From Separation Logic to First-Order Logic

Cristiano Calcagno, Philippa Gardner, and Matthew Hague

Department of Computing
Imperial College
University of London

Abstract. Separation logic is a spatial logic for reasoning locally about heap structures. A decidable fragment of its assertion language was presented in [1], based on a bounded model property. We exploit this property to give an encoding of this fragment into a first-order logic containing only the propositional connectives, quantification over the natural numbers and equality. This result is the first translation from Separation Logic into a logic which does not depend on the heap, and provides a direct decision procedure based on well-studied algorithms for first-order logic. Moreover, our translation is compositional in the structure of formulae, whilst previous results involved enumerating either heaps or formulae arising from the bounded model property.

1 Introduction

Separation Logic [2] is a spatial logic for reasoning about mutable heap structures. It provides an elegant method for reasoning locally about separate areas of memory, and combining the results in a modular way. Its primary application is as the basis of a Hoare Logic for reasoning about memory update. An essential task is therefore to study decision procedures for validity checking, as part of a wider goal to develop verification tools for analysing C-programs.

The assertion language of Separation Logic is very expressive, due to the presence of two connectives: the separating conjunction $\phi_1 * \phi_2$ which asserts the existence of a split of the current heap into two disjoint sub-heaps that satisfy ϕ_1 and ϕ_2 respectively; and its adjunct implication $\phi_1 \twoheadrightarrow \phi_2$ which asserts that, whenever a fresh heap that satisfies ϕ_1 is composed with the current heap, then the result satisfies ϕ_2. In particular, validity checking is internalizable, which means that finding decision procedures is difficult.

Validity checking for the full Separation logic is undecidable [1]. Calcagno *et al.* have therefore been studying decidable fragments of the logic [1, 3]. They have shown that the Propositional Separation Logic (no quantifiers) is decidable [1], based on a finite model property which bounds the number of heaps that need to be checked. This is a surprising result since there is an implicit existential quantification in $*$, and more significantly an implicit universal quantification over fresh heaps in $\twoheadrightarrow$. However, their result does not provide a pragmatic decision procedure, since it relies on checking all the heaps of a certain size. In this paper we study a new approach. We provide a translation of Propositional Separation

Logic into a decidable fragment of first-order logic, for which decision procedures have been widely studied. We avoid the inefficient enumeration of the heaps by using the universal quantification of first-order logic.

As well as the results in [1], we take inspiration from the work of Dal Zilio *et al.* [4] which provides a novel decision procedure for the Static Ambient Logic [5]. Calcagno *et al.* adapted the decidability result of Propositional Separation Logic [1] to show decidability for the Static Ambient Logic, which relied this time on a finite model property for trees. Dal Zilio spotted a more efficient decision procedure for the Ambient Logic, that used a combination of Presburger Arithmetic and automata which did not depend on tree enumeration.

We provide a translation from Propositional Separation Logic into first order logic with only the propositional connectives, quantification over the natural numbers and equality. Our results rely on the bounded model property of [1]. The main idea is that vectors of a fixed length are used to represent all the states up to a given size. This means that we can represent sets of bounded states directly as first-order formulae over a fixed number of variables. The crucial cases in our translation are the connectives $*$ and $-\!*$. Since the current heap is decomposed by $*$ and extended by $-\!*$, the vector representation must change across subformulae. We define vector operations that represent decomposition and composition of heaps, and show that they simulate $*$ and $-\!*$. These results are then used to give a simple proof of correctness of our translation.

The expressiveness of Separation Logic can thus be obtained in an ordinary classical logic that is independent of heap structures. This is interesting because the translation provides a more elegant decision procedure than the one in [1] (which was based on enumerating all the heaps in a finite set arising from the finite model property). Since our translation is polynomial in the length of formula, we will be able to take advantage of the maturity of existing tools for first-order logic to provide an efficient decision procedure for Propositional Separation Logic.

In [6, 7], Lozes shows a related result that the spatial connectives can be eliminated from Propositional Separation Logic. His result is obtained by using the finite model property to produce a formula that is a disjunction of (characteristic formulae of) all heaps that satisfy the given formula. Their result differs from ours in that their target logic is not independent of heap structures and the method for translating the logic requires a decision procedure for a fragment of Separation Logic. More importantly, our translation is compositional in the structure of the formulae, and is not based on an enumeration of the exponential number of satisfying heaps. An immediate consequence of our approach is that a prover can use an existing axiomatization of first-order logic to output a direct proof. A complete axiomatization for Propositional Separation Logic is still an open problem.

The structure of the paper is the following. We begin in section 2 by introducing Propositional Separation Logic and its bounded model properties. In section 3 we present our vector representation of bounded heaps and the translation into first-order logic. In section 4 we discuss the conclusions of our work and describe several avenues for further research.

2   Propositional Separation Logic

In this section we present Propositional Separation Logic. This fragment of Separation Logic has the property that formulae can be assigned a size, which bounds the size of the states that need to be considered to check validity.

We begin by defining the sets of stacks and heaps, for which we need some notation.

Definition 1 (Notation). *We use the following notation. A partial function* $f : X \rightarrow_{fin} Y$ *is a finite map f from X to Y. We write $f\#g$ to indicate that partial maps f and g have disjoint domains. The composition of two partial functions f and g with disjoint domains is defined as $(f * g)(x) = y$ iff $f(x) = y$ or $g(x) = y$. The empty map is denoted $[]$. We use the notation $|\_|$ to indicate the cardinality of sets (which will be overloaded to also represent the size of formulae in Definition 3).*

Values, stacks, heaps, and states are defined as follows:

$$
\begin{aligned}
v \in \quad Val &\triangleq Loc \cup \{0\} \\
s \in Stack &\triangleq Var \rightarrow_{fin} Val \\
h \in Heap &\triangleq Loc \rightarrow_{fin} Val \times Val \\
(s, h) \in State &\triangleq Stack \times Heap
\end{aligned}
$$

where locations *Loc* are the natural numbers greater than zero. The value 0 represents the null location. A heap maps locations to binary heap cells and its domain indicates which locations are currently allocated. A stack is a partial function mapping program variables to values.

