On Symbolic Heaps Modulo Permission Theories
S Demri, E Lozes, Denis Lugiez

To cite this version:
S Demri, E Lozes, Denis Lugiez. On Symbolic Heaps Modulo Permission Theories. 37th IARCS Foundation on Software Technology and Theoretical Computer Science, Dec 2017, Kanpur, India. 10.4230/LIPIcs.FSTTCS.2017.25 . hal-01788798

HAL Id: hal-01788798
https://amu.hal.science/hal-01788798
Submitted on 9 May 2018

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.
On Symbolic Heaps Modulo Permission Theories

S. Demri¹, E. Lozes¹, and D. Lugiez¹,²

¹ LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
{demri,lozes}@lsv.fr

² Aix-Marseille Univ, LIF, CNRS, Marseille, France
denis.lugiez@univ-amu.fr

Abstract
We address the entailment problem for separation logic with symbolic heaps admitting list predicates and permissions for memory cells that are essential to express ownership of a heap region. In the permission-free case, the entailment problem is known to be in P. Herein, we design new decision procedures for solving the satisfiability and entailment problems that are parameterised by the permission theories. This permits the use of solvers dealing with the permission theory at hand, independently of the shape analysis. We also show that the entailment problem without list predicates is coNP-complete for several permission models, such as counting permissions and binary tree shares but the problem is in P for fractional permissions. Furthermore, when list predicates are added, we prove that the entailment problem is coNP-complete when the entailment problem for permission formulae is in coNP, assuming the write permission can be split into as many read permissions as desired. Finally, we show that the entailment problem for any Boolean permission model with infinite width is coNP-complete.

1998 ACM Subject Classification D.2.4, Software/Program Verification, F.3. Logics and Meaning of Programs

Keywords and phrases separation logic, entailment, permission, reasoning modulo theories

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.25

1 Introduction

Separation logics with permissions. In program verification, proving properties of the memory is one of the most difficult tasks and separation logic has been devised for this goal [14]. Separation logic with permissions [4] can express that the ownership of a given heap region is shared with other threads. A permission can be thought of as a “quantity of ownership” associated to each cell of the heap. This quantity prescribes whether write accesses are allowed or not on this cell and how such a write access may be restored in the future. This abstract notion has lead to many permission theories and separation logics, including fractional permissions [5], token-based permissions [4], combinations of the two, binary tree shares [7], and yet some other models. Separation logic with permissions is supported by several tools like VeriFast [12], Hip/Sleek [11], or Heap-Hop [16]. Usually, these tools support only one permission model and demand that permissions are explicit values. For instance, in a tool that supports fractional permissions, to express that a cell x is shared by two threads for read access, one may write $x \xrightarrow{0.3} y$ and $x \xrightarrow{0.7} y$ making an arbitrary choice for permissions (0.3 and 0.7) when a better approach would use $x \xrightarrow{\alpha} y$ and $x \xrightarrow{\beta} y$ and the constraint $1 = \alpha + \beta$ (as it is done in the paper). This hides the logical structure of the proof and ties it to a specific arbitrary permission model.

Our motivations. We wish to get rid of the use of explicit permission models and to provide a separation logic with permissions which can use symbolic permission expressions.
such as $1 = \alpha + \beta$. Furthermore, we aim at lifting the results obtained so far for separation logic with lists but without permissions to separation logic with lists and symbolic permissions.

**Our contributions.** We devise a separation logic based on symbolic heaps with list predicates [3] modulo an unspecified permission theory (containing separation logic without permission as an instance). As far as we know, a uniform treatment with both features is new. We give generic decision procedures modulo a permission theory $\mathcal{P}$ for the satisfiability problem $\text{SATSH}(\mathcal{P})$, and for the entailment problem $\text{ENTSH}(\mathcal{P})$. Then we simply instantiate the permission theory by the desired theory (fractional models, token model, binary tree-share model, . . .). This approach has many advantages: (a) the reasoning on the spatial part is separated from the reasoning on permissions, (b) the latter part can be discharged to a dedicated solver, for instance any SMT specialised in the relevant permission theory (see e.g. [1]; and also [13] for the fractional case), and (c) we obtain optimal worst-case complexity results (obviously, the whole complexity depends on the complexity of the permission theory of interest). Since our logic contains the constant $\top$, we can treat both the intuitionistic and the non-intuitionistic case of the entailment problem in a uniform setting, see e.g. [6, 10]. Let us detail more precisely the technical contributions as well as the plan of the paper.

**Outline of the paper.** Permissions and separation logic with lists and permissions are introduced in Section 2. In Section 3, we treat separation logic with permissions but without lists and we give PTIME algorithms for $\text{SATSH}(\mathcal{P})$ and $\text{ENTSH}(\mathcal{P})$ using an oracle for the corresponding problems on permission theories. As a byproduct, $\text{SATSH}(\mathcal{P}_{\text{Boy}})$ and $\text{ENTSH}(\mathcal{P}_{\text{Boy}})$ without list predicates are in PTIME for the fractional model $\mathcal{P}_{\text{Boy}}$. In Section 4, we prove that $\text{SATSH}(\mathcal{P})$ is NP-hard and that $\text{ENTSH}(\mathcal{P})$ is coNP-hard even for a permission theory $\mathcal{P}$ which is in PTIME, showing a complexity gap between the logic without permissions and the logic with permissions. At the end of Section 4, we design a non-deterministic polynomial-time procedure solving $\text{ENTSH}(\mathcal{P})$ (fully parametrised by the entailment problem for the permission theory $\mathcal{P}$). A key ingredient is the notion of SL-graphs that are used to abstract formulae and several variants of homomorphisms between graphs used to prove the entailment property. This approach is clearly inspired by [6] but on one hand we can take advantage of nondeterminism since there is little hope for a PTIME algorithm, and on the other hand permissions lead to technical complications (such as the need to respect the linearisation induced by an SL-graph). In Section 5, we give our results on permission theories: (i) the fractional model $\mathcal{P}_{\text{Boy}}$ has PTIME satisfiability and entailment problems, (ii) we introduce the notion of Boolean permission models $\mathcal{P}_B$ that encompasses all classical permission models but the trivial one and $\mathcal{P}_{\text{Boy}}$ and (iii) we prove that SAT($\mathcal{P}_B$) is NP-complete and ENT($\mathcal{P}_B$) is coNP-complete in Boolean permission models $\mathcal{P}_B$ that have an infinite width (which is the case for the aforementioned models). Section 6 concludes the paper.

## 2 Preliminaries

We introduce permission formulae and permission models which are the building blocks for defining symbolic heaps with permissions and their related decision problems.

### 2.1 Permission models

Permission formulae are defined by the grammar below:

\[
p ::= \ 1 \ | \ \alpha \ | \ p \oplus p \quad \text{(permission term)}
\]

\[
A ::= \top \ | \ p \equiv p \ | \ p \preceq p \ | \ \text{defined}(p) \ | \ A \land A \quad \text{(permission formula)}
\]
where $\mathbb{P} \mathbb{V} \mathbb{A} \mathbb{R} = \{\alpha, \beta, \ldots\}$ is a countably infinite set of permission variables. Permission formulae are interpreted in permission models, defined below.

Definition 1. A permission model is a tuple $\mathfrak{P} = (P_\mathfrak{P}, \underline{1}_\mathfrak{P}, \oplus_\mathfrak{P})$ such that

- $P_\mathfrak{P} = \{\pi, \ldots\}$ is a set of permissions,
- $\underline{1}_\mathfrak{P} \in P_\mathfrak{P}$ is a distinguished permission called the write permission or the total permission,
- $\oplus_\mathfrak{P} : P_\mathfrak{P} \times P_\mathfrak{P} \rightarrow P_\mathfrak{P}$ is a partial composition that is cancellative, commutative and associative,
- the relation $\prec_\mathfrak{P} \overset{\text{def}}{=} \{(\pi', \pi) \mid \pi = \pi' \oplus_\mathfrak{P} \pi'' \text{ for some } \pi''\}$ is irreflexive and transitive, with maximum element $\underline{1}_\mathfrak{P}$.

