $\{x_1 = u, \ldots, x_q = u, x_1 \mapsto u, \ldots, x_q \mapsto u\} \cap P = \emptyset$ and as a consequence we get $0 \in L$ by definition of $L$. Hence $l = 0$.

From the three previous properties we deduce $\text{dom}(h) \subseteq \text{p}(\eta(s, h)) \cup \{l\}$ and hence the inclusion $\text{dom}(h) \subseteq \eta(s, h) \cup \{l\}$ holds.

Let us finally show that for any basic formula $B \in \text{Basic}^u$ we have $(s, h) \models I B$ iff $B \in P$. We proceed by case analysis on $B$:

- if $B$ is $x_i = x_j$. Then $(s, h) \models I x_i = x_j$ iff $s(x_i) = s(x_j)$ iff $x_i = x_j \in P$ by Equivalence (D.1);
- if $B$ is $x_i \mapsto x_j$. Let us first assume $(s, h) \models I x_i \mapsto x_j$ and show $x_i \mapsto x_j \in P$. We have $h(s(x_i)) = s(x_j)$ hence $\{s(x_i), s(x_j)\} \subseteq H$. By the characteristic Property P2 of $H$, the only possibility is that there exists $k$ such that $s(x_k) = s(x_i)$ and $x_k \mapsto x_j \in P$. Hence by Equivalence (D.1), we have $x_k = x_i \in P$. But $P$ is closed under the rule

$$x_k = x_i \quad x_i \mapsto x_k \quad \frac{x_i \mapsto x_j}{x_i \mapsto x_j}$$

hence we derive $x_i \mapsto x_j \in P$.

Let us now assume $x_i \mapsto x_j \in P$. Then $(s(x_i), s(x_j)) \in H$ by definition of $H$ and thus $(s, h) \models I x_i \mapsto x_j$;

- if $B$ is $\text{conv}(x_i, x_j)$. Let us first assume $(s, h) \models I \text{conv}(x_i, x_j)$ and show $\text{conv}(x_i, x_j) \in P$. We have $h(s(x_i)) = h(s(x_j)) = u$. Hence $\{s(x_i), u, s(x_j), u\} \subseteq H$. By the characteristic Property P2 of $H$ and Property (D.3), we have two cases:

- $v = s(x_k)$ and $v = s(x_l)$ with $\{x_i \mapsto x_k, x_j \mapsto x_l\} \subseteq P$. We deduce $s(x_k) = s(x_j)$ and thus $x_k = x_r \in P$ by Equivalence (D.1). But $P$ is closed under the rules

$$x_k = x_r \quad x_i \mapsto x_k \quad \frac{x_i \mapsto x_r}{x_i \mapsto x_r} \quad \text{conv}(x_i, x_j)$$

hence we get $\text{conv}(x_i, x_j) \in P$;

- $v = h_i$ and $v = h_j$ with $\{\text{conv}(x_i, x_i), \text{conv}(x_j, x_j)\} \subseteq P$. From $h_i = h_j$, we deduce $\text{conv}(x_i, x_i) \in P$ by Equivalence (D.2);

Now let us assume $\text{conv}(x_i, x_j) \in P$ and let us show $(s, h) \models I \text{conv}(x_i, x_j)$. We have two cases:
if \( x_i \leftarrow x_k \in P \) holds for some \( k \in [1, q] \) then as \( P \) is closed under the rule

\[
\frac{\text{conv}(x_i, x_j), x_i \leftarrow x_k}{x_j \leftarrow x_k}
\]

then \( x_j \leftarrow x_k \in P \) and \( \{(s(x_i), x_k), (s(x_j), x_k)\} \subseteq H \) by definition of \( H \). Hence \((s, h) \models_{\text{conv}} x_i, x_j\):
- otherwise \( \{x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q\} \cap P = \emptyset \). From \( \text{conv}(x_i, x_j) \in P \) we deduce \( h_i = h_j \) and \( \text{conv}(x_j, x_j) \in P \) by Equivalence (D.2). Hence by definition of \( H \) we have \( \{(s(x_i), h_i), (s(x_j), h_j)\} \subseteq H \) and we conclude \((s, h) \models_{\text{conv}} x_i, x_j\);

if \( B \) is \( \text{btwm}(x_i, x_j) \). Let us first assume \((s, h) \models_{\text{btwm}} x_i, x_j\) and show \( \text{btwm}(x_i, x_j) \in P \). We have \( \{(s(x_i), v), (v, s(x_j))\} \subseteq H \) for some \( v \). By characteristic Property P2 of \( H \), we have two cases:
- \( v = s(x_k) \) with \( x_i \leftarrow x_k \in P \). From \( (s(x_k), s(x_j)) \in H \) we deduce \((s, h) \models_{\text{btwm}} x_k, x_j\) and thus \( x_k \leftarrow x_j \in P \) (from the earlier case \( B = x_k \leftarrow x_j \)). Since \( P \) is closed under the rule

\[
\frac{x_i \leftarrow x_k, x_k \leftarrow x_j}{\text{btwm}(x_i, x_j)}
\]

we deduce \( \text{btwm}(x_i, x_j) \in P \);
- \( v = h_i \) with \( \text{conv}(x_i, x_j) \) and \( \{x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q\} \cap P = \emptyset \). By characteristic Property P3 of \( H \) and Property (D.3), there is only one possible case for \((h_i, s(x_j)) \in H \) such that \( s(x_i) = s(x_k) \) and \( \text{btwm}(x_i, x_k) \in P \). By Equivalence (D.1), we deduce \( x_k = x_j \in P \). Since \( P \) is closed under the rule

\[
\frac{x_k = x_j}{\text{btwm}(x_i, x_j)}
\]

we deduce \( \text{btwm}(x_i, x_j) \in P \);