The syntax of Propositional Separation Logic is defined as follows

$E ::=$		Expressions
	x, y	Variables
	0	Nil
$\phi, \psi ::=$		Formulae
	$E = E$	Equality
	false	Falsity
	$\phi \Rightarrow \psi$	Implication
	$E \mapsto E_1, E_2$	Binary heap cell
	emp	Empty heap
	$\psi * \psi$	Composition
	$\psi \mathrel{-\!*} \psi$	Composition adjunct

The binary cell formula $E \mapsto E_1, E_2$ asserts that the location denoted by the expression E is the only allocated cell, and that it contains (E_1, E_2). The formula emp asserts that the heap is empty, i.e. no location is allocated. Composition $\phi * \psi$ means that the current heap can be split into two disjoint sub-heaps satisfying ϕ and ψ respectively. Its adjunct $\phi \mathrel{-\!*} \psi$ asserts that all heaps disjoint from the current heap and satisfying ϕ, when composed with the current heap satisfy ψ. The semantics is given by the satisfaction relation between formulae and states

Table 1. Semantics of formulae given a stack s and a heap h

$$J x K_s \triangleq s(x)$$
$$J 0 K_s \triangleq 0$$

$(s, h) \models E_1 = E_2$	iff $J E_1 K_s = J E_2 K_s$
$(s, h) \models$ false	never
$(s, h) \models \phi_1 \Rightarrow \phi_2$	iff $s, h \models \phi_1$ then $s, h \models \phi_2$
$(s, h) \models (E \mapsto E_1, E_2)$	iff $dom(h) = \{J E K_s\}$ and $h(J E K_s) = (J E_1 K_s, J E_2 K_s)$
$(s, h) \models$ emp	iff $dom(h) = \emptyset$
$(s, h) \models \phi_1 * \phi_2$	iff there exists h_1 and h_2 such that
	$h_1 \# h_2$; $h_1 * h_2 = h$; $s, h_1 \models \phi_1$ and $s, h_2 \models \phi_2$
$(s, h) \models \phi_1 \twoheadrightarrow \phi_2$	iff for all h_1 such that $h \# h_1$
	and $(s, h_1 \models \phi_1)$, $(s, h * h_1) \models \phi_2$

defined in Table 1. Standard logical connectives are defined as derived operators, such as $\neg \phi \triangleq (\phi \Rightarrow \text{false})$.

Definition 2 (Validity). *A formula ϕ is* valid *iff $(s, h) \models \phi$ holds for all states (s, h).*

Given a fixed stack, we can use $\twoheadrightarrow$ to reduce satisfaction for all heaps to satisfaction for the empty heap.

Lemma 1. *Given a stack s and a formula ϕ,*

$$(\forall h. \ (s, h) \models \phi) \iff ((s, []) \models (\neg \phi) \twoheadrightarrow \text{false})$$

Proof. Since $h * [] = h$, the assertion $(s, []) \models (\neg \phi) \twoheadrightarrow$ false states that any heap that satisfies $\neg \phi$ must also satisfy false. That is, no heap satisfies $\neg \phi$ and so ϕ holds for all heaps.

We now introduce the notion of size of formulae, as in [1].

Definition 3 (Size of Formulae). *Given a formula ϕ, its size $|\phi|$ is defined by*

$$|E_1 = E_2| = 0 \qquad\qquad\qquad |\text{false}| = 0$$
$$|\phi \Rightarrow \psi| = max(|\phi|, |\psi|) \qquad |(E \mapsto E_1, E_2)| = 1$$
$$|\text{emp}| = 1 \qquad\qquad\qquad |\phi * \psi| = |\phi| + |\psi|$$
$$|\phi \twoheadrightarrow \psi| = |\psi|$$

The size of a formula is used to determine a bound to the size of the heaps that need to be considered when checking validity, and to bound the size of new heaps needed to check satisfaction for formulae of the form $P \twoheadrightarrow Q$. Technically, one can define an equivalence relation $\sim_n$ on states, parameterized on the size

parameter n. The main property is that formulae of size n cannot distinguish between $\sim_n$-related states. For example, the size of $(x \mapsto y, z)$ is one because, in order to satisfy it or its negation, it is enough to consider heaps with at most one allocated location. The size of $\phi * \psi$ is the sum since $*$ combines subheaps together. The size of $\phi \twoheadrightarrow \psi$ is $|\psi|$ because $\sim_n$ is a congruence, and adding identical heaps in parallel (the ϕ part) does not affect the distinguishing power of formulae.

Because the semantics of $\twoheadrightarrow$ quantifies over all heaps, algorithmically determining if $(s, h) \models \phi$ for any formula ϕ is not straightforward. The following Proposition, which is an adaptation of an analogous one in [1], shows how to bound the size of new heaps that need to be considered.

Proposition 1. *For a given a state (s, h) and formulae ϕ_1 and ϕ_2, $(s, h) \models \phi_1 \twoheadrightarrow \phi_2$ holds iff for all h_1 such that,*

- *$h \# h_1$ and $(s, h_1) \models \phi_1$, and*
- *$|dom(h_1)| \leq max(|\phi_1|, |\phi_2|) + |FV(\phi_1) \cup FV(\phi_2)|$*

we have that $(s, h * h_1) \models \phi_2$.