An example of permission model is Boyland’s fractional model $\mathfrak{P}_{\text{Boy}} = ((0, 1], 1, \oplus_{\text{Boy}})$ [5], where $\pi \oplus_{\text{Boy}} \pi'$ is defined when the sum is at most 1. The width of a permission model $\mathfrak{P}$ is $\text{width}(\mathfrak{P}) \in \mathbb{N} \cup \{\omega\}$ such that $\text{width}(\mathfrak{P}) \overset{\text{def}}{=} \sup\{n \geq 1 \mid \exists \pi_1, \ldots, \pi_n \in P_\mathfrak{P} \text{ such that } \pi_1 \oplus \cdots \oplus \pi_n = \underline{1}_\mathfrak{P}\}$.

Given $\mathfrak{P} = (P_\mathfrak{P}, \underline{1}_\mathfrak{P}, \oplus_\mathfrak{P})$, a $\mathfrak{P}$-interpretation is a map $\iota : \mathbb{P} \mathbb{V} \mathbb{A} \mathbb{R} \rightarrow P_\mathfrak{P}$. The map $\iota$ is extended to a partial map from the set of permission terms to $P_\mathfrak{P}$ so that $\iota(\mathbf{1}) \overset{\text{def}}{=} \underline{1}_\mathfrak{P}$ and $\iota(p \oplus p') \overset{\text{def}}{=} \iota(p) \oplus \iota(p')$ if $\iota(p)$ and $\iota(p')$ are defined. Otherwise $\iota(p \oplus p')$ is undefined. We may write $p [\iota]$, for $\iota(p)$ and $\iota \models A$ to denote that $\iota$ satisfies the permission formula $A$, following the clauses below:

- $\iota \models \mathbf{T} ; \iota \models \text{defined}(p) \overset{\text{def}}{=} \iota(p)$ is defined; $\iota \models A \land A' \overset{\text{def}}{=} \iota \models A$ and $\iota \models A'$.
- $\iota \models p = p' \overset{\text{def}}{=} \iota$ is defined for both $p$ and $p'$ and $\iota(p) = \iota(p')$.
- $\iota \models p \preceq p' \overset{\text{def}}{=} \iota$ is defined for both $p$ and $p'$ and, either $\iota(p) = \iota(p')$ or $\iota(p) \prec_\mathfrak{P} \iota(p')$.

For example, the permission formula $\alpha \oplus \alpha = \mathbf{1}$, is satisfied by the $\mathfrak{P}_{\text{Boy}}$-interpretation $\iota$ defined by $\iota(\alpha) = 0.5$. We write $\perp$ to denote $\mathbf{1} \oplus \mathbf{1} = \mathbf{1}$. Observe that $\iota \not\models \perp$ for all $\iota$ (by irreflexivity of $\prec_\mathfrak{P}$).

2.2 Separation logic with permissions

A symbolic heap with list predicates and symbolic permissions is a formula $(\Pi, \Sigma)$ where $\Pi$ is a pure formula and $\Sigma$ a spatial formula according to the grammar below:

\[
\Pi ::= \top \mid x = y \mid x \neq y \mid \mathbf{A} \mid \Pi \land \Pi \quad \text{(pure formula)}
\]

\[
\Sigma ::= \mathbf{emp} \mid \top \mid x \overset{\rho}{\mathbf{P}} y \mid \mathbf{Iseg}_p(x, y) \mid \Sigma * \Sigma \quad \text{(spatial formula)}
\]

where $\mathbb{L} \mathbb{V} \mathbb{A} \mathbb{R} = \{x, y, \ldots\}$ is a countably infinite set of location/program variables. We write $\Pi_{\mathbb{L} \mathbb{V} \mathbb{A} \mathbb{R}}$ and $\Pi_{\mathbb{P} \mathbb{V} \mathbb{A} \mathbb{R}}$ to denote respectively the permission constraints and the program variable constraints that appear in $\Pi$, so that $\Pi$ is logically equivalent to $\Pi_{\mathbb{L} \mathbb{V} \mathbb{A} \mathbb{R}} \land \Pi_{\mathbb{P} \mathbb{V} \mathbb{A} \mathbb{R}}$. We write $\mathbb{L} \mathbb{V} \mathbb{A} \mathbb{R}(\varphi)$ [resp. $\mathbb{P} \mathbb{V} \mathbb{A} \mathbb{R}(\varphi)$] to denote the set of location [resp. permission] variables occurring in $\varphi$.

Example 2. The following symbolic heaps are used throughout the paper.

\[
(\Pi_1, \Sigma_1) \overset{\text{def}}{=} (\bigwedge_{1 \leq i \leq 3} \mathbf{x}_i \mathbf{\forall} x_j, \mathbf{Iseg}_a(x_1, x_3))
\]

\[
(\Pi_2, \Sigma_2) \overset{\text{def}}{=} (\bigwedge_{1 \leq i \leq 3} \mathbf{x}_i \mathbf{\forall} x_j, \mathbf{Iseg}_a(x_1, x_2) \ast \mathbf{Iseg}_a(x_2, x_3))
\]

\[
(\Pi_3, \Sigma_3) \overset{\text{def}}{=} \bigwedge_{1 \leq i \leq 3} \mathbf{x}_i \mathbf{\forall} x_j, \mathbf{Iseg}_a(x_1, x_3) \ast \mathbf{Iseg}_a(x_2, x_3) \ast \mathbf{Iseg}_a(x_2, x_1)).
\]

Let $\mathfrak{P} = (P_\mathfrak{P}, \underline{1}_\mathfrak{P}, \oplus_\mathfrak{P})$ be a fixed permission model and let $\mathbf{L} \mathbf{o} \mathbf{c} = \{\ell, \ldots\}$ be a countably infinite set of locations (by default, $\mathbf{L} \mathbf{o} \mathbf{c} = \mathbb{N}$). A $\mathfrak{P}$-memory state is a triple $(s, h, \iota)$ where:

---

1 in particular, whenever a sum $\pi_1 \oplus_\mathfrak{P} \pi_2 \ldots \oplus_\mathfrak{P} \pi_n$ is defined, each subsum $\pi_{i_1} \oplus_\mathfrak{P} \ldots \oplus_\mathfrak{P} \pi_{i_k}$ for each $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ is defined.
s is a store i.e. a function \( s : \text{LVAR} \to \text{Loc} \) that assigns to each variable a location,

- \( h \) is a \( \mathcal{P} \)-heap i.e. a partial function with a finite domain \( h : \text{Loc} \to_{\text{fin}} \mathcal{P} \times \text{Loc} \),

- \( \iota \) is a \( \mathcal{P} \)-interpretation.

Intuitively, \( h(\ell) = (\pi, \ell') \) holds if the cell at address \( \ell \) is allocated and points to the location \( \ell' \), and that the thread that owns \( \ell \) has permission \( \pi \) on the cell \( \ell \).

Before defining the semantics of symbolic heaps, we define the composition of \( \mathcal{P} \)-heaps. The composition \( h_1 \cdot h_2 \) of two \( \mathcal{P} \)-heaps \( h_1 \) and \( h_2 \) is defined whenever there is no \( \ell \in \text{dom}(h_1) \cap \text{dom}(h_2) \) with \( h_1(\ell) = (\pi_1, \ell_1) \) and \( h_2(\ell) = (\pi_2, \ell_2) \) such that either \( \ell_1 \neq \ell_2 \) or \( \pi_1 \oplus \pi_2 \pi_2 \) is undefined. When \( h_1 \cdot h_2 \) is defined, say equal to the \( \mathcal{P} \)-heap \( h \), it takes the unique value satisfying the conditions below:

- if \( \ell \not\in \text{dom}(h_1) \cup \text{dom}(h_2) \), then \( \ell \not\in \text{dom}(h) \),

- if \( \ell \in \text{dom}(h_1) \setminus \text{dom}(h_2) \), then \( \ell \in \text{dom}(h) \) and \( h(\ell) = h_1(\ell) \) (for all \( i \neq j \in \{1, 2\} \)),

- if \( \ell \in \text{dom}(h_1) \cap \text{dom}(h_2) \), then \( \ell \in \text{dom}(h) \) and \( h(\ell) = (\pi_1 \oplus \pi_2, \ell') \) with \( h_1(\ell) = (\pi_1, \ell') \), \( h_2(\ell) = (\pi_2, \ell') \) and \( \pi_1 \oplus \pi_2 \pi_2 \) is defined (otherwise \( \ell \not\in \text{dom}(h) \)).