Now let us assume \( \text{btwm}(x_i, x_j) \in P \) and let us show \((s, h) \models_{\text{btwm}} x_i, x_j\). We have two cases:
- either \( x_i \leftarrow x_k \in P \) holds for some \( k \in [1, q] \). As \( P \) is closed under the rule

\[
\frac{x_i \leftarrow x_k}{\text{btwm}(x_i, x_j)}
\]

we deduce \( x_k \leftarrow x_j \in P \) and thus we have \( \{(s(x_i), s(x_k)), (s(x_j), s(x_k))\} \subseteq H \) by definition of \( H \). As a consequence, we get \((s, h) \models_{\text{btwm}} x_i, x_j\);
- or \( \{x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q\} \cap P = \emptyset \). Since \( P \) is closed under the rule

\[
\frac{\text{btwm}(x_i, x_j)}{\text{conv}(x_i, x_j)}
\]

we deduce \( \text{conv}(x_i, x_j) \in P \) and thus \( \{(s(x_i), h_i), (h_i, s(x_j))\} \subseteq H \) by definition of \( H \). As a consequence we get \((s, h) \models_{\text{btwm}} x_i, x_j\);

if \( B \) is \( \text{toloop}(x_i) \). Let us first assume \((s, h) \models_{\text{toloop}} x_i\) and show \( \text{toloop}(x_i) \in P \). We have \( \{(s(x_i), v), (v, v)\} \subseteq H \) for some \( v \in N \). By characteristic Property P2 of \( H \), we have two cases for \((s(x_i), v) \in H \):
- \( v = s(x_j) \) with \( x_i \leftarrow x_j \in P \). From \( (s(x_j), s(x_j)) \in H \) we deduce \((s, h) \models_{\text{btwm}} x_j, x_j\) and thus \( x_j \leftarrow x_j \in P \) (from the earlier case \( B = x_j \leftarrow x_j \)). Since \( P \) is closed under the rule

\[
\frac{x_i \leftarrow x_j, x_j \leftarrow x_j}{\text{toloop}(x_i)}
\]

we deduce \( \text{toloop}(x_i) \in P \);
- \( v = h_i \) with \( \text{conv}(x_i, x_j) \in P \) and \( \{x_i \leftarrow x_1, \ldots, x_i \leftarrow x_q\} \cap P = \emptyset \). By characteristic Property P3 of \( H \) and Property (D.3), from \((h_i, h_j) \in H \) we deduce \( \text{toloop}(x_i) \in P \);
Now let us assume $\text{toloop}(x_i) \in P$ and let us show $(s, h) \models \text{toloop}(x_i)$. We have two cases:

- either $x_i \rightarrow x_j \in P$ holds for some $j \in [1, q]$. As $P$ is closed under the rule

\[
x_i \rightarrow x_j, \text{toloop}(x_i) \quad \quad \quad \quad x_j \rightarrow x_j
\]

we deduce $x_j \rightarrow x_j \in P$ and thus we have both $(s, h) \models x_i \rightarrow x_j$ and $(s, h) \models x_j \rightarrow x_j$ (from the earlier case $B = x_i \rightarrow x_j$ and $B = x_j \rightarrow x_j$). Hence we derive $(s, h) \models \text{toloop}(x_i)$;

- or $\{x_i \rightarrow x_1, \ldots, x_i \rightarrow x_q\} \cap P = \emptyset$. As $P$ is closed under the rule

\[
\text{toloop}(x_i), \text{talloc}(x_i) \quad \quad \quad \quad \text{talloc}(x_i), \text{conv}(x_i, x_j)
\]

we also get $\text{conv}(x_i, x_j) \in P$ and thus $\{(s(x_i), h_i), (h_i, h_i)\} \subseteq H$ by definition of $H$. Hence $(s, h) \models \text{toloop}(x_i)$.

if $B$ is $\text{talloc}(x_i)$. Let us first assume $(s, h) \models \text{talloc}(x_i)$ and show $\text{talloc}(x_i) \in P$.

We have $\{(s(x_i), v), (v, w)\} \subseteq H$ for some $v, w \in \mathbb{N}$. By characteristic Property P2 of $H$, we have two cases for $(s(x_i), v) \in H$:

- $v = s(x_j)$ with $x_i \rightarrow x_j \in P$. From $(s(x_j), v) \in H$ we deduce $(s, h) \models \text{conv}(x_j, x_j)$ and thus $\text{conv}(x_j, x_j) \in P$ (from the earlier case $B = \text{conv}(x_j, x_j)$). Since $P$ is closed under the rule

\[
x_i \rightarrow x_j, \text{conv}(x_j, x_j) \quad \quad \quad \quad x_j \rightarrow x_j
\]

we deduce $\text{talloc}(x_i) \in P$;

- $v = h_i$ with $\text{btwn}(x_i, x_j) \in P$. But $P$ is closed under the rule

\[
\text{btwn}(x_i, x_j), \text{talloc}(x_i)
\]

hence $\text{talloc}(x_i) \in P$;

- $w = h_i$ with $\text{toloop}(x_i) \in P$. But $P$ is closed under the rule

\[
\text{toloop}(x_i), \text{talloc}(x_i)
\]

hence $\text{talloc}(x_i) \in P$;

- $w = 0$ and in this case $\text{talloc}(x_i) \in P$.

Now let us assume $\text{talloc}(x_i) \in P$ and let us show $(s, h) \models \text{talloc}(x_i)$. We have four cases:

- either $x_i \rightarrow x_j \in P$ holds for some $j \in [1, q]$. As $P$ is closed under the rule

\[
x_i \rightarrow x_j, \text{talloc}(x_i) \quad \quad \quad \quad \text{conv}(x_j, x_j)
\]

we deduce $\text{conv}(x_j, x_j) \in P$ and thus we have both $(s, h) \models x_i \rightarrow x_j$ and $(s, h) \models \text{conv}(x_j, x_j)$ (from the earlier cases $B = x_i \rightarrow x_j$ and $B = \text{conv}(x_j, x_j)$). Hence we derive $(s, h) \models \text{talloc}(x_i)$;

- or $\{x_i \rightarrow x_1, \ldots, x_i \rightarrow x_q, \text{btwn}(x_i, x_1), \ldots, \text{btwn}(x_i, x_q)\} \cap P = \emptyset$ and $\text{toloop}(x_i) \in P$.

Then we have $(s, h) \models \text{toloop}(x_i)$ (from the earlier case $B = \text{toloop}(x_i)$). We deduce $(s, h) \models \text{talloc}(x_i)$;
- or \( \{x_i \mapsto x_1, \ldots, x_j \mapsto x_j, \text{btwn}(x_i, x_1), \ldots, \text{btwn}(x_j, x_q), \text{tooop}(x_i)\} \cap P = \emptyset \). Since 
\( P \) is closed under the rule 
\[
\text{toalloc}(x_i) \\
\text{conv}(x_i, x_i)
\]
we deduce \( \text{conv}(x_i, x_i) \in P \) and thus \( \{(s(x_i), b_1), (b_1, 0)\} \subseteq H \) hence we conclude 
\((s, h) \models \text{toalloc}(x_i) \);

if \( B \) is \( x_i \leftarrow u \). Let us first assume \((s, h) \models x_i \leftarrow u \) and show \( x_i \leftarrow u \in P \). We have \( l = s(x_i) \).