Proof. The proposition is a corollary of Proposition 1 given on page 7 of [1].

The above Proposition requires $max(|\phi_1|, |\phi_2|)$ since the observations that $\phi_1 \twoheadrightarrow \phi_2$ can make on the *current* heap depend on both ϕ_1 and ϕ_2. It is worth noting that the set of heaps satisfying the properties in Proposition 1 is infinite (the size of heaps is bounded but the values contained are arbitrary), whereas the similar proposition in [1] explicitly defines a finite set of heaps. A finite set of heaps was necessary in [1] to give a direct decision procedure enumerating those heaps. However, our translation to first-order logic only depends on the *size* of heaps, so we chose a more abstract property.

To conclude the section we define bounded states, and give a bounding property for validity of formulae, which will be used in the translation presented in the next section. Bounded stacks and heaps are defined as follows.

Definition 4 ($S^{\mathbf{X}}$). *We write $S^{\mathbf{X}}$ to denote the set of stacks such that $s \in S^{\mathbf{X}}$ iff $dom(s) = \mathbf{X}$, where $\mathbf{X} \subseteq Var$.*

Definition 5 (H_p). *Given a size $p \in \mathbb{N}$, we write H_p to denote the set of heaps such that $h \in H_p$ iff $|dom(h)| \leq p$.*

Proposition 2. *Given a formula ϕ,*

$$\left(\forall(s, h). \ (s, h) \models \phi\right) \iff \left(\forall(s, h) \in S^{\mathbf{X}} \times H_p. \ (s, h) \models \phi\right)$$

where $\mathbf{X} = FV(\phi)$ and $p = |\phi| + |FV(\phi)|$.

Proof. The proposition follows immediately from Lemma 2 and Lemma 3 below.

Lemma 2. *Given a stack s and a formula ϕ,*

$$\left(\forall h. \ (s, h) \models \phi\right) \iff \left(\forall h \in H_{|\phi|+|FV(\phi)|}. \ (s, h) \models \phi\right)$$

Proof. By Lemma 1 we know that,

$$\left(\forall h. \ (s, h) \models \phi\right) \iff \left((s, [\,]) \models (\neg\phi) \twoheadrightarrow \text{false}\right)$$

By proposition 1, it follows that $(s, [\,]) \models (\neg\phi) \twoheadrightarrow \text{false}$ iff for all h_1 such that,

- $[\,] \# h_1$ and $(s, h_1) \models \neg\phi, and$
- $|dom(h_1)| \leq max(|\neg\phi|, |\text{false}|) + |FV(\neg\phi) \cup FV(\text{false})|$

we have that $(s, [\,] * h_1) \models \text{false}$. Which is equivalent to,

$$\forall h_1 \in H_{|\phi|+|FV(\phi)|}. \ (s, h) \models \phi$$

since $[\,] \# h_1$, $h_1 = [\,] * h_1$ and $max(|\neg\phi|, |\text{false}|) + |FV(\neg\phi) \cup FV(\text{false})| = |\phi| + |FV(\phi)|$. Therefore,

$$\left(\forall h. \ (s, h) \models \phi\right) \iff \left(\forall h \in H_{|\phi|+|FV(\phi)|}. \ (s, h) \models \phi\right)$$

as required.

Lemma 3. *Given a formula ϕ,*

$$\left(\forall(s, h). \ (s, h) \models \phi\right) \iff \left(\forall s \in S^{FV(\phi)}. \ \forall h. \ (s, h) \models \phi\right)$$

Proof. This is immediate from the semantics of Separation Logic since the values of variables that are not in $FV(\phi)$ do not affect the truth of ϕ.

3   Translating Separation Logic to First-Order Logic

In this section we present a translation from Separation Logic to first-order logic.

3.1   Representing States as Vectors

We represent bounded stacks in $S^{\mathbf{X}}$ and heaps in H_p as vectors of fixed length. This will allow us to replace quantification over bounded states by ordinary first-order quantification using a fixed number of variables.

Given a stack $s \in S^{\mathbf{X}}$, with $\{x_1, \ldots, x_n\} = \mathbf{X}$, we assume a fixed ordering on variables and define its representation $vs(s)$ simply as the vector $(s(x_1), \ldots, s(x_n))$. Heaps in H_p are represented as vectors $\mathbf{b}$ of p triples of values. The i-th triple $(\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \mathbf{b}_{i,3})$ potentially represents a heap cell. If $\mathbf{b}_{i,1}$ is a location (not 0), then the cell is allocated and contains the pair of values $(\mathbf{b}_{i,2}, \mathbf{b}_{i,3})$. If $\mathbf{b}_{i,1} = 0$ then the i-th triple does not represent a heap cell. For example, H_2

contains the singleton heap $(1 \mapsto 2, 3)$, which can be represented by the vector $((1, 2, 3), (0, 6, 7))$ or $((0, 8, 9), (1, 2, 3))$. The values $6, 7, 8, 9$ are unimportant since they do not belong to an active cell.

Note that all heaps in H_p have several vector representations, because the order of the heap cells, and the values of cells whose location is 0, are irrelevant. Also, not all vectors represent a valid heap, since the same location could occur more than once in the vector. We formalize the representation relation as a partial function vh_p from vectors to bounded heaps, defined in Table 2. A particular vector $\mathbf{b}$ is in the domain of vh_p iff it represents a well-formed heap.