The composition of heaps is partial, commutative, associative, and cancellative. We write Satisfiability and entailment problems. There is a corresponding problem in the permission model.

\( \iota \) and \( \alpha \) are defined by unique value satisfying the conditions below:

\[
\begin{align*}
s, h, \iota \models_{\mathcal{P}} x = y & \quad \text{iff} \quad s(x) = s(y) \\
s, h, \iota \models_{\mathcal{P}} x \neq y & \quad \text{iff} \quad s(x) \neq s(y) \\
s, h, \iota \models_{\mathcal{P}} \alpha & \quad \text{iff} \quad \iota \models \alpha \\
s, h, \iota \models_{\mathcal{P}} \Pi_1 \land \Pi_2 & \quad \text{iff} \quad s, h, \iota \models_{\mathcal{P}} \Pi_1 \text{ and } s, h, \iota \models_{\mathcal{P}} \Pi_2 \\
s, h, \iota \models_{\mathcal{P}} \text{exp} & \quad \text{iff} \quad \text{dom}(h) = \emptyset \\
s, h, \iota \models_{\mathcal{P}} x \mapsto y & \quad \text{iff} \quad \text{dom}(h) = \{s(x)\}, [p]_x \text{ is defined, and } h(s(x)) = ([p]_x, s(y)) \\
s, h, \iota \models_{\mathcal{P}} \exists \text{seg}(p,x,y) & \quad \text{iff} \quad [p]_x \text{ is defined, and either } s(x) = s(y) \text{ and } \text{dom}(h) = \emptyset \text{ or } h = \{\ell_0 \mapsto ([p]_x, \ell_1), \ell_1 \mapsto ([p]_x, \ell_2), \ldots, \ell_{n-1} \mapsto ([p]_x, \ell_n)\} \text{ with } n \geq 1, \ell_0 = s(x), \ell_n = s(y) \text{ and for all } i \neq j \in [0, n], \ell_i \neq \ell_j \\
s, h, \iota \models_{\mathcal{P}} \Sigma_1 \ast \Sigma_2 & \quad \text{iff} \quad \text{there are subheaps } h_1, h_2 \text{ such that } h = h_1 \cdot h_2, s, h_1, \iota \models_{\mathcal{P}} \Sigma_1 \text{ and } s, h_2, \iota \models_{\mathcal{P}} \Sigma_2.
\end{align*}
\]

Our definitions of symbolic heaps and models encompass the standard definitions without permissions: choose \( \mathcal{P}_1 = (\{1\}, 1, \oplus_{\mathcal{P}_1}) \) that has only the write permission and the always undefined composition. By way of example, given the permission model \( \mathcal{P}_\text{seg} \), let \((s, h, \iota)\) be defined by \( s(x_1) = 1, s(x_2) = 2, s(x_3) = 3, h = \{1 \mapsto (0.5, 3), 3 \mapsto (0.25, 2), 2 \mapsto (0.25, 2)\} \) and \( \iota(\alpha) = 0.25 \). Then, taking the \((\Pi_1, \Sigma_1)\)'s from Example 2, we have \( s, h, \iota \not\models_{\mathcal{P}_\text{seg}} (\Pi_1, \Sigma_1) \) but \( s, h, \iota \models_{\mathcal{P}_\text{seg}} (\Pi_2, \Sigma_2) \) (since 0.5 can be split into 0.25 + 0.25) and \( s, h, \iota \models_{\mathcal{P}_\text{seg}} (\Pi_3, \Sigma_3) \).

**Satisfiability and entailment problems.** A symbolic heap \((\Pi, \Sigma)\) is \( \mathcal{P} \)-satisfiable if there is a \( \mathcal{P} \)-memory state \((s, h, \iota)\) such that \( s, h, \iota \models_{\mathcal{P}} \Pi \) and \( s, h, \iota \models_{\mathcal{P}} \Sigma \) and we say that \((s, h, \iota)\) is a \( \mathcal{P} \)-model of \((\Pi, \Sigma)\). Two symbolic heaps \((\Pi, \Sigma)\) and \((\Pi', \Sigma')\) are equivalent, written \((\Pi, \Sigma) \equiv_{\mathcal{P}} (\Pi', \Sigma')\), if they have the same \( \mathcal{P} \)-models. The satisfiability problem w.r.t. \( \mathcal{P} \), written \( \text{SAT}_{\mathcal{P}}(\mathcal{P}) \), takes as input \((\Pi, \Sigma)\) and asks whether \((\Pi, \Sigma)\) has a \( \mathcal{P} \)-model. The entailment problem w.r.t. \( \mathcal{P} \), written \( \text{ENT}_{\mathcal{P}}(\mathcal{P}) \) takes as input two symbolic heaps \((\Pi, \Sigma)\) and \((\Pi', \Sigma')\) and asks whether every \( \mathcal{P} \)-model of \((\Pi, \Sigma)\) is a \( \mathcal{P} \)-model of \((\Pi', \Sigma')\) (written \((\Pi, \Sigma) \models_{\mathcal{P}} (\Pi', \Sigma')\)). In the paper, the decision procedures are parameterised by the corresponding problems in the permission model \( \mathcal{P} \), written \( \text{SAT}(\mathcal{P}) \) and \( \text{ENT}(\mathcal{P}) \).
We give our first results for symbolic heaps with permission but without lists. In the rest of
trivial. This new rewrite relation, denoted by
of Figure 2. Intuitively, we remove each points-to predicate of
check the existence of such a map by means of the rewrite rules

Lemma 4. The rewrite relation \( \rightarrow \) has the following properties.

- If \( (\Pi, \Sigma) \equiv (\Pi', \Sigma') \) then \( (\Pi, \Sigma) \equiv (\Pi', \Sigma') \).
- Any rewrite sequence starting from \( (\Pi, \Sigma) \) terminates in time \( O(|(\Pi, \Sigma)|) \).

The proof of the first part is a direct analysis of the rules according to the semantics
of symbolic heaps and the termination proof is straightforward. Note that the rule SUBST
cannot be applied indefinitely because of the second side-condition. From now on, unless
otherwise stated, normal form refers to a normal form with respect to \( \rightarrow \).

3.1 Normalising formulae

In Figure 2, we present a set \( R \) of rewrite rules that are used to normalise formulae. The
reduction \( \rightarrow \) is the rewrite relation associated to \( R \) and \( \rightarrow^* \) is its reflexive and transitive
closure. Note that if a rewrite sequence starts from a symbolic heap not containing an
expression \( lseg_p(x, y) \), then the rule MERGLIST never applies. We write \( |(\Pi, \Sigma)| \) to denote
the size of the symbolic heap for some reasonably succinct encoding.

\[ (\text{SUBST}) \quad (\Pi, \Sigma) \quad \rightarrow \quad (\Pi, \Sigma[y/x]) \quad \text{if} \quad \Pi \models x = y, \{x, y\} \subseteq \text{LVAR}(\Sigma) \]

\[ (\text{MERGE}) \quad (\Pi, \Sigma \star x \xrightarrow{\bowtie} y \star x' \xrightarrow{\bowtie} z) \quad \rightarrow \quad (\Pi \land y = z, \Sigma \star x \xrightarrow{p_{\bowtie}y'} y) \]

\[ (\text{FAIL}) \quad (\Pi, \Sigma) \quad \rightarrow \quad \bot \quad \text{if} \quad \Pi_{pv} \models x \neq y \quad \text{and} \quad \Pi_{pv} \models x = y \]

\[ (\text{EMPTY}) \quad (\Pi, \Sigma \star \text{emp}) \quad \rightarrow \quad (\Pi, \Sigma) \quad \text{if non-empty} \quad \Sigma \]

\[ (\text{TRUE}) \quad (\Pi, \Sigma \star \top \star \top) \quad \rightarrow \quad (\Pi, \Sigma \star \top) \]

\[ (\text{MERGLIST}) \quad (\Pi, \Sigma \star \text{lseg}_p(x, y) \star \text{lseg}_{p'}(x, y)) \quad \rightarrow \quad (\Pi, \Sigma \star \text{lseg}_{p → p'}(x, y)) \]

Figure 1 Rewrite system \( R \)