According to the definition of \( L \) and Property (D.3), we must have \( l = s(x_j) \) with \( x_j \leftarrow u \notin P \). But then we have \( s(x_i) = s(x_j) \) hence \( x_i = x_j \leftarrow u \in P \). As 
\( P \) is closed under the rule
\[
x_i = x_j, \quad x_j = u \\
x_i = u
\]
we get \( x_i \leftarrow u \in P \).

Conversely, if we assume \( x_i \leftarrow u \in P \) then by definition of \( L \) we have \( s(x_i) \in L \) and thus 
\( l = s(x_i) \). As a consequence, we have \((s, h) \models x_i = u \);

if \( B \) is \( x_i \leftarrow u \). Let us first assume \((s, h) \models x_i \leftarrow u \) and show \( x_i \leftarrow u \in P \). We have \((s(x_i), l) \in H \). By the characteristic Property P2 of \( H \), we have two cases:

- either \( l = s(x_j) \) with \( x_i \leftarrow x_j \in P \) for some \( j \in [1, q] \). We derive \((s, h) \models x_i \leftarrow x_j \) from the earlier case \( B = x_j \leftarrow x_i \) and thus we get \((s, h) \models x_j = u \). Hence we have 
\( x_j = u \in P \) (from the earlier case \( B = x_j = u \)). As \( P \) is closed under the rule
\[
x_j = u, \quad x_i \leftarrow x_j \\
x_i \leftarrow u
\]
we get \( x_i \leftarrow u \in P \);

- or \( l = b_1 \). But in that case, according to the definition of \( L \) and Property (D.3), we must have \( x_i \leftarrow u \in P \);

Now let us assume \( x_i \leftarrow u \in P \) and let us show \((s, h) \models x_i \leftarrow u \). We have two cases:

- either \( x_i \leftarrow u \in P \) for some \( j \in [1, q] \). As \( P \) is closed under the rules
\[
x_j = u, \quad x_i \leftarrow x_j \\
x_i \leftarrow u
\]
we get \( x_i \leftarrow x_j \in P \). Then we have \((s, h) \models x_i = u \) and \((s, h) \models x_i \leftarrow x_j \) (from the earlier cases \( B = x_j \leftarrow x_i \) and \( B = x_i \leftarrow x_j \)). Hence we deduce \((s, h) \models x_i \leftarrow u \);

- or \( x_i = u \). But \( P \) is closed under the rule
\[
x_i \leftarrow u
\]
and thus \( s(x_i) \in L \) and \( l = b_1 \) hence we conclude \((s, h) \models x_i \leftarrow u \);

if \( B \) is \( x_i \leftarrow x_i \). Let us first assume \((s, h) \models x_i \leftarrow x_i \) and show \( u \leftarrow x_i \in P \). According to the 
\( L \), for \( l \in L \) we have three cases:

- either \( l = s(x_j) \) with \( x_j = u \in P \) for some \( j \in [1, q] \). From \((s(x_j), s(x_i)) \in H \), using characteristic Property P2 of \( H \) and Property (D.3), we deduce \( x_j \leftarrow x_i \). But \( P \) is closed under the rule
\[
x_j = u, \quad x_i \leftarrow x_j \\
u \leftarrow x_i
\]
then by definition of \( \text{conv}(x_i, x_i) \) \( \in \emptyset \) and thus \((s, h) \models x_i \leftarrow u \);

- or \( l = b_j \) with \( x_j \leftarrow u \in P \) for some \( j \in [1, q] \). From \((b_j, s(x_i)) \in H \), using characteristic Property P3 of \( H \) and Property (D.3), we deduce \( \text{btwn}(x_j, x_i) \). Since \( P \) is closed under the rule
\[
x_j \leftarrow u, \quad \text{btwn}(x_j, x_i) \\
u \leftarrow x_i
\]
we get \( u \leftarrow x_i \in P \);
– or \( l = 0 \). From \((0, s(x_i)) \in H\), using characteristic Property P4 of \( H \) and Property (D.3), we deduce \( s(x_i) = s(x_j) \) and \( u \mapsto x_j \in P \). From Equivalence (D.1) we get \( x_j = x_i \in P \) and as \( P \) is closed under the rule

\[
\begin{align*}
x_j = x_i & \quad u \mapsto x_j \\
u \mapsto x_i
\end{align*}
\]

we conclude \( u \mapsto x_i \in P \).

Now let us assume \( u \mapsto x_i \in P \) and let us show \((s, h) \models u \mapsto x_i \). We have three cases for \( l \in L \):

– either \( l = s(x_j) \) with \( x_j = u \in P \) for some \( j \in [1, q] \). As \( P \) is closed under the rules

\[
\begin{align*}
x_j = u & \quad u \mapsto x_j \\
u \mapsto x_i
\end{align*}
\]

we get \( x_j \mapsto x_i \in P \) and thus \( (s, h) \models x_j \mapsto x_i \) from the earlier case \( B = x_j \mapsto x_i \).

We deduce \((s, h) \models u \mapsto x_i \);

– or \( l = b_j \) with \( x_j \mapsto u \in P \) and \( \{x_1 = u, \ldots, x_q = u\} \cap P = \emptyset \). As \( P \) is closed under the rules

\[
\begin{align*}
x_j \mapsto u & \quad u \mapsto x_i \\
\text{btwn}(x_j, x_i) & \quad x_j \mapsto x_p \quad x_j \mapsto u
\end{align*}
\]

we get \( \text{btwn}(x_j, x_i) \in P \) and \( \{x_j \mapsto x_1, \ldots, x_j \mapsto x_q\} \cap P = \emptyset \). Hence by definition of \( H \) we get \((b_j, s(x_1)) \in H\). We deduce \((s, h) \models u \mapsto x_i \);

– or \( l = 0 \) and \( \{x_1 = u, \ldots, x_q = u, x_1 \mapsto u, \ldots, x_q \mapsto u\} \cap P = \emptyset \). Then \((0, s(x_i)) \in H\) by definition of \( H \) and we deduce \((s, h) \models u \mapsto x_i \);

if \( B \) is \( u \mapsto u \). Let us first assume \((s, h) \models u \mapsto u \) and show \( u \mapsto u \in P \). According to the definition of \( L \), for \( l \in L \) we have three cases:

– either \( l = s(x_i) \) with \( x_i = u \in P \) for some \( i \in [1, q] \). From the earlier case \( B = x_i \mapsto x_i \). We deduce \((s, h) \models u \mapsto x_i \) and as a consequence \( x_i \mapsto x_i \in P \). But \( P \) is closed under the rules

\[
\begin{align*}
x_i = u & \quad u \mapsto x_i \\
u \mapsto x_i
\end{align*}
\]

hence \( u \mapsto u \in P \);

– or \( l = b_i \) with \( x_i \mapsto u \in P \) and \( \{x_i = u, \ldots, x_q = u\} \cap P = \emptyset \) for some \( i \in [1, q] \).