Table 2. Definition of $vh_p(\mathbf{b})$

$$vh_p : (\mathbb{N} \times \mathbb{N} \times \mathbb{N})^p \rightharpoonup H_p$$

$$vh_p(\mathbf{b}) = \begin{cases} Undef & \text{if } \exists i, j \in 1..p. \ i \neq j \wedge \mathbf{b}_{i,1} = \mathbf{b}_{j,1} \wedge \mathbf{b}_{i,1} \neq 0 \wedge \mathbf{b}_{j,1} \neq 0 \\ \{(\mathbf{b}_{i,1} \mapsto \mathbf{b}_{i,2}, \mathbf{b}_{i,3}) | \mathbf{b}_{i,1} \neq 0 \wedge i \in 1..p\} & \text{otherwise} \end{cases}$$

Lemma 4. *For all p, vh_p is surjective:*

$$\forall h \in H_p \exists \mathbf{b}. \ vh_p(\mathbf{b}) = h$$

3.2   Representing Heaps in First-Order Logic

In this section we show how to use first-order formulae to represent heaps, and operations on heap representations corresponding to $*$ and $-\!\!*$.

We have seen that heaps are represented as vectors of triples of values. We now show how to represent assertions about heaps as first-order formulae from the following grammar

$$A ::= E = E \mid \text{false} \mid A \Rightarrow A \mid \forall x. A$$

with free variables drawn from a vector $\mathbf{B}$ of triples of variables. We write $\forall \mathbf{B}'. A$ as an abbreviation for $\forall \mathbf{B}'_{1,1} \forall \mathbf{B}'_{1,2} \cdots \forall \mathbf{B}'_{p,3}. A$ when $\mathbf{B}'$ is a vector of p triples of variables, and similarly for $\exists \mathbf{B}'. A$. We use the standard notation $\bigwedge_{i \in 1..n} . A$ for $A[1/i] \wedge \cdots \wedge A[n/i]$, and similarly for $\bigvee_{i \in 1..n} . A$. Given a vector of values $\mathbf{b}$ and a formula A with free variables from a vector $\mathbf{B}$, we write $[\mathbf{B} \ Z\!\!\Rightarrow \mathbf{b}] \models A$ for the usual satisfaction relation of first-order logic, where $[\mathbf{B} \ Z\!\!\Rightarrow \mathbf{b}]$ is the assignment of values to the variables.

We begin by defining the derived first-order formula $heap(\mathbf{B})$ that imposes restrictions on the values of the variables in $\mathbf{B}$ to ensure that they represent a valid heap.

Definition 6. *Given a vector of variables* $\mathbf{B}$,

$$heap(\mathbf{B}) \triangleq \left(\bigwedge_{\substack{i \in 1..|\mathbf{B}| \\ j \in 1..|\mathbf{B}| \\ i \neq j}} \left(\mathbf{B}_{i,1} = 0 \vee \mathbf{B}_{j,1} = 0 \vee \mathbf{B}_{i,1} \neq \mathbf{B}_{j,1} \right) \right)$$

The following lemma states that $heap(\mathbf{B})$ holds for a vector of values $\mathbf{b}$ exactly when $\mathbf{b}$ represents a heap, that is $\mathbf{b}$ will be in the domain of $vh_{|\mathbf{b}|}$.

Lemma 5. *Given vectors* $\mathbf{B}$, $\mathbf{b}$ *such that* $|\mathbf{B}| = |\mathbf{b}|$,

$$\mathbf{b} \in dom(vh_{|\mathbf{B}|}) \iff [\mathbf{B} \; Z\!\!\Rightarrow \mathbf{b}] \models heap(\mathbf{B})$$

Proof. Immediate from the definitions of $heap(\mathbf{B})$ and $vh_{|\mathbf{B}|}$.

We present two operators on vectors for constructing and deconstructing representations of heaps. These distinct operators are required because the spatial connectives $*$ and $-\!\!*$ manipulate the heap in different ways. First consider the composition connective $*$, which splits the current heap into two disjoint subheaps whose size and contents are limited by the original heap. We use the formula $\mathbf{B} = \mathbf{B}' \circledast \mathbf{B}''$, defined below, to capture this property where the vector of variables $\mathbf{B}$ represents the current heap, and the variables $\mathbf{B}', \mathbf{B}''$ represent the two subheaps. Because we do not know exactly how the heap will be split, the size of vectors $\mathbf{B}'$ and $\mathbf{B}''$ must each equal the size of $\mathbf{B}$, as in the worst case splitting the current heap will result in the current heap on one side and the empty heap on the other.

Definition 7 (Decomposition). *For vectors of variables* $\mathbf{B}, \mathbf{B}', \mathbf{B}''$ *such that* $|\mathbf{B}| = |\mathbf{B}'| = |\mathbf{B}''|$, *define*

$$\mathbf{B} = \mathbf{B}' \circledast \mathbf{B}'' \triangleq \bigwedge_{i \in 1..|\mathbf{B}|} \left(\left(\begin{array}{c} \mathbf{B}'_{i,1} = \mathbf{B}_{i,1} \wedge \mathbf{B}''_{i,1} = 0 \\ \wedge \mathbf{B}'_{i,2} = \mathbf{B}_{i,2} \wedge \mathbf{B}'_{i,3} = \mathbf{B}_{i,3} \end{array} \right) \\ \vee \left(\begin{array}{c} \mathbf{B}'_{i,1} = 0 \wedge \mathbf{B}''_{i,1} = \mathbf{B}_{i,1} \\ \wedge \mathbf{B}''_{i,2} = \mathbf{B}_{i,2} \wedge \mathbf{B}''_{i,3} = \mathbf{B}_{i,3} \end{array} \right) \right)$$

The extension to vectors of values is as follows

$$\mathbf{b} = \mathbf{b}' \circledast \mathbf{b}'' \quad \textit{iff} \quad [\mathbf{B} \; Z\!\!\Rightarrow \mathbf{b}, \mathbf{B}' \; Z\!\!\Rightarrow \mathbf{b}', \mathbf{B}'' \; Z\!\!\Rightarrow \mathbf{b}''] \models \mathbf{B} = \mathbf{B}' \circledast \mathbf{B}''$$

The following lemma shows that if $heap(\mathbf{B})$ holds then so does its decomposition.