3 Reasoning Modulo Permission Theories Without List Predicates

3.1 Normalising formulae

In Figure 2, we present a set \( R \) of rewrite rules that are used to normalise formulae. The
reduction \( \rightarrow \) is the rewrite relation associated to \( R \) and \( \rightarrow^* \) is its reflexive and transitive
closure. Note that if a rewrite sequence starts from a symbolic heap not containing an
expression \( lseg_p(x, y) \), then the rule MERGLIST never applies. We write \( |(\Pi, \Sigma)| \) to denote
the size of the symbolic heap for some reasonably succinct encoding.

\[ (\text{SUBST}) \quad (\Pi, \Sigma) \quad \rightarrow \quad (\Pi, \Sigma[y/x]) \quad \text{if} \quad \Pi \models x = y, \{x, y\} \subseteq \text{LVAR}(\Sigma) \]

\[ (\text{MERGE}) \quad (\Pi, \Sigma \star x \xrightarrow{\bowtie} y \star x' \xrightarrow{\bowtie} z) \quad \rightarrow \quad (\Pi \land y = z, \Sigma \star x \xrightarrow{p_{\bowtie}y'} y) \]

\[ (\text{FAIL}) \quad (\Pi, \Sigma) \quad \rightarrow \quad \bot \quad \text{if} \quad \Pi_{pv} \models x \neq y \quad \text{and} \quad \Pi_{pv} \models x = y \]

\[ (\text{EMPTY}) \quad (\Pi, \Sigma \star \text{emp}) \quad \rightarrow \quad (\Pi, \Sigma) \quad \text{if non-empty} \quad \Sigma \]

\[ (\text{TRUE}) \quad (\Pi, \Sigma \star \top \star \top) \quad \rightarrow \quad (\Pi, \Sigma \star \top) \]

\[ (\text{MERGLIST}) \quad (\Pi, \Sigma \star \text{lseg}_p(x, y) \star \text{lseg}_{p'}(x, y)) \quad \rightarrow \quad (\Pi, \Sigma \star \text{lseg}_{p → p'}(x, y)) \]

Figure 1 Rewrite system \( R \)

3 Reasoning Modulo Permission Theories Without List Predicates

3.1 Normalising formulae

In Figure 2, we present a set \( R \) of rewrite rules that are used to normalise formulae. The
reduction \( \rightarrow \) is the rewrite relation associated to \( R \) and \( \rightarrow^* \) is its reflexive and transitive
closure. Note that if a rewrite sequence starts from a symbolic heap not containing an
expression \( lseg_p(x, y) \), then the rule MERGLIST never applies. We write \( |(\Pi, \Sigma)| \) to denote
the size of the symbolic heap for some reasonably succinct encoding.

\[ (\text{SUBST}) \quad (\Pi, \Sigma) \quad \rightarrow \quad (\Pi, \Sigma[y/x]) \quad \text{if} \quad \Pi \models x = y, \{x, y\} \subseteq \text{LVAR}(\Sigma) \]

\[ (\text{MERGE}) \quad (\Pi, \Sigma \star x \xrightarrow{\bowtie} y \star x' \xrightarrow{\bowtie} z) \quad \rightarrow \quad (\Pi \land y = z, \Sigma \star x \xrightarrow{p_{\bowtie}y'} y) \]

\[ (\text{FAIL}) \quad (\Pi, \Sigma) \quad \rightarrow \quad \bot \quad \text{if} \quad \Pi_{pv} \models x \neq y \quad \text{and} \quad \Pi_{pv} \models x = y \]

\[ (\text{EMPTY}) \quad (\Pi, \Sigma \star \text{emp}) \quad \rightarrow \quad (\Pi, \Sigma) \quad \text{if non-empty} \quad \Sigma \]

\[ (\text{TRUE}) \quad (\Pi, \Sigma \star \top \star \top) \quad \rightarrow \quad (\Pi, \Sigma \star \top) \]

\[ (\text{MERGLIST}) \quad (\Pi, \Sigma \star \text{lseg}_p(x, y) \star \text{lseg}_{p'}(x, y)) \quad \rightarrow \quad (\Pi, \Sigma \star \text{lseg}_{p → p'}(x, y)) \]
On Symbolic Heaps Modulo Permission Theories

preserves the entailment validity: if \((\Pi, \Sigma_l, \Sigma_r) \implies_{AS} (\Pi', \Sigma_l', \Sigma_r')\) then \((\Pi, \Sigma_l) \models (\top, \Sigma_r)\) iff \((\Pi', \Sigma_l') \models (\top, \Sigma_r')\).

\[(\text{ALIGN})\quad (\Pi, \Sigma_l \ast x \xrightarrow{p} y, \Sigma_r \ast x' \xrightarrow{p'} y') \quad \implies \quad (\Pi \wedge \text{defined}(p), \Sigma_l, \Sigma_r) \quad \text{if } \Pi \models p = p' \wedge x = x' \wedge y = y'.\]

\[(\text{SUBTRACT})\quad (\Pi, \Sigma_l \ast x \xrightarrow{p} y, \Sigma_r \ast x' \xrightarrow{p'} y') \quad \implies \quad (\Pi \wedge p = p' \oplus \alpha, \Sigma_l \ast x \xrightarrow{p} y, \Sigma_r) \quad \text{if } \Pi \models x = x' \wedge y = y'.\]

\(\text{Lemma 6. The algorithm terminates, and returns true iff } (\Pi, \Sigma) \models (\Pi', \Sigma').\)

\(\text{Theorem 7. The problem } \text{ENTSH}(\mathfrak{P}) \text{ restricted to symbolic heaps without list predicates can be decided in polynomial time with an oracle for } \text{ENT}(\mathfrak{P}).\)

Entailment checking without lists

\begin{itemize}
  \item \textbf{Input}: two symbolic heaps \((\Pi, \Sigma)\) and \((\Pi', \Sigma')\) without list predicates
  \item \textbf{Output}: returns \textbf{true} if \((\Pi, \Sigma) \models (\Pi', \Sigma')\), and \textbf{false} otherwise
\end{itemize}

\begin{algorithm}[H]
  \procedure{EntCheck}{$\Pi, \Sigma, \Pi', \Sigma'$}
  \begin{algorithmic}
    \State \textbf{put } \(\Pi, \Sigma\) in normal form
    \If {\(\Pi, \Sigma = \bot\) or \(\Pi_{pe} \wedge \text{defined}(\Sigma)\) is not \(\mathfrak{P}\)-satisfiable}
      \Return \textbf{true}
    \EndIf
    \If {\(\Pi_{pe} \wedge \text{defined}(\Sigma) \models \bot\)}
      \Return \textbf{false}
    \EndIf
    \ForAll {\text{atomic formulae } \varphi \text{ of } \Pi_{pe}}
      \State \textbf{let } \((\Pi''', \Sigma'')\) be the normal form of \((\Pi \wedge \neg \varphi, \Sigma)\)
      \If {\(\Pi''', \Sigma''\) \neq \bot \text{ and } \Pi_{pe} \wedge \text{defined}(\Sigma'')\) is \(\mathfrak{P}\)-satisfiable}
        \Return \textbf{false}
      \EndIf
    \EndFor
    \State \textbf{put } \(\Pi, \Sigma, \Sigma'\) in normal form w.r.t. \(\implies_{AS}\)
    \State \textbf{put } \(\Pi, \Sigma', \Sigma'\) in normal form w.r.t. \(\implies\)
    \State \Return \((\Sigma' = \top)\) or \((\Sigma = \Sigma' = \text{emp})\)
  \end{algorithmic}
\end{algorithm}

Using the results from Section 5, it follows that \text{ENTSH}(\mathfrak{P}_{Boy}) restricted to symbolic heaps without list predicates is in \text{PTime}.