We deduce \((s, h) \models x_i \mapsto u \) from the earlier case \( B = x_i \mapsto u \). Hence we derive \((s, h) \models \text{toloop}(x_i) \) and thus \( \text{toloop}(x_i) \in P \) from the earlier case \( B = \text{toloop}(x_i) \).

But \( P \) is closed under the rule

\[
\begin{align*}
x_i \mapsto u & \quad \text{toloop}(x_i) \\
u \mapsto u
\end{align*}
\]

hence \( u \mapsto u \in P \);

– or \( l = 0 \). Then \((0, 0) \in H \) and by characteristic Property P4 of \( H \) and Property (D.3), we must have \( u \mapsto u \in P \).

Now let us assume \( u \mapsto u \in P \) and let us show \((s, h) \models u \mapsto u \). We have three cases for \( l \in L \) we have three cases:

– either \( l = s(x_i) \) with \( x_i = u \in P \) for some \( i \in [1, q] \). As \( P \) is closed under the rules

\[
\begin{align*}
x_i = u & \quad u = x_i \quad u \mapsto x_i \\
u \mapsto x_i
\end{align*}
\]

we get \( x_i \mapsto x_i \in P \) hence \((s, h) \models x_i \mapsto x_i \) (from the earlier case \( B = x_i \mapsto x_i \)). Since \( l = s(x_i) \) we deduce \((s, h) \models u \mapsto u \);
− or \( l = h_i \) with \( x_i \leftarrow u \in P \). As \( P \) is closed under the rule
\[
\text{alloc} (u) \\
\xrightarrow{\text{tloop}(x_i)} \\
x_i \leftarrow u, u \leftarrow u
\]
we get \( \text{tloop}(x_i) \in P \). From the earlier cases \( B = x_i \leftarrow u \) and \( B = \text{tloop}(x_i) \) we
deduce \((s, h) \models \text{id}, x_i \leftarrow u \) and \((s, h) \models \text{id}, \text{tloop}(x_i) \) hence \((s, h) \models \text{id}, u \leftarrow u \);
− or \( l = 0 \) and \( \{x_1 = u, \ldots , x_q = u, x_1 \leftarrow u, \ldots , x_q \leftarrow u \} \cap P = \emptyset \). Then \((0,0) \in H \) by
the definition of \( H \) and we deduce \((s, h) \models \text{id}, u \leftarrow u \);

if \( B \) is \( \text{alloc}(u) \). Let us first assume \((s, h) \models \text{id}, \text{alloc}(u) \) and show \( \text{alloc}(u) \in P \). According
to the definition of \( L \), for \( l \in L \) we have three cases:
− either \( l = s(x_i) \) with \( x_i = u \in P \) for some \( i \in [1,q] \). We deduce \( s(x_i) \in \text{dom}(h) \) and thus
\((s, h) \models \text{id}, \text{conv}(x_i, x_i) \). Using the earlier case \( B = \text{conv}(x_i, x_i) \), we get \( \text{conv}(x_i, x_i) \in P \).
But \( P \) is closed under the rule
\[
x_i = u \xrightarrow{\text{conv}(x_i, x_i)} \text{alloc}(u)
\]

− or \( l = h_i \) with \( x_i = u \in P \) for some \( i \in [1,q] \). Using the earlier case \( B = x_i \leftarrow u \), we
deduce \((s, h) \models \text{id}, x_i \leftarrow u \) and then \((s, h) \models \text{id}, \text{tloop}(x_i) \). Hence we get \( \text{tloop}(x_i) \in P \)
(from the earlier case \( B = \text{tloop}(x_i) \)). As \( P \) is closed under the rule
\[
\text{tloop}(x_i) \xrightarrow{\text{alloc}(u)} x_i \leftarrow u
\]

− or \( l = 0 \). Then \((0, v) \in H \) for some \( v \in N \). Using characteristic Property P4 we
deduce \( \{x_1 = u, \ldots , x_q = u, x_1 \leftarrow u, \ldots , x_q \leftarrow u \} \cap P = \emptyset \) and:
− either \( v = s(x_i) \) and \( u \leftarrow x_i \in P \) for some \( i \in [1,q] \). As \( P \) is closed under the rule
\[
\text{alloc}(u) \\
\xrightarrow{u \leftarrow x_i}
\]

− or \( v = 0 \) and \( u \leftarrow u \in P \) and \( \{u \leftarrow x_1, \ldots , u \leftarrow x_q \} \cap P = \emptyset \). As \( P \) is closed
under the rule
\[
\text{alloc}(u) \\
\xrightarrow{u \leftarrow u}
\]

Now let us assume \( \text{alloc}(u) \in P \) and let us show \((s, h) \models \text{id}, \text{alloc}(u) \). We have three cases
for \( l \in L \):
− either \( l = s(x_i) \) with \( x_i = u \in P \) for some \( i \in [1,q] \). As \( P \) is closed under the rules
\[
x_i = u \xrightarrow{u \leftarrow x_i} \text{alloc}(u)
\]

we get \( \text{conv}(x_i, x_i) \in P \) and thus \((s, h) \models \text{id}, \text{conv}(x_i, x_i) \) from the earlier case \( B = \text{conv}(x_i, x_i) \).
Hence \( l = s(x_i) \in \text{dom}(h) \) and we deduce \((s, h) \models \text{id}, \text{alloc}(u) \);
− or \( l = h_i \) with \( x_i = u \in P \) and \( \{x_1 = u, \ldots , x_q = u \} \cap P = \emptyset \). As \( P \) is closed under the rules
\[
x_i \leftarrow u \xrightarrow{\text{alloc}(u)} \text{tloop}(x_i)
\]

we get \( \text{tloop}(x_i) \in P \). We derive \((s, h) \models \text{id}, x_i \leftarrow u \) (from the earlier case \( B = x_i \leftarrow u \))
and \((s, h) \models \text{id}, \text{tloop}(x_i) \) (from the earlier case \( B = \text{tloop}(x_i) \)). Hence we get
\((s, h) \models \text{id}, \text{alloc}(u) \);
or \( l = 0 \) and \( \{ x_1 = u, \ldots, x_q = u, x_1 \mapsto u, \ldots, x_q \mapsto u \} \cap P = \emptyset \). We consider three cases:

- either \( u \mapsto x_i \in P \) holds for some \( i \in [1, q] \). In this case, \((0, s(x_i)) \in H\) by definition of \( H \) and we get \((s, h) \models_{\text{alloc}} \text{alloc}(u)\);
- or \( u \mapsto x_i, \ldots, x_q \mapsto u \in P \). In this case, \((0, 0) \in H\) by definition of \( H \) and we get \((s, h) \models_{\text{alloc}} \text{alloc}(u)\);
- or \( \{ u \mapsto x_1, \ldots, u \mapsto x_q, u \mapsto u \} \cap P = \emptyset \). In that case \((0, 2q + 1) \in H\) by definition of \( H \) and we get \((s, h) \models_{\text{alloc}} \text{alloc}(u)\).

\[\tag*{\Box}\]

**Proposition D.1.** Let \( q \geq 1 \). Let \( s \) be a store, \( h_1 \) and \( h_2 \) be two heaps and \( l \) be a location. We assume that \( \forall(s, h_1) \cup \{ l \} = \forall(s, h_2) \cup \{ l \} \) and that \( h_1 \) and \( h_2 \) are identical maps on that subset of locations. Then \((s, h_1, l) \models_{\text{alloc}} (s, h_2, l)\).

**Proof.** Let us denote \( D = \forall(s, h_1) \cup \{ l \} = \forall(s, h_2) \cup \{ l \} \). We show that \((s, h_1) \models_{\text{alloc}} (s, h_2) \models B\) by case analysis on \( B\):

- \( B = x_1 = x_2 \): if \((s, h_1) \models x_1 = x_2 \) then \( s(x_1) = s(x_2)\) and thus \((s, h_2) \models x_1 = x_2 \);
- \( B = x_1 \mapsto x_2 \): if \((s, h_1) \models x_1 \mapsto x_2 \) then \( h_1(s(x_1)) = s(x_2)\). Hence \( s(x_1) \in \text{ref}(s, h_1) \subseteq D \) and we deduce \( h_2(s(x_1)) = h_1(s(x_1)) = s(x_2)\). We conclude \((s, h_2) \models x_1 \mapsto x_2 \);
- \( B = \text{conv}(x_1, x_2) \): if \((s, h_1) \models \text{conv}(x_1, x_2) \) then \( h_1(s(x_1)) = h_1(s(x_2))\). Hence we have \( \{s(x_1), s(x_2)\} \subseteq D \) and we deduce \( h_2(s(x_1)) = h_2(s(x_2)) = h_1(s(x_1)) = s(x_2)\). We conclude \((s, h_2) \models \text{conv}(x_1, x_2) \);
- (checking the remaining cases is similar to the first one)

We show that \((s, h_1) \models_{\text{alloc}} (s, h_2) \models \text{alloc}(u)\) then \( l \in \text{dom}(h_1)\).

**Proposition 5.5.** Let \( q \geq 1 \). Let \( s : V \to \mathbb{N} \) be a store, \( h : \mathbb{N} \to \mathbb{N} \) be a heap and \( l \in \mathbb{N} \) be a location. Let \((p_1, \ldots, p_q, l, 1)\) be a cardinality assignment such that:

1. \( s(x_i) = s(x_j) \) implies \( p_i = p_j \) for all \( i, j \in [1, q] \);
2. \( \text{card}(\text{pred}(s, h, l)) \leq p_i \) for any \( i \in [1, q] \);
3. \( \text{card}(\text{loop}(s, h)) \leq l \);
4. \( \text{card}(\text{rem}(s, h)) \leq r \).

There exists a heap \( h' \) such that:

- \((s, h, l) \models_{\text{alloc}} (s, h', l)\);
- \( \text{card}(\text{pred}(s, h', i)) = p_i \) for any \( i \in [1, q] \);
- \( \text{card}(\text{loop}(s, h')) = l \);
- \( \text{card}(\text{rem}(s, h')) = r \).

\[\tag*{\Box}\]
Proof Let us define i = min{j ∈ [1, q] | s(x_i) = s(x_j)} for every i ∈ [1, q], m = maxval(s, h, l), n = \max\{p_1, \ldots, p_l, r\}, p'_i = p_i − card(pred_{>}(s, h, i)) for every i ∈ [1, q], l' = l − card(\text{loop}(s, h, i)) and r' = r − card(rem_{>}(s, h, i)). We define h' by the following rules:

- h'(u) = v when u ≤ m and h(u) = v;
- h'(u) = v when u = m + 2 + ln + d, v = s(x_i), i ∈ [1, q], 1 ≤ d ≤ l';
- h'(u) = v when u = m + 2 + (q + 1)n + d, v = u and 1 ≤ d ≤ l';
- h'(u) = v when u = m + 2 + (q + 2)n + d, v = m + 1 and 1 ≤ d ≤ r'.

Then it is easy to check that h' is a heap that satisfies the following properties:

- \text{succ}(s, h') ∪ \{l\} = \text{succ}(s, h) ∪ \{l\} and the restrictions of h and h' to \text{succ}(s, h) ∪ \{l\} are identical maps;
- \text{pred}_{>}(s, h') = \text{pred}_{>}(s, h, i) ∪ \{l\} = [m + 2 + ln + 1, m + 2 + ln + p'_i];
- \text{loop}_{>}(s, h') = \text{loop}_{>}(s, h) ∪ \{m + 2 + (q + 2)n + 1, m + 2 + (q + 2)n + l', m + 2 + (q + 2)n + r', m + 2 + (q + 2)n + p'_i\};
- \text{rem}_{>}(s, h') = \text{rem}_{>}(s, h, i) ∪ \{m + 2 + (q + 2)n + 1, m + 2 + (q + 2)n + l', m + 2 + (q + 2)n + r', m + 2 + (q + 2)n + p'_i\}.