Lemma 6. *For all vectors* $\mathbf{B}$, $\mathbf{B}'$, $\mathbf{B}''$, *the following is valid*

$$(\mathbf{B} = \mathbf{B}' \circledast \mathbf{B}'' \wedge heap(\mathbf{B})) \Rightarrow (heap(\mathbf{B}') \wedge heap(\mathbf{B}''))$$

Lemma 7 and Lemma 8 show that a splitting of heaps can be simulated by a corresponding splitting of representations, and vice versa.

Lemma 7. *For all* p, $\mathbf{b}$ *and* $h, h_1, h_2 \in H_p$,

$$h = h_1 * h_2 \wedge vh_p(\mathbf{b}) = h \Rightarrow \exists \mathbf{b}', \mathbf{b}''. \left(\begin{array}{c} \mathbf{b} = \mathbf{b}' \circledast \mathbf{b}'' \wedge \\ vh_p(\mathbf{b}') = h_1 \wedge vh_p(\mathbf{b}'') = h_2 \end{array} \right)$$

Lemma 8. *For all p, $\mathbf{b}, \mathbf{b}', \mathbf{b}''$ and $h \in H_p$,*

$$\mathbf{b} = \mathbf{b}' \circledast \mathbf{b}'' \wedge vh_p(\mathbf{b}) = h \Rightarrow h = vh_p(\mathbf{b}_1) * vh_p(\mathbf{b}'')$$

The composition adjunct $\twoheadrightarrow$ requires the addition of fresh heap cells to the current heap. The heap formed by the addition of these new cells may exceed the size that can be expressed by the current set of variables, which means that new variables need to be used to represent the new cells. We introduce the derived 'append' connective $\bullet$ to capture the addition of new heap cells.

Definition 8 ($\mathbf{B}' \bullet \mathbf{B}''$). *Given vectors $\mathbf{B}'$ and $\mathbf{B}''$ we define $\mathbf{B}' \bullet \mathbf{B}''$ as vector concatenation: $|\mathbf{B}' \bullet \mathbf{B}''| = |\mathbf{B}'| + |\mathbf{B}''|$ and for all $i \in 1..|\mathbf{B}' \bullet \mathbf{B}''|$,*

$$(\mathbf{B}' \bullet \mathbf{B}'')_i = \begin{cases} \mathbf{B}'_i & \text{if } i \in 1..|\mathbf{B}'| \\ \mathbf{B}''_i & \text{if } i \in (|\mathbf{B}'| + 1)..|\mathbf{B}' \bullet \mathbf{B}''| \end{cases}$$

The following lemma shows that if the result of appending two vectors represents a valid heap, then each vector represents a valid heap.

Lemma 9. *For all vectors $\mathbf{B}$, $\mathbf{B}'$, $\mathbf{B}''$ such that $\mathbf{B} = \mathbf{B}' \bullet \mathbf{B}''$, the following is valid*

$$heap(\mathbf{B}) \Rightarrow heap(\mathbf{B}') \wedge heap(\mathbf{B}'')$$

The following lemma captures the relationship between the composition of heaps and the appending of vectors.

Lemma 10. *For all, p_1, p_2, $\mathbf{b}', \mathbf{b}''$ and $h \in H_{p_1+p_2}$ such that $|\mathbf{b}'| = p_1$ and $|\mathbf{b}''| = p_2$,*

$$vh_{p_1+p_2}(\mathbf{b}' \bullet \mathbf{b}'') = h \iff h = vh_{p_1}(\mathbf{b}') * vh_{p_2}(\mathbf{b}'')$$

3.3   The Translation

We now have all the ingredients necessary to present the translation, which is defined in Table 3.

The translation $tran(\phi, \mathbf{B})$ produces a first-order formula with free variables in $\phi, \mathbf{B}$. For simplicity of notation we assume that the variables in ϕ and $\mathbf{B}$ are always disjoint (formally, we could use two syntactic categories). The translation begins with an implication, which effectively ignores all variable assignments that do not represent a heap. The bulk of the translation lies in $tran'(\phi, \mathbf{B})$.

The translations of $(E_1 = E_2)$, false, $(\phi_1 \Rightarrow \phi_2)$ and emp are fairly straightforward, but the translations of $(E \mapsto E_1, E_2)$, $(\phi_1 * \phi_2)$ and $(\phi_1 \twoheadrightarrow \phi_2)$ may benefit from an explanation.

The translation of the cell formula $E \mapsto E_1, E_2$ states that only one of the location variables $\mathbf{B}_{i,1}$ has a value that is non-zero — that is, the heap represented by the values of the variables has one cell only. Also, the values of the variables $(\mathbf{B}_{i,1}, \mathbf{B}_{i,2}, \mathbf{B}_{i,3})$ match the values of the expressions E, E_1 and E_2.

The Composition case $tran(\phi_1 * \phi_2, \mathbf{B})$ requires that we can split the current heap (the values of the variables in $\mathbf{B}$) into two parts, using $\mathbf{B} = \mathbf{B}' \circledast \mathbf{B}''$, such that the parts satisfy ϕ_1 and ϕ_2 respectively.