4 Reasoning on Symbolic Heaps with Lists and Permissions

Below, we design algorithms for \text{SATSH}(\mathfrak{P}) and \text{ENTSH}(\mathfrak{P}) respectively, parameterised by decision problems for \(\mathfrak{P}\). Assuming that \(\Sigma\) and \(\Sigma'\) are \(\top\)-free and \(\text{emp}\)-free spatial formulae, we introduce the following subproblems of \text{ENTSH}(\mathfrak{P}):

\(\text{implies}_{I} = (\Pi, \Sigma) \models_{I} (\Pi', \Sigma') \iff (\Pi, \Sigma \ast \top) \models (\Pi', \Sigma' \ast \top).\) (intuitionistic entailment)

\(\text{implies}_{NI} = (\Pi, \Sigma) \models_{NI} (\Pi', \Sigma') \iff (\Pi, \Sigma) \models (\Pi', \Sigma').\) (non-intuitionistic entailment)

Below, we provide the developments for \(\models_{NI}\) only but all our results can be adapted for the full problems \text{SATSH}(\mathfrak{P}) and \text{ENTSH}(\mathfrak{P}). In the permission model \(\mathfrak{P}_1\), \text{SATSH}(\mathfrak{P}_1)\) and \text{ENTSH}(\mathfrak{P}_1)\) are in \text{PTime} [6] but tractability of \text{SATSH}(\mathfrak{P}) and \text{ENTSH}(\mathfrak{P}) happens quite quickly, even with the rather simple permission model \(\mathfrak{P}_{Boy}\). A positive consequence of Theorem 9 is that we can use non-determinism to get optimal complexity bounds.
4.1 Lower bounds

In this short section, we explain how the combination of list predicates and permission leads to NP/coNP-hardness. Let $G = (V, E)$ be an instance of the three-colorability problem, known to be NP-complete. W.l.o.g., we can assume that $V = \{x_1, \ldots, x_n\}$ for some $n \geq 1$ and $E$ is a set of edges of the form $(x_i, x_j)$ with $i \neq j$. The hardness proof is by reduction from the three-colorability problem following a similar treatment when conjunctions are added to spatial formulae in the standard symbolic heap fragment, see [6, Section 5]. The main difference below rests on the replacement of Boolean conjunctions by separating conjunctions. Let $\Pi_G$ be the pure formula $\bigwedge_{(x_i, x_j) \in E} (x_i \neq x_j) \land y_1 \neq y_2 \land y_1 \neq y_3 \land y_2 \neq y_3$. Let $\Sigma_G$ be the spatial formula $y_1 \leftarrow\rightarrow x_1 \leftarrow\rightarrow y_2 \leftarrow\rightarrow y_3 \leftarrow\rightarrow y_3 \leftarrow\rightarrow y_1 \leftarrow\rightarrow x_1 \leftarrow\rightarrow (\text{seg}_{\alpha_1}(y_1, x_1) \leftarrow\rightarrow \text{seg}_{\alpha_1}(x_i, y_3))$, where the $\alpha_i$’s and $\alpha_i'$’s are distinct permission variables.

**Lemma 8.** Assuming that $\text{width}(\mathcal{P}) = \omega$, $G$ has a three-coloring iff $(\Pi_G, \Sigma_G)$ is satisfiable.

Consequently, we get the following hardness results.

**Theorem 9.** If $\text{width}(\mathcal{P}) = \omega$ then $\text{SATSH(\mathcal{P})}$ is NP-hard and $\text{ENTSH(\mathcal{P})}$ is coNP-hard.

Observe that $\Pi, \Sigma$ is not $\mathcal{P}$-satisfiable iff $\Pi, \Sigma \models_{\mathcal{P}} x \neq x$. So, the NP-hardness of $\text{SATSH(\mathcal{P})}$ entails the coNP-hardness of $\text{ENTSH(\mathcal{P})}$.

4.2 SL-graphs and homomorphisms

We assume a fixed permission model $\mathcal{P}$ and a fixed set of variables $\{x_1, \ldots, x_q\}$ with its ordering $x_1 < \ldots < x_q$. All the symbolic heaps are assumed to be built from $\{x_1, \ldots, x_q\}$. Given $\emptyset \neq X \subseteq \{x_1, \ldots, x_q\}$, $\text{min}(X)$ denotes the variable in $X$ with the minimal index.

Below, we introduce a notion of SL-graph that can be understood as a graphical representation of a symbolic heap in which the permission part of the pure formula is encoded directly by a permission formula, the location variable part of a pure formula is encoded by an inequality relation (\(\notin\)) and by a labelling (L). Besides, the atomic spatial formulae are encoded by the two relations $\rightarrow$ and $\Rightarrow$. Such structures are quite convenient to characterise entailment between symbolic heaps via homomorphisms, as it is done in the permission-free setting in [6]. Contrary to the developments in [6] that aim to reach a PTIME upper bound, nondeterminism will be essential below since this is the best we can hope for, see Theorem 9.

\[\text{Figure 3 SL-graphs}\]

An **SL-graph** $G$ is either $\perp$ or a tuple $(A, V, \rightarrow, \Rightarrow, L)$ such that

- $A$ is a permission formula and $V$ is a non-empty finite subset of $\mathbb{N}$ (the nodes).
- $\rightarrow$ and $\Rightarrow$ are finite subsets of $V \times PT \times V$ where $PT$ is the set of permission terms. We also write $v \leftarrow\rightarrow v'$ [resp. $v \Rightarrow v'$] instead of $(v, p, v') \in \rightarrow$ [resp. $(v, p, v') \in \Rightarrow$]. We require a functionality condition: $v \leftarrow\rightarrow v'$ implies the unicity of $p$ and $v'$.
- $\leftarrow\rightarrow$ is an irreflexive and symmetric binary relation on $V$. 

\[\text{FSTTCS 2017}\]
On Symbolic Heaps Modulo Permission Theories

Given $v \in V$, we write $\vars(v)$ to denote the (non-empty) set $\{x \mid L(x) = v\}$ and $\var(v) \overset{\text{def}}{=} \min(\vars(v))$. As expected, the arrow $v \xrightarrow{A} v'$ (resp. $v \xleftarrow{A} v'$) is intended to represent the atomic formula $x \overset{L}{\rightarrow} x'$ [resp. $\lseg_p(x, x')$] whenever $L(x) = v$ and $L(x') = v'$. Moreover, we write $v \xrightarrow{P} v'$ whenever $v \xrightarrow{P} v'$ or $v \xrightarrow{P} v'$. Our notion of SL-graph shares in spirit the one from [6] but there are essential differences (permission formulae and terms as well as slight simplifications). Figure 3 presents SL-graphs where dashed lines encode the inequality relation, thick arrows encode $\Rightarrow$, normal arrows encode $\rightarrow$ and the permission formulae are omitted.

Below, we provide a semantics to the SL-graphs by defining a symbolic heap for each SL-graph. Given $G = (A, V, \rightarrow, \Rightarrow, L)$, $(\text{pure}(G), \text{spatial}(G))$ denotes the symbolic heap defined from $G$:

$$\text{pure}(G) \overset{\text{def}}{=} A \land (\bigwedge_{x_i, x_j, L(x_i) = L(x_j)} x_i = x_j) \land (\bigwedge_{v \in v^0} \var(v) \neq \var(v')).$$

$$\text{spatial}(G) \overset{\text{def}}{=} (\bigotimes_{v \in v^0} \var(v) \overset{P}{\rightarrow} \var(v')) \ast (\bigotimes_{v \in v^0} \text{lseg}_p(\var(v), \var(v'))) \ast v^0.$$

An empty separating conjunction is understood as $\text{emp}$. If $G = \perp$, then the corresponding symbolic heap is $(x \neq x, \top)$. In the sequel, $\text{spatial}(G)$ is also written $\bigotimes_{v \in v^0} \chi(v \overset{P}{\rightarrow} v')$, where $\chi(v \overset{P}{\rightarrow} v') \overset{\text{def}}{=} \var(v) \overset{P}{\rightarrow} \var(v')$ and $\chi(v \overset{P}{\rightarrow} v') \overset{\text{def}}{=} \text{lseg}_p(\var(v), \var(v')))$. An SL-graph $(A, V, \rightarrow, \Rightarrow, L)$ is deterministic iff for all $v \in V$, $\text{card}([v' \mid v \overset{P}{\rightarrow} v']) \leq 1$ and, for all $v \neq v' \in V$, we have $v \not \overset{P}{\leftrightarrow} v'$ ($G_2$ and $G_2$ are deterministic in Figure 3 unlike $G_1$ and $G_2$). A deterministic SL-graph can be viewed as a syntactic structure whose interpretation stands between the $P$-memory states (thanks to the deterministic and the syntactic nature of the inequality relation) and the SL-graphs (no $P$-interpretation and no permission values are involved). Each deterministic SL-graph carries a lot of structural properties about the $P$-memory states that satisfy it, which explains why this is a crucial structure to consider (see also the dependency graphs in [8]).