Hence the cardinality identities card(\text{pred}_{>}(s, h', i)) = p_i for i ∈ [1, q], card(\text{loop}_{>}(s, h')) = l and card(\text{rem}_{>}(s, h')) = r are obvious. The basic equivalence (s, h, l) ∼_b (s', h', l') comes from Proposition D.1 (page 76 in Appendix D).

\textbf{Proposition 5.7} If the conjunction of the formulas in B^+ ∪ ¬B^− ∪ S is satisfiable then the triple (B^+ , B^− , S) is n-consistent for any n ∈ \{1, 2, 3\}.

Proof Let us first prove the result for 1-consistency. Let us fix a triple (B^+ , B^− , S) and consider a memory state (s, h) and a location l such all the formulas in B^+ ∪ ¬B^− ∪ S are satisfied in (s, h, l). Let us show that Conditions C1.1–4 hold:

C.1.1 No formula in B^− is satisfied in (s, h, l) and by Proposition 5.1, all the formulas of cl(B^+ ) are satisfied in (s, h, l). Hence we deduce B^− ∩ cl(B^+ ) = \emptyset;

C.1.2 If x_i = x_j ∈ cl(B^+ ) and \# \text{pred}_{>}(x_i) ≥ a, \# \text{pred}_{>}(x_j) ≥ b then we deduce s(x_i) = s(x_j), card(\text{pred}_{>}(s, h, i)) ≥ a and card(\text{pred}_{>}(s, h, j)) ≥ b. But s(x_i) = s(x_j) implies \text{pred}_{>}(s, h, i) = \text{pred}_{>}(s, h, j) hence a < b;

C.1.3 If \# \text{loop}_{>}(s, h) ≥ a, \# \text{loop}_{>}(s, h) ≥ b then the relations card(\text{loop}_{>}(s, h)) ≥ a and card(\text{loop}_{>}(s, h)) ≥ b hold hence a < b;

C.1.4 If \# \text{rem}_{>}(s, h) ≥ a, \# \text{rem}_{>}(s, h) ≥ b ≤ S then the relations card(\text{rem}_{>}(s, h)) ≥ a and card(\text{rem}_{>}(s, h)) ≤ S hold hence a < b.

Let us prove the result for 2-consistency. Let us fix a triple (B^+ , B^− , S). Let us consider a memory state (s, h) and a location l such all the formulas in B^+ ∪ ¬B^− ∪ S are satisfied in (s, h, l). We have already established that (B^+ , B^− , S) is 1-consistent in this case. Let us show that Conditions C2.1–2 hold:

C.2.1 If \{x_i = x_j, u → x_i, l\} ⊆ cl(B^+ ) and \# \text{pred}_{>}(x_i) ≥ 1 ∈ S then we deduce s(x_i) = s(x_j), h(l) = s(x_i) and card(\text{pred}_{>}(s, h, j)) < l. But then h(l) = s(x_j) hence l ∈ \text{pred}_{>}(s, h, j). From \text{pred}_{>}(s, h, j) = \emptyset we derive l ∈ \text{pred}_{>}(s, h, j) hence l ∈ \text{succ}(s, h) ⊆ s(V) ∪ h(s(V)).

Hence (s, h, l) satisfies at least one formula B ∈ Eq_{n} ∪ T_{Da}. We deduce that all the formulas of B^+ ∪ \{B\} ∪ ¬B^− ∪ S are satisfied in (s, h, l) hence (B^+ ∪ \{B\}, B^− , S) is 1-consistent;

C.2.2 If u → u ∈ cl(B^+ ) and \# \text{loop}_{>}(s, h) ≥ 1 ∈ S then we have h(l) = l and card(\text{loop}_{>}(s, h)) < 1. From \text{loop}_{>}(s, h) = \emptyset and l ∈ \text{loop}_{>}(s, h) we deduce l ∈ \text{succ}(s, h) ⊆ s(V) ∪ h(s(V)).

Hence (s, h, l) satisfies at least one formula B ∈ Eq_{n} ∪ T_{Da}. We deduce that all the formulas of B^+ ∪ \{B\} ∪ ¬B^− ∪ S are satisfied in (s, h, l) hence (B^+ ∪ \{B\}, B^− , S) is 1-consistent;

Let us prove the result for 3-consistency. Let us fix a triple (B^+ , B^− , S) and consider a memory state (s, h) and a location l such all the formulas in B^+ ∪ ¬B^− ∪ S are satisfied in (s, h, l). We have already established that (B^+ , B^− , S) is 2-consistent in this case. Let us show that Condition C3.1 holds: if alloc(u) ∈ cl(P) and \# \text{rem}_{>} ≥ 1 ∈ S then l ∈ dom(h) and rem_{>}(s, h) = \emptyset. Hence by Lemma 2.6, either l ∈ \text{succ}(s, h) or \text{pred}_{>}(s, h, i) for some i ∈ [1, q] or \text{loop}_{>}(s, h, i). As a consequence, (s, h, l) satisfies at least one formula of B ∈ Eq_{n} ∪ T_{Da} ∪ F_{ma} [u → u]. We deduce that all the formulas of B^+ ∪ \{B\} ∪ ¬B^− ∪ S are satisfied in (s, h, l) hence (B^+ ∪ \{B\}, B^− , S) is 2-consistent. □
Proposition 5.8 If the triple \((B^+, B^-, S)\) is 3-consistent then the conjunction of the formula in \(B^+ \cup \neg B^- \cup S\) is satisfiable.

Proof Let us first consider the case where \((B^+, B^-, S)\) is 1-consistent (which is the weakest of the assumptions of 1-, 2- or 3-consistency). We define a cardinality assignment \((p_1, \ldots, p_q, l, r)\) by:

\[
\begin{align*}
p_i &= \max\{a \mid \exists k \in [1, q], x_i = x_k \in \text{cl}(B^+) \land \#\text{pred}(x_i) \geq a \in S\} \quad \text{for } i \in [1, q] \\
l &= \max\{a \mid \#\text{loop} \geq a \in S\} \\
r &= \max\{a \mid \#\text{rem} \geq a \in S\}
\end{align*}
\]

where we assume \(\max(\mathcal{G}) = 0\). Since \((B^+, B^-, S)\) is 1-consistent, we check that the following properties hold for any \(a \in \mathbb{N}\) and all \(i, j \in [1, q]\):

(P0) if \(x_i = x_j \in \text{cl}(B^+)\) then \(p_i = p_j\);
(P1) if \(\#\text{pred}(x_i) \geq a \in S\) then \(p_i \geq a\);  
(P2) if \(\neg\#\text{pred}(x_i) \geq a \in S\) then \(p_i < a\);
(P3) if \(\#\text{loop} \geq a \in S\) then \(l \geq a\);  
(P4) if \(\neg\#\text{loop} \geq a \in S\) then \(l < a\);
(P5) if \(\#\text{rem} \geq a \in S\) then \(r \geq a\);  
(P6) if \(\neg\#\text{rem} \geq a \in S\) then \(r < a\).