Table 3. Definition of $tran(\phi, \mathbf{B})$

$$tran(\phi, \mathbf{B}) \triangleq heap(\mathbf{B}) \Rightarrow tran'(\phi, \mathbf{B})$$

$$tran'(E_1 = E_2, \mathbf{B}) \triangleq E_1 = E_2$$
$$tran'(\text{false}, \mathbf{B}) \triangleq \text{false}$$
$$tran'(\phi_1 \Rightarrow \phi_2, \mathbf{B}) \triangleq tran'(\phi_1, \mathbf{B}) \Rightarrow tran'(\phi_2, \mathbf{B})$$
$$tran'(E \mapsto E_1, E_2, \mathbf{B}) \triangleq \bigvee_{i \in 1..|\mathbf{B}|} \left(\begin{array}{l} \mathbf{B}_{i,1} \neq 0 \wedge \bigwedge_{\substack{j \in 1..|\mathbf{B}| \\ i \neq j}} \left[\mathbf{B}_{j,1} = 0 \right] \\ \wedge\ \mathbf{B}_{i,1} = E \\ \wedge\ \mathbf{B}_{i,2} = E_1 \wedge \mathbf{B}_{i,3} = E_2 \end{array} \right)$$
$$tran'(\text{emp}, \mathbf{B}) \triangleq \bigwedge_{i \in 1..|\mathbf{B}|} \mathbf{B}_{i,1} = 0$$

$$tran'(\phi_1 * \phi_2, \mathbf{B}) \triangleq \exists \mathbf{B}', \mathbf{B}''. \left(\begin{array}{l} \mathbf{B} = \mathbf{B}' \circledast \mathbf{B}'' \\ \wedge\ tran'(\phi_1, \mathbf{B}') \\ \wedge\ tran'(\phi_2, \mathbf{B}'') \end{array} \right)$$

$$tran'(\phi_1 -\!\!* \phi_2, \mathbf{B}) \triangleq \forall \mathbf{B}'. \left(\begin{array}{l} tran'(\phi_1, \mathbf{B}') \\ \wedge\ heap(\mathbf{B} \bullet \mathbf{B}') \\ \quad \Rightarrow tran'(\phi_2, \mathbf{B} \bullet \mathbf{B}') \end{array} \right)$$
$$\text{where}$$
$$|\mathbf{B}'| = max(|\phi_1|, |\phi_2|) + |FV(\phi_1) \cup FV(\phi_2)|$$

Finally, the translation of $\phi_1 -\!\!* \phi_2$ quantifies over all heaps that satisfy ϕ_1 by universally quantifying over a new collection of heap variables — enough to represent all heaps up to the size required by Proposition 2. The formula $heap(\mathbf{B}' \bullet \mathbf{B})$ ensures that the combination of the old and new vectors still represent a heap, which implies that the new heap is disjoint from the current heap. The translation asserts that if the new heap satisfies ϕ_1 and it can be composed with the current heap, then the composition of both heaps satisfies ϕ_2, as required by the semantics of $-\!\!*$.

We now prove the correctness of the translation.

The free variables of the translated formula are the original stack variables plus the variables used to represent the current heap.

Lemma 11. *For any ϕ, $\mathbf{B}$,*

$$FV(tran(\phi, \mathbf{B})) = FV(\phi) \cup FV(\mathbf{B})$$

We show that, on related states, satisfaction is preserved by the translation.

Theorem 1. *For any ϕ, p, $\mathbf{B}$, $\mathbf{X}$, $\mathbf{b}$ where $|\mathbf{B}| = p$, $FV(\phi) \subseteq \mathbf{X}$, $(s, h) \in S^{\mathbf{X}} \times H_p$ and $vh_p(\mathbf{b}) = h$,*

$$(s, h \models \phi) \iff [\mathbf{B} \, \mathrm{Z}\!\!\Rightarrow \mathbf{b}, \mathbf{X} \, \mathrm{Z}\!\!\Rightarrow vs(s)] \models tran'(\phi, \mathbf{B})$$

A consequence of the theorem above is that the formula resulting from the translation cannot distinguish between two vectors representing the same heap.

Finally, we show that a formula is valid iff its translation is valid.

Theorem 2. *For any ϕ, $\mathbf{B}$, $\mathbf{X}$ such that $|\mathbf{B}| = |\phi| + |FV(\phi)|$ and $FV(\phi) \subseteq \mathbf{X}$,*

$$(\forall(s,h).\ (s,h) \models \phi) \iff \forall(\mathbf{b},\mathbf{v})\ [\mathbf{B} \mathrel{Z\!\!\Rightarrow} \mathbf{b}, \mathbf{X} \mathrel{Z\!\!\Rightarrow} \mathbf{v}] \models tran(\phi,\mathbf{B})$$

3.4   Decision Procedure and Complexity

Our decision procedure for Propositional Separation Logic simply consists of applying the translation followed by one of the existing decision procedures for first-order logic. The validity problem for first-order logic (on an empty signature) is a classical PSPACE-complete problem. In [1] it was proved that validity of Propositional Separation Logic is also PSPACE-complete.

Our translation into first-order logic generates a formula whose length is $\mathbf{O}(n^5)$ where n denotes the length[1] of the Separation Logic formula. This can be seen because, for each connective, the length of the vector (initially $\mathbf{O}(n)$) may increase by $\mathbf{O}(n)$ in the worst case (the $-\!\!*$ connective). Therefore, the length of the vector is always $\mathbf{O}(n^2)$. The translation of $E \mapsto E_1, E_2$ and $heap(\mathbf{B})$ are $\mathbf{O}(v^2)$, where v is the length of the vector. So, these formulae are $\mathbf{O}(n^4)$. In the worst case $\mathbf{O}(n)$ of these cases will occur, and therefore, the result of the translation will be $\mathbf{O}(n^5)$ in length.

This shows that the translation produces a limited increase in the length of formulae, therefore our decision procedure runs in polynomial space and has optimal theoretical complexity.