Let $G_1 = (A_1, V_1, \rightarrow_1, \Rightarrow_1, L_1)$ and $G_2 = (A_2, V_2, \rightarrow_2, \Rightarrow_2, L_2)$ be two SL-graphs with $G_2$ being deterministic. We introduce below a notion of precise homomorphism that will admit a counterpart in terms of entailment, see e.g. Theorem 11. A map $f : V_1 \rightarrow V_2$ is a precise homomorphism from $G_1$ to $G_2$ whenever the conditions below hold are satisfied:

(H0) $f$ is surjective.

(H1) $A^* \models A_1$ where $A^* = A_2 \land \bigwedge_i \text{defined}(p_i) \land \bigwedge_j \text{defined}(p'_j)$, $\rightarrow_2 = \{v_1 \overset{P_2}{\rightarrow} v_1', \ldots, v_n \overset{P_2}{\rightarrow} v_n'\}$ and $\Rightarrow_2 = \{\{u_1 \overset{\mathbf{P}_2}{\rightarrow} u_1', \ldots, u_m \overset{\mathbf{P}_2}{\rightarrow} u_m'\}\}$ (multiset notation).

(H2) For all $v \in V_1$, we have $\vars(v) \subseteq \vars(f(v))$.

(H3) For all $v, v' \in V_1$, $v \not \overset{\mathbf{P}_1}{\leftrightarrow} v'$ implies $f(v) \not \overset{\mathbf{P}_2}{\leftrightarrow} f(v')$.

(H4) For all $v, v' \in V_1$, $v \overset{\mathbf{P}_1}{\rightarrow} v'$ implies there is some permission term $p'$ such that $f(v) \overset{p'}{\rightarrow} f(v')$. We say that the edge $f(v) \overset{p'}{\rightarrow} f(v')$ contributes to the edge $v \overset{\mathbf{P}_1}{\rightarrow} v'$.

(H5) For all $v, v' \in V_1$, $v \overset{\mathbf{P}_1}{\rightarrow} v'$ implies either $(f(v) = f(v')$ and $A^* \models \text{defined}(p))$ or there is a path $v_0 \overset{P_1}{\rightarrow} v_1 \overset{P_1}{\rightarrow} v_2 \cdots \overset{P_1}{\rightarrow} v_n$ with $n \geq 1$, $v_0 = f(v)$, $v_n = f(v')$ and for all $i < j \in [0, n]$, $v_i \neq v_j$ (equivalent to $v_1 \not \overset{\mathbf{P}_1}{\leftrightarrow} v_j$ since $G_2$ is deterministic). For each edge $v_k \overset{\mathbf{P}_2}{\rightarrow} v_{k+1}$, we say that it contributes to the edge $v \overset{\mathbf{P}_1}{\rightarrow} v'$. Since $G_2$ is deterministic, there is a unique path satisfying the above condition (if any).
(H6′) Each edge $v \overset{p_1}{\rightarrow} v'$ in $G_2$ contributes to at least one edge of $G_1$, and we have $A^* \models \oplus\{\{p_1 \cup v \overset{p_1}{\rightarrow} v' \text{ contributes to } u \overset{p_1}{\rightarrow} u' \in G_1\}\} = p'$.

Above, given a non-empty and finite multiset $T = \{p_1, \ldots, p_k\}$ of permission terms, we write $\oplus T$ instead of $p_1 \oplus \cdots \oplus p_k$ (the ordering of the terms is irrelevant because $\oplus$ is AC). Precise homomorphisms could be defined between two arbitrary SL-graphs (as done in [6]) but the unicity of the path in the condition (H5) is not anymore guaranteed. We assume that $G_2$ is deterministic to have unicity, which also leads to the right upper bounds for the complexity.

The existence of a precise homomorphism $f$ implies that for all $u \overset{p_1}{\rightarrow} u' \in G_1$, we have $A^* \models \text{defined}(p)$, which is partly justified by $\text{defined}(p_1 \oplus p_2) \equiv \text{defined}(p_1) \land \text{defined}(p_2)$. The dotted arrows in Figure 3 partly materialize two precise homomorphisms from $G_1$ to $G_2'$ or to $G_2''$ (assuming the permission formulae match).

A precise homomorphism $f$ is strongly precise $\iff$ (H1') $A_2 = A_1 \land \bigwedge_{v \overset{p_1}{\rightarrow} v' \in G_1} \text{defined}(p)$ (which implies (H1)), (H5') $A$ is deterministic to have unicity, which also leads to the right upper bounds for the complexity.

4.3 From symbolic heaps to SL-graphs

Let $(\Pi, \Sigma)$ be a symbolic heap, possibly with list predicates and, $\Sigma$ is emp-free and $\top$-free. Let us define the SL-graph $\text{slg}(\Pi, \Sigma)$ as follows. If $(\Pi, \Sigma) \Rightarrow^* \bot$, then $\text{slg}(\Pi, \Sigma)$ is defined as the inconsistent SL-graph $\bot$ (see Section 3 for the definition of $\Rightarrow^*$).

Otherwise, assume that $(\Pi, \Sigma) \Rightarrow^* (\Pi', \Sigma')$ and $(\Pi', \Sigma')$ is in normal form (i.e., no rule can be further fired from $(\Pi', \Sigma')$). Let us define $\text{slg}(\Pi, \Sigma) \overset{\text{def}}{=} (A, V, \to, \varphi, \text{slg}, L)$ as follows:

- $A \overset{\text{def}}{=} \Pi_{\varphi, \Sigma}, V \overset{\text{def}}{=} \{i \in [1, q] \mid \text{there is no } j < i \text{ such that } \Pi' \models x_i = x_j\}.
- L(x_j) \overset{\text{def}}{=} \min\{j \in V \mid \Pi' \models x_i = x_j\}.
- \text{If } x \overset{\varphi}{\rightarrow} y \text{ occurs in } \Sigma', \text{ then } L(x) \overset{\varphi}{\rightarrow} L(y); \text{ if } \text{lsseg}(x, y) \text{ occurs in } \Sigma', \text{ then } L(x) \overset{\varphi}{\rightarrow} L(y).
- \text{For all } i \neq j \in [1, q], \text{ we have } \{L(x_i), L(x_j)\} \subseteq \overset{\varphi}{\rightarrow} \Pi' \models x_i \neq x_j.

In Figure 3, $G_1 = \text{slg}(\top, \text{lsseg}(\alpha, \varphi)(x_1, x_3) \ast \text{lsseg}(\alpha, \varphi)(x_3, x_2) \ast \overset{\beta}{\rightarrow} x_1)$. The soundness of the constructions between symbolic heaps and SL-graphs is best illustrated by Lemma 10.

> **Lemma 10.** $(\Pi, \Sigma) \equiv_\Psi (\text{pure}(\text{slg}(\Pi, \Sigma)), \text{spatial}(\text{slg}(\Pi, \Sigma)))$.  

4.4 Relating memory states and deterministic SL-graphs

Given $G_1$ and $G_2$, we write $f_{G_1, G_2} : V_1 \rightarrow V_2$ to denote the map s.t. $f_{G_1, G_2}(v) \overset{\text{def}}{=} L_2(\text{var}(v))$. Without any further assumption, note that $f_{G_1, G_2}$ is not necessarily a precise homomorphism.

Given a $\Psi$-memory state $(s, h, i)$ and a deterministic SL-graph $G = (A, V, \to, \varphi, \text{slg}, L)$, we write $(s, h, i) \overset{\text{in}}{\equiv} G$ whenever for all $v, v', v'' \in V$, if there are non-empty paths from $v$ to $v'$ and from $v'$ to $v''$ in $G$, then for all variables $x, x', x''$ such that $L(x) = v, L(x') = v'$ and $L(x'') = v''$, one of the conditions below holds:
1. there is no non-empty sequence of memory cells in \((s, h, i)\) from \(s(x)\) to \(s(x'')\),
2. there is no non-empty sequence of memory cells in \((s, h, i)\) from \(s(x')\) to \(s(x)\).