Property (P0) let us assume \(x_i = x_j \subseteq \text{cl}(B^+)\) and let us show \(p_i \leq p_j\). Let \(a \in \mathbb{N}\) and \(k \in [1, q]\) be such that \(x_i = x_k \in \text{cl}(B^+)\) and \(\#\text{pred}(x_k) \geq a \in S\). Let us show \(a \leq p_j\).

Since \(\text{cl}(B^+)\) is closed under rules

\[
\begin{align*}
x_i &= x_j & x_j &= x_i \\
x_i &= x_k & x_k &= x_i \\
x_j &= x_k & x_k &= x_j
\end{align*}
\]

we deduce \(x_i = x_k \subseteq \text{cl}(B^+)\). Hence by definition of \(p_i\) (max.), we get \(a \leq p_j\). We conclude \(p_i \leq p_j\). The relation \(p_i \leq p_j\) is derived directly because \(x_j = x_k \subseteq \text{cl}(B^+)\) holds as well.

Property (P1) if \(\#\text{pred}(x_i) \geq a \in S\) then, as \(\text{cl}(B^+)\) is closed under rules \(x_i = x_k \subseteq \text{cl}(B^+)\) and thus \(a \leq p_i\) by definition of \(p_i\);

Property (P2) let us assume \(\neg\#\text{pred}(x_i) \geq a \in S\) and let us show \(p_i < a\). Hence, let \(b \in \mathbb{N}\) and \(k \in [1, q]\) be such that \(x_i = x_k \subseteq \text{cl}(B^+)\) and \(\#\text{pred}(x_k) \geq b \in S\) and let us show \(b < a\). From \(x_i = x_k \subseteq \text{cl}(B^+)\) we deduce \(x_k = x_i \subseteq \text{cl}(B^+)\). As we also have \(\#\text{pred}(x_k) \geq b, \neg\#\text{pred}(x_i) \geq a \in S\), by Property C1.2 (which holds for 1-consistency) we deduce \(b < a\). We conclude \(p_i < a\);

Property (P3) if \(\#\text{loop} \geq a \in S\) then by definition of \(l\) we have \(a \leq l\);

Property (P4) let us assume \(\neg\#\text{loop} \geq a \in S\) and let us show \(l < a\). Hence, let \(b \in \mathbb{N}\) be \(\#\text{loop} \geq b \in S\) and let us show \(b < a\). We have \(\#\text{loop} \geq b, \neg\#\text{loop} \geq a \subseteq S\) hence by Property C1.3 we deduce \(b < a\). We conclude \(l < a\);

Property (P5) if \(\#\text{rem} \geq a \in S\) then by definition of \(r\) we have \(a \leq r\);

Property (P6) let us assume \(\neg\#\text{rem} \geq a \in S\) and let us show \(r < a\). Hence, let \(b \in \mathbb{N}\) be such that \(\#\text{rem} \geq b \in S\) and let us show \(b < a\). We have \(\#\text{rem} \geq b, \neg\#\text{rem} \geq a \subseteq S\) hence by Property C1.4 we deduce \(b < a\). We conclude \(r < a\).

From Property (P0), we deduce that in the pre-canonical model \((s, h, l)\) of \(\text{cl}(B^+)\), if \(s(x_i) = s(x_j)\) then \(p_i = p_j\) by Proposition 5.5.

Now we show that the conjunction of the formulae in \(B^+ \cup \neg B^- \cup S\) is satisfiable if one of the following holds:

1. \((B^+, B^-, S)\) is 1-consistent and either \(\text{alloc}(u) \not\subseteq \text{cl}(B^+)\) or \((\text{Eq}_u \cup \text{To}_u) \cap \text{cl}(B^+) \neq \emptyset\);
2. \((B^+, B^-, S)\) is 2-consistent and \((\text{Eq}_u \cup \text{To}_u \cup \text{Fm}_u \cup \{u \mapsto u\}) \cap \text{cl}(B^+) \neq \emptyset\);
3. \((B^+, B^-, S)\) is 3-consistent.
Let us show (S1). We assume that \((B^+, B^-, S)\) is 1-consistent and either \(\text{alloc}(u) \notin \text{cl}(B^+)\) or \((\text{Eq}, T_0, \text{cl}(B^+)) \neq \emptyset\) hold, and we show that \(B^+ \cup \neg B^- \cup S\) is satisfiable. We consider the canonical pre-model \((s, h, l)\) of \(\text{cl}(B^+)\); see Proposition 5.3. If \(\text{alloc}(u) \notin \text{cl}(B^+)\) holds then \(l \notin \text{dom}(h)\); and if \((\text{Eq}, T_0, \text{cl}(B^+)) \neq \emptyset\) holds then \(l \in \text{p}^+(s, h, l)\). As \(\text{dom}(h) \subseteq \bigcup(s, h) \cup \{l\}\), under any of the two hypothesis \(\text{alloc}(u) \notin \text{cl}(B^+)\) or \((\text{Eq}, T_0, \text{cl}(B^+)) \neq \emptyset\) we have \(\text{dom}(h) \subseteq \bigcup(s, h)\). Hence \(\text{pred}_{\text{cl}(s, h, l)} = \text{loop}_{\text{cl}(s, h)} = \text{rem}_{\text{cl}(s, h)} = \emptyset\) for any \(i \in [1, q]\). From Proposition 5.5 with Property (P0), there exists a heap \(h'\) such that \((s, h, l) \models B^+ \cup \neg B^- \cup S\). Hence the conjunction of \(B^+ \cup \neg B^- \cup S\) is satisfiable.