4   Conclusions and Future Work

In this paper we provided a translation from Propositional Separation Logic into first-order logic with only the propositional connectives, equality and quantification over the natural numbers. The translation has two main properties: a state satisfies a formula iff the state's vector encoding satisfies the translation, and a formula is valid iff its translation is valid. This translation shows that Separation Logic can be expressed in a classical logic that has no notion of a heap or spatial connectives. It also provides a new decision procedure that can utilise existing tools for first-order logic.

A natural direction for future work is implementing and evaluating the new decision procedure. In [8], we implemented the decision procedure for Tree Logic which inspired the work presented here. Using several optimisations, we found that the decision procedure was viable. We hope that, utilising possible optimisations, an implementation of this work may show similar results.

---

[1] We use 'length' with the usual meaning: the number of connectives in the formula, not the size of Definition 3.

For example, we may reduce the number of existentially quantified variables when translating $\phi_1 * \phi_2$ by only quantifying one set of variables ($\mathbf{B}'$) and calculating the second ($\mathbf{B}''$) in situ through the use of expressions rather than variables.

We may also wish to consider different fragments of Separation Logic or extensions of the fragment studied in this paper. For example, if we change the target logic of the translation to Presburger Arithmetic, we gain addition of natural numbers. This would allow us to augment the quantifier-free fragment of Separation Logic with arithmetic on stack variables. However, allowing arithmetic on the heap may invalidate the size argument on which Proposition 1 and Proposition 2 are based. Another extension is allowing quantification of variables ($\exists x. \phi$). The presence of full existential quantifiers also invalidates the size argument of Proposition 1 and Proposition 2. However, it is likely that restricted (e.g. guarded) forms of quantification admit a size argument. In those cases, the translation can be extended by mapping existentials to existentials, since the proofs extend trivially. We may also attempt to extend our results to the more practically motivated fragment of Separation Logic in [3], which was designed for reasoning about linked lists. That fragment presents a different technical challenge to the one presented here: there is no $-\!*$ but there is an inductive definition for linked lists. We expect our techniques to prove useful also in that setting.

A new related area of research into Spatial Logics [5, 9, 10, 11] is 'trees with pointers', which add location identifiers and cross-references to Tree Logic [12]. A practical example of this model is XML cross-references. This model combines Tree Logic and Separation Logic because the tree structures have locations on nodes, and pointers as data. Preliminary work on decision procedures for this model has identified several subtleties. First, a notion of size must be identified. A likely candidate is the maximum number of locations required at any level of the tree and the maximum depth of the tree. Secondly, a succinct method for ensuring that all locations are unique is required. At a single level of the tree this task is exactly the same as for Separation Logic. However, as the decision procedure divides the tree into independent sub-trees, enforcing the uniqueness of locations becomes a more difficult task.

Finally, we would like to study decidability properties of Context Logic [13]. This new logic uses contexts or 'trees with holes' to allow reasoning about smaller sub-trees within larger arbitrary trees. Context logic has been used to provide a Hoare logic for reasoning about tree updates, where the portion of tree left untouched by the update has the shape of a tree context. A decision procedure for this logic presents a further challenge to the 'trees with pointers' model because it would require a different notion of size.

References

1. Calcagno, C., Yang, H., O'Hearn, P.: Computability and complexity results for a spatial assertion language for data structures. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01), Springer (2001) 108–119 volume 2245 of Lecture Notes in Computer Science.
2. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS, IEEE (2002) 55–74
3. Berdine, J., Calcagno, C., O'Hearn, P.: A decidable fragment of separation logic. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04), Springer (2004) to appear.
4. Zilio, S.D., Lugiez, D., Meyssonnier, C.: A logic you can count on. In: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM Press (2004) 135–146
5. Cardelli, L., Gordon, A.D.: Anytime, anywhere: Modal logics for mobile ambients. In: 27th Symposium on Principles of Programming Languages (POPL'00), ACM (2000) 365–377
6. Lozes, E.: Separation logic preserves the expressive power of classical logic. As published at:
http://www.diku.dk/topps/space2004/space_final/etienne.pdf (2004)
7. Lozes, E.: Elimination of spatial connectives in static spatial logics. To Appear in TCS (2004)
8. Hague, M.: Static checkers for tree structures and heaps. Master's thesis, Imperial College London, Department of Computing (2004) http://www.doc.ic.ac.uk/~ajf/Teaching/Projects/Distinguished04/MatthewHague.pdf.
9. Cardelli, L., Caires, L.: A spatial logic for concurrency (part I). Journal of Information and Computation **186(2)** (2003)
10. Cardelli, L., Caires, L.: A spatial logic for concurrency (part II). To Appear in Theoretical Computer Science (2004)
11. Cardelli, L., Gardner, P., Ghelli, G.: A spatial logic for querying graphs. Proceedings of ICALP'02 (2002)
12. Cardelli, L., Gardner, P., Ghelli., G.: Querying trees with pointers. Unpublished Notes, 2003; talk at APPSEM 2001 (2003)
13. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic and tree update. To appear in POPL (2005)

A   Appendix: Selected Proofs

A.1   Proof of Theorem 1 from Section 3.3

Theorem 1 states that for any ϕ, p, $\mathbf{B}$, $\mathbf{X}$, $\mathbf{b}$ where $|\mathbf{B}| = p$, $FV(\phi) \subseteq \mathbf{X}$, $(s, h) \in S^{\mathbf{X}} \times H_p$ and $vh_p(\mathbf{b}) = h$,

$$(s, h \models \phi) \iff [\mathbf{B} \, \mathrm{Z}\!\Rightarrow \mathbf{b}, \mathbf{X} \, \mathrm{Z}\!\Rightarrow vs(s)] \models tran'(\phi, \mathbf{B})$$

Proof. The proof is by induction over ϕ. We only consider some interesting cases.

Case $\phi = (\phi_1 * \phi_2)$.