Roughly speaking, \((s, h, i) \overset{\text{lin}}{\approx} G\) holds when \((s, h, i)\) respects the linearisation induced by the graphical part of \(G\).

Below, we establish an equivalence between the existence of a precise homomorphism and the entailment \(\models_{NT}\). A similar statement can be found in [6] but herein we deal with permissions and with the deterministic SL-graphs. This is a key result at the heart of our whole enterprise. An auxiliary definition is needed. Given a deterministic SL-graph \(G'\) and a symbolic heap \((\Pi, \Sigma)\), we write \(\text{pure}(G''), \text{spatial}(G'') \models_{\text{lin}} \Pi, \Sigma\) iff for all \(\mathcal{Q}\)-memory states \((s, h, i)\) such that \((s, h, i) \overset{\text{lin}}{\approx} G\), if \((s, h, i) \models (\text{pure}(G''), \text{spatial}(G''))\), then \((s, h, i) \models \Pi, \Sigma\).

**Lemma 11.** Let \(G'\) be a deterministic SL-graph such that \((\text{pure}(G''), \text{spatial}(G''))\) is satisfiable. For all SL-graphs \(G\), the statements below are equivalent:

- The map \(f_{G,G'}\) is a precise homomorphism from \(G\) to \(G'\).
- \((\text{pure}(G''), \text{spatial}(G'')) \models_{\text{lin}} (\text{pure}(G), \text{spatial}(G))\).

Let us briefly explain below why in Lemma 11, \(\models_{\text{lin}}\) cannot be replaced by \(\models\). Let us consider the deterministic SL-graphs \(G_1, G_2\) and \(G_3\) with respective permission formulae \(\top\), \(\top \land \text{defined}(\alpha)\) and \(\top \land \text{defined}(\alpha) \land \text{defined}(\alpha)\) (all of them equivalent to \(\top\)).

Note that \(f_{G_1, G_2}\) and \(f_{G_2, G_3}\) are strongly precise homomorphisms and for \(i \in \{1, 2, 3\}\), \((\text{pure}(G_i), \text{spatial}(G_i))\) is logically equivalent to \((\Pi_i, \Sigma_i)\) from Example 2. However, not \((\text{pure}(G_2), \text{spatial}(G_2)) \models_{\text{lin}} \text{pure}(G_1), \text{spatial}(G_1))\). Indeed, let us consider the \(\mathcal{Q}\)-memory state \((s, h, i)\) defined in Section 2. We have \((s, h, i) \models (\text{pure}(G_2), \text{spatial}(G_2))\) and \((s, h, i) \not\models (\text{pure}(G_1), \text{spatial}(G_1))\). Actually, \((s, h, i) \overset{\text{lin}}{\approx} G_2\) is false. By contrast, \((s, h, i) \overset{\text{lin}}{\approx} G_3\).

By Lemma 11, we conclude however that \((\text{pure}(G_3'), \text{spatial}(G_3)) \models_{\text{lin}} (\text{pure}(G_2), \text{spatial}(G_2))\) and \((\text{pure}(G_2'), \text{spatial}(G_2')) \models_{\text{lin}} (\text{pure}(G_1), \text{spatial}(G_1))\), which is sufficient for our needs.

### 4.5 Decision procedures modulo permission theories

Let us characterise non-entailment between two symbolic heaps in the non-intuitionistic setting (leading to entailment of symbolic heaps from Figure 3 with \(G_2 = \text{alg}(x_1 \neq x_3 \land x_1 \neq x_2 \land \alpha = \alpha', \text{Is}^0_\alpha(x_1, x_2) \land \text{Is}^0_\alpha(x_1, x_3) \land x_2^\alpha \rightarrow x_1)\). This induces a nondeterministic algorithm by guessing the appropriate \(G\) (see below). Such a guess could be formalised in a proof system, as done for a fragment in Section 3, but herein we focus on the characterisation. In Theorem 12 below, note the use of \(\models_{\text{NT}}\) (instead of \(\models_{\text{lin}}\)).

**Theorem 12.** Let \((\Pi, \Sigma), (\Pi', \Sigma')\) be symbolic heaps s.t. neither \(\text{alg}(\Pi, \Sigma)\) nor \(\text{alg}(\Pi', \Sigma')\) is equal to \(\bot\). \((\Pi, \Sigma) \not\models_{\text{NT}} (\Pi', \Sigma')\) iff there is a deterministic SL-graph \(G\) that satisfies:

**\text{SMALL}** For every \(v, v'\) in \(G\), the permission term \(p\) is a sum of at most \(|\Sigma|\) terms from \(\text{alg}(\Pi, \Sigma)\). Moreover, the permission formula in \(G\) is precisely the permission part of \(\Pi\).

**\text{SAT}** \((\text{pure}(G), \text{spatial}(G))\) is satisfiable.

**\text{PRE}** \(f_{\text{alg}(\Pi, \Sigma), G}\) is a strongly precise homomorphism.
that (SMALL), (SAT) and (PRE) hold.

The singleton model is not equal to inequalities. Other classical permission models are: $P$, $separation logic without permission is a special case of the general case. The fractional model $P$ model Boolean permission models that generalise all existing permission models but the trivial $P$ model. Herein, we review permission models introduced for separation logic and we classify these $P$ model. Currently, the permission terms do not allow constants other than $1$ but for many permission models, constants can be easily added. For instance, in $P_{Boy}$, one can deal with $\frac{3}{4}$ by using the variable $\alpha$ thanks to $(\alpha' \oplus \alpha' \oplus \alpha' \oplus \alpha' = 1) \land (\alpha = \alpha' \oplus \alpha' \oplus \alpha')$ (generalisation to other rational numbers is obvious). Of course, depending on the permission models in mind, constants should be either introduced in the language of permission terms (according to other encoding) or can be enforced directly in the language, as it is the case for $P_{Boy}$.

5 Solving Permission Constraints

Herein, we review permission models introduced for separation logic and we classify these models according to the complexity of decision problems. In this process, we introduce Boolean permission models that generalise all existing permission models but the trivial model $P_{1}$ and the fractional model $P_{Boy}$ that admit PTime decision problems. We give optimal complexity results for the satisfiability and entailment problems. This is motivated by Theorem 15 below, which is a consequence of the parameterised decision procedures from Section 4 that separate the reasoning on the memory shapes from the one on permissions.

Theorem 14. (I) SAT($P_{Boy}$) is NP-complete and ENTSH($P_{Boy}$) is coNP-complete. (II) For any permission model $P$ such that width($P$) = $\omega$ and, ENT($P$) is in coNP, SATSH($P$) is NP-complete and ENTSH($P$) is coNP-complete.

Currently, the permission terms do not allow constants other than $1$ but for many permission models, constants can be easily added. For instance, in $P_{Boy}$, one can deal with $\frac{3}{4}$ by using the variable $\alpha$ thanks to $(\alpha' \oplus \alpha' \oplus \alpha' \oplus \alpha' = 1) \land (\alpha = \alpha' \oplus \alpha' \oplus \alpha')$ (generalisation to other rational numbers is obvious). Of course, depending on the permission models in mind, constants should be either introduced in the language of permission terms (according to a specified encoding) or can be enforced directly in the language, as it is the case for $P_{Boy}$.

Theorem 15. Let $P$ and $P'$ be permission models. (I) SAT($P$) = SAT($P'$) implies SATSH($P$) = SATSH($P'$). (II) ENT($P$) = ENT($P'$) implies ENTSH($P$) = ENTSH($P'$).

We briefly review the permission models that have been introduced in the literature. The singleton model $P_{1}$ presented in Section 2.2 is simply an artefact used to show that separation logic without permission is a special case of the general case. The fractional model $P_{Boy}$ already presented in Section 2, is one of the most popular permission models and it enjoys nice complexity properties stated below.

Theorem 16. SAT($P_{Boy}$) and ENT($P_{Boy}$) are in PTime.