Let us show (S2). We assume that \((B^+, B^-, S)\) is 2-consistent and \((\text{Eq}, T_0, \text{Fm}_h \cup \{u \mapsto u\}) \cap \text{cl}(B^+) \neq \emptyset\) and we show that \(B^+ \cup \neg B^- \cup S\) is satisfiable. We can further assume that \(\text{alloc}(u) \in \text{cl}(B^+)\) and \((\text{Eq}, T_0, \text{cl}(B^+)) \neq \emptyset\) because otherwise, as \((B^+, B^-, S)\) is 1-consistent, by Property (S1) we already have that \(B^+ \cup \neg B^- \cup S\) is satisfiable. Hence have either \(\text{Fm}_h \cap \text{cl}(B^+) \neq \emptyset\) or \(u \mapsto u \in \text{cl}(B^+)\):

- if \(u \mapsto x_i \in \text{cl}(B^+)\) for some \(i \in [1, q]\). In the canonical pre-model \((s, h, l)\) of \(\text{cl}(B^+)\), we have \(l \notin \text{pred}_{\text{cl}(s, h, l)}\). But since \(\text{dom}(h) \subseteq \bigcup(s, h) \cup \{l\}\), we deduce \(\text{pred}_{\text{cl}(s, h, l)} = \emptyset\) if \(s(x_i) = s(x_j)\), \(\text{pred}_{\text{cl}(s, h, l)} = \emptyset\) if \(s(x_i) \neq s(x_j)\) and \(\text{loop}_{\text{cl}(s, h)} = \text{rem}_{\text{cl}(s, h)} = \emptyset\). We consider two sub-cases depending on \([\# \text{pred}_{\text{cl}(s, h, l)} = 1 | x_i = x_j \in \text{cl}(B^+)\] \& \(S = \emptyset\).

- if \([\# \text{pred}_{\text{cl}(s, h, l)} = 1 | x_i = x_j \in \text{cl}(B^+)\] \& \(S = \emptyset\). Then let us define a new

Using the cardinality assignment \((p'_1, \ldots, p'_l, r)\), we extend the canonical pre-model \((s, h, l)\) of \(\text{cl}(B^+)\) using Proposition 5.5 and we get a heap \(h'\) such that \((s, h, l) \models B^+ \cup \neg B^- \cup S\) and \((s, h', l)\).

Let us check that \((s, h', l)\) satisfies all the formulae of \(B^+ \cup \neg B^- \cup S\). We show that \((s, h', l)\) satisfies the formulae of \(S\):

- if \([\# \text{pred}_{\text{cl}(s, h', l)} = 0 | a \in S\] then by Property (P1) we have \(a \leq p_j = p_j^{(p')} = \text{card}_{\text{pred}(s, h', l)}\), hence \((s, h', l) \models B^+ \cup \neg B^- \cup S\).

- if \([\# \text{loop}_{\text{cl}(s, h', l)} = 0 | a \in S\] then by Property (P4) we get \(\text{card}_{\text{loop}(s, h')} = 1 \leq a\), hence \((s, h', l) \models B^+ \cup \neg B^- \cup S\).

- if \([\# \text{rem}_{\text{cl}(s, h', l)} = 0 | a \in S\] then by Property (P6) we get \(\text{card}_{\text{rem}(s, h')} = r \leq a\), hence \((s, h', l) \models B^+ \cup \neg B^- \cup S\).
(B⁺ ∪ {B}, B⁻, S) is 1-consistent for some B ∈ Eq⁺ ∪ To. By Property (S1), we deduce that the conjunction of the formula of B⁺ ∪ {B} ∪ B⁻ ∪ S is satisfiable. Hence the conjunction of the formula of B⁺ ∪ B⁻ ∪ S is satisfiable as well:

- if u → u ∈ cl(B⁺). In the canonical pre-model of cl(B⁺), we have pred⁺(s, h, i) = rem⁺(s, h) = ∅ for any i ∈ [1, q] and loop⁺(s, h) = {l}. We consider two sub-cases:
  - either ¬# loop⁺ ≥ 1 ∉ S. As earlier, we extend the canonical pre-model (s, h, l) under the cardinality assignment (p₁, ..., pₚ, max(1, r)) using Proposition 5.5 and we get a heap h' such that (s, h, l) ≃₁ (s, h', l), card(pred⁺(s, h', i)) = pᵢ for any i ∈ [1, q], card(loop⁺(s, h')) = max(1, l) and card(rem⁺(s, h')) = r. We can then show that (s, h', l) satisfies the conjunction of the formula of B⁺ ∪ B⁻ ∪ S;
  - or ¬# loop⁺ ≥ 1 ∈ S. Then by Condition C2.2, (B⁺ ∪ {B}, B⁻, S) is 1-consistent for some B ∈ Eq⁺ ∪ To. By Property (S1), we deduce that the conjunction of the formula of B⁺ ∪ {B} ∪ B⁻ ∪ S is satisfiable. Hence the conjunction of the formula of B⁺ ∪ B⁻ ∪ S is satisfiable as well.

Let us finally show (S3). We assume that (B⁺, B⁻, S) is 3-consistent and we prove that the conjunction of the formula of B⁺ ∪ B⁻ ∪ S is satisfiable. We further assume that alloc(u) ∈ cl(B⁺) and (Eq⁺ ∪ To ∪ Fm ∪ {u ← u}) ∩ cl(B⁺) = ∅ because otherwise we can either apply Property (S1) or Property (S2). Hence in the canonical pre-model (s, h, l) of cl(B⁺), we have l ∈ rem⁺(s, h). But since dom(h) ≤ dom(s, h) ∪ {l}, we deduce pred⁺(s, h, i) = ∅ for any i ∈ [1, q], loop⁺(s, h) = ∅ and rem⁺(s, h) = {l}. We consider two cases:

- either ¬# rem⁺ ≥ 1 ∉ S. As earlier, we extend the canonical model (s, h, l) under the cardinal assignment (p₁, ..., pₚ, max(1, r)) using Proposition 5.5 and we get a heap h' such that (s, h, l) ≃₁ (s, h', l), card(pred⁺(s, h', i)) = pᵢ for any i ∈ [1, q], card(loop⁺(s, h')) = 1 and card(rem⁺(s, h')) = max(1, r). We deduce that (s, h', l) satisfies the conjunction of the formula of B⁺ ∪ B⁻ ∪ S;
- or ¬# rem⁺ ≥ 1 ∈ S. Then by Condition C3.1, (B⁺ ∪ {B}, B⁻, S) is 2-consistent for some B ∈ Eq⁺ ∪ To ∪ Fm ∪ {u ← u}. By Property (S2), we deduce that the conjunction of the formula of B⁺ ∪ {B} ∪ B⁻ ∪ S is satisfiable. Hence the conjunction of the formula of B⁺ ∪ B⁻ ∪ S is satisfiable as well. □