$\Rightarrow$: Assume $(s, h) \models \phi_1 * \phi_2$. Therefore $h = h_1 * h_2$ and $(s, h_1) \models \phi_1$ and $(s, h_2) \models \phi_2$. Therefore, by Lemma 7 there exist $\mathbf{b^1}, \mathbf{b^2}$ such that,

$$\mathbf{b} = \mathbf{b^1} \circledast \mathbf{b^2} \wedge vh_p(\mathbf{b^1}) = h_1 \wedge vh_p(\mathbf{b^2}) = h_2$$

By induction and since $vh_p(\mathbf{b^1}) = h_1$ and $vh_p(\mathbf{b^2}) = h_2$,

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b^1}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models tran'(\phi_1, \mathbf{B})$$

and

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b^2}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models tran'(\phi_2, \mathbf{B})$$

Therefore,

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models \exists \mathbf{B^1}, \mathbf{B^2}. \begin{pmatrix} \mathbf{B} = \mathbf{B^1} \circledast \mathbf{B^2} \\ \wedge\ tran'(\phi_1, \mathbf{B^1}) \\ \wedge\ tran'(\phi_2, \mathbf{B^2}) \end{pmatrix}$$

And so,

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models tran'(\phi, \mathbf{B})$$

$\Leftarrow$: Assume,

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models \exists \mathbf{B^1}, \mathbf{B^2}. \begin{pmatrix} \mathbf{B} = \mathbf{B^1} \circledast \mathbf{B^2} \\ \wedge\ tran'(\phi_1, \mathbf{B^1}) \\ \wedge\ tran'(\phi_2, \mathbf{B^2}) \end{pmatrix}$$

Therefore, there exists $\mathbf{b_1}, \mathbf{b_2}$ such that $\mathbf{b} = \mathbf{b_1} \circledast \mathbf{b_2}$,

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b_1}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models tran'(\phi_1, \mathbf{B})$$

and

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b_2}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models tran'(\phi_2, \mathbf{B})$$

By Lemma 8, letting $h_1 = vh_p(\mathbf{b_1})$ and $h_2 = vh_p(\mathbf{b_2})$, we know $h = h_1 * h_2$ and by induction $(s, h_1) \models \phi_1$ and $(s, h_2) \models \phi_2$. Therefore $(s, h) \models (\phi_1 * \phi_2)$, that is, $(s, h) \models \phi$.

Case $\phi = (\phi_1 \mathbin{-\!\!*} \phi_2)$.

$\Rightarrow$: Assume $(s, h) \models (\phi_1 \mathbin{-\!\!*} \phi_2)$. Therefore, for all h_1 such that $(s, h_1) \models \phi_1$ and $h \# h_1$, $(s, h * h_1) \models \phi_2$.

We now assume $\mathbf{b'}$ such that,

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b}, \mathbf{B'}\ Z{\Rightarrow}\ \mathbf{b'}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models tran'(\phi_1, \mathbf{B'}) \wedge heap(\mathbf{B} \bullet \mathbf{B'})$$

By Lemma 9 we know $[\mathbf{B'}\ Z{\Rightarrow}\ \mathbf{b'}] \models heap(\mathbf{B'})$ and so $\mathbf{b'} \in dom(vh_{|\mathbf{B'}|})$ by Lemma 5. Let $vh_{|\mathbf{B'}|}(\mathbf{b'}) = h_1$, we know by induction that $(s, h_1) \models \phi_1$. By Lemma 5 $vh_{p+|\mathbf{B'}|}(\mathbf{b}\bullet\mathbf{b'})$ is defined. Therefore by Lemma 10 $h * h_1 = vh_p(\mathbf{b}) * vh_{|\mathbf{B'}|}(\mathbf{b'}) = vh_{p+|\mathbf{B'}|}(\mathbf{b} \bullet \mathbf{b'})$. By assumption $s, h * h_1 \models \phi_2$. Consequently, by induction we have,

$$[\mathbf{B}\ Z{\Rightarrow}\ \mathbf{b}, \mathbf{B'}\ Z{\Rightarrow}\ \mathbf{b}, \mathbf{X}\ Z{\Rightarrow} vs(s)] \models tran'(\phi_2, \mathbf{B} \bullet \mathbf{B'})$$

Therefore,

$$[\mathbf{B} \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}, \mathbf{X} \; \mathbf{Z}{\Rightarrow} \; vs(s)] \models \forall \mathbf{B}'. \begin{pmatrix} tran'(\phi_1, \mathbf{B}') \\ \wedge \; heap(\mathbf{B} \bullet \mathbf{B}') \\ \Rightarrow tran'(\phi_2, \mathbf{B} \bullet \mathbf{B}') \end{pmatrix}$$

and

$$[\mathbf{B} \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}, \mathbf{X} \; \mathbf{Z}{\Rightarrow} \; vs(s)] \models tran'(\phi, \mathbf{B})$$

$\Leftarrow$: Assume,

$$[\mathbf{B} \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}, \mathbf{X} \; \mathbf{Z}{\Rightarrow} \; vs(s)] \models \forall \mathbf{B}'. \begin{pmatrix} tran'(\phi_1, \mathbf{B}') \\ \wedge \; heap(\mathbf{B} \bullet \mathbf{B}') \\ \Rightarrow tran'(\phi_2, \mathbf{B} \bullet \mathbf{B}') \end{pmatrix}$$

where $|\mathbf{B}'| = max(|\phi_1|, |\phi_2|) + |FV(\phi_1) \cup FV(\phi_2)|$.

By Proposition 1 $(s, h) \models (\phi_1 \; {-\!\!*} \; \phi_2)$ iff for all $h_1 \in H_q$ such that $q = max(|\phi_1|, |\phi_2|) + |FV(\phi_