The PTime bound is obtained by reduction to the satisfiability of a system of linear inequalities. Other classical permission models are:

- Bornat-Parkinson’s permission model $[4]$ with tokens is $P_{Tok} = (P_{Tok}, 1_{P_{Tok}}, \oplus_{P_{Tok}})$, where a permission $\pi \in P_{Tok}$ is either a finite or a co-finite, non-empty subset of $\mathbb{N}$, $1_{P_{Tok}}$ is $\mathbb{N}$, and $\pi \oplus_{P_{Tok}} \pi'$ is defined if $\pi \cap \pi' = \emptyset$ and is then equal to $\pi \cup \pi'$.
- Dockins-Hobor model, a.k.a binary shares $[7]$, is $P_{Bin} = ((\mathcal{T}(F)/ \equiv) \setminus \{0\}, \oplus, 1)$ where $\mathcal{T}(F)$ is the set of closed terms constructed over the function symbols $F = \{f : 2, 0 : 0, 1 : 0\}$, $\equiv$ is the least congruence such that $f(0, 0) \equiv 0$ and $f(1, 1) \equiv 1$, $\mathcal{T}(F)/ \equiv$ denotes the quotient, and $\oplus$ is defined by $0 \oplus 1 = 1 \oplus 0 = 1, 0 \oplus 0 = 0, 1 \oplus 0 = 1, 1 \oplus 1$ is undefined, and $f(\pi_1, \pi_2) \oplus f(\pi'_1, \pi'_2) \equiv f(\pi_1 \oplus \pi'_1, \pi_2 \oplus \pi'_2)$.
Note that in these two permission models a permission term $\alpha \oplus \alpha$ has no interpretation since the partial function $\oplus$ is not defined for identical elements. As a consequence, it holds for instance that \text{defined($\alpha \oplus \alpha$)} is unsatisfiable and \text{IsSeg$_{a \oplus a}(x,y) \models_{\mathfrak{B}} x \neq x$}.

These two permission models are particular instances of what we call Boolean permission models, i.e. permission models defined on top of a given Boolean algebra, as explained below. Let $B = (B_B, \wedge_B, \lor_B, \top_B, \bot_B, \neg_B)$ be a Boolean algebra. The permission model $\Psi_B$ associated to $B$ is $\Psi_B \defeq \langle P_B, \cup_B, \cap_B \rangle$ where $P_B = B_B \setminus \{ \bot_B \}$ and $\pi \oplus_B \pi'$ is defined when $\pi \land_B \pi' = \bot_B$, and in that case $\pi \oplus_B \pi' \defeq \pi \lor_B \pi'$. A permission model is Boolean if it is isomorphic to $\Psi_B$ for some Boolean algebra $B$. Both $\Psi_{Tok}$ and $\Psi_{Bin}$ are Boolean, the first one through the Boolean algebra of finite or co-finite subsets of $\mathbb{N}$, the second one through the Boolean algebra of open-closed sets of $\{0,1\}^\omega$, see e.g. [7]. As stated below, Boolean permission models are canonical in some sense.

\textbf{Lemma 17.} Let $A_L, A_R$ be permission formulae. Let $\Psi$ be a Boolean permission model with at least two elements such that $A_L \models_{\Psi} A_R$, and $\text{width(}\Psi\text{)} \geq \text{card(}\text{PVar}(A_L))$. Then for all Boolean permission models $\Psi'$, we have $A_L \models_{\Psi'} A_R$.

Lemma 17 entails $\text{ENT}(\Psi_{Tok}) = \text{ENT}(\Psi_{Bin})$. The case $\text{card}(P_B) = 1$ cannot be added to Lemma 17 since $\top \models_{\Psi_{a\alpha}} \alpha_1 = \alpha_2$ but $\bot \not\models_{\Psi_{a\alpha}} \alpha_1 = \alpha_2$ with $\alpha_1$ different from $\alpha_2$. Note also that if we could express the atomicity of a permission, the two models could be distinguished.

\textbf{Theorem 18.} $\text{ENT}(\Psi_{Tok}) = \text{ENT}(\Psi_{Bin})$ and $\text{SAT}(\Psi_{Tok}) = \text{SAT}(\Psi_{Bin})$.

From now on, we consider an arbitrary Boolean permission model $\Psi_B$ associated to a Boolean algebra $B$ such that width($\Psi_B$) = $\omega$. Boolean permission models behave quite nicely and below we establish that their decision problems are in $\text{coNP}$.

\textbf{Theorem 19.} Let $\Psi_B$ be a Boolean permission model such that width($\Psi_B$) = $\omega$. SAT($\Psi_B$) is $\text{NP}$-complete and, ENT($\Psi_B$) is $\text{coNP}$-complete.

A reduction from the NP-complete problem 1-in-3 SAT [15] gives the hardness result.

We give below the proof idea for SAT($\Psi_B$) is in NP. Actually, we can consider only permission terms equal to $1$ and of the form $\bigoplus \alpha_i$ and atomic permission formulae of the form $\bigoplus \alpha_i = \bigoplus \alpha_j$, $\bigoplus \alpha_i \leq \bigoplus \alpha_j$, $\bigoplus \alpha_i = 1$ or defined($\bigoplus \alpha_i$) and we can assume that each $A$ contains a conjunct defined($p$) for each permission term $p$. Let $A$ be a permission formula built on $\alpha_1, \ldots, \alpha_n$. We introduce an arithmetical formula $\psi_A$ such that $A$ is satisfiable iff $\psi_A$ is satisfiable. The Boolean/arithmetic variables of $\psi_A$ taking their values in $\{0,1\}$ are precisely $x_1^n, \ldots, x_1^{n_1}, \ldots, x_1^{n_m}$. The formula $\psi_A$ is a conjunction of formulae $\psi_1 \land \cdots \land \psi_n$ where each $\psi_i$ is built on the variables $x_1^{i_1}, \ldots, x_1^{i_n}$. For each $i \in [1,n]$, we define $t'(\alpha_j) \defeq x_{i_j}$, and $t'(p)$ replaces each occurrence of $\alpha_j$ by $t'(\alpha_j)$ and each occurrence of $\oplus$ by $+$, each occurrence of $1$ by $1$. Each $\psi_i$ is a conjunction of constraints defined by: (a) for each $p = p'$ [resp. $p \leq p'$] in $A$, $\psi_i$ contains $t'(p) \leq t'(p')$ and $t'(p') \leq t'(p)$ [resp. $t'(p) \leq t'(p')$]; (b) for each formula defined($p$) in $A$, $\psi_i$ contains $t'(p) \leq 1$, and $x_1^{i_1} = 1$ belongs to $\psi_i$. The next lemma relates $A$ and $\psi_A$.

\textbf{Lemma 20.} $A$ is satisfiable iff $\psi_A$ is satisfiable.

This lemma entails that SAT($\Psi_B$) is in NP.

\section{Conclusion}

Our results provide optimal complexity results about several standard permission models and are summarized by the following table. The algorithms can be implemented using any
checker following the standard viewpoint for SMT solvers [2] for reasoning on permission. Besides, this work could be continued in several directions, for instance to consider enriched permission theories (e.g., adding inequalities between permission terms), permission models without infinite width, to allow existential quantifications or to design sequent-style proof systems for checking entailment based on our characterisations.

|       | $\mathcal{P}_1$ | $\mathcal{P}_{\text{Boy}}$ | Boolean $\mathcal{P}_i$, width($\mathcal{P}_i$) = $\omega$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT($\mathcal{P}_i$)</td>
<td>in PTIME</td>
<td>in PTIME (Th. 16)</td>
<td>NP-C. (Th. 19)</td>
</tr>
<tr>
<td>ENT($\mathcal{P}_i$)</td>
<td>in PTIME</td>
<td>in PTIME (Th. 16)</td>
<td>coNP-C. (Th. 19)</td>
</tr>
<tr>
<td>SATSH($\mathcal{P}_i$)</td>
<td>in PTIME [6]</td>
<td>NP-C. (Th. 9,14(I))</td>
<td>NP-C. (Th. 9,14(II))</td>
</tr>
<tr>
<td>ENTSH($\mathcal{P}_i$) \ list pred.</td>
<td>in PTIME [6]</td>
<td>in PTIME (Th. 7)</td>
<td>coNP-C. (Th. 9,14(II))</td>
</tr>
<tr>
<td>ENTSH($\mathcal{P}_i$)</td>
<td>in PTIME [6]</td>
<td>coNP-C. (Th. 9,14(I))</td>
<td>coNP-C. (Th. 9,14(II))</td>
</tr>
</tbody>
</table>

Acknowledgements. We thanks the anonymous referees for their remarks and suggestions.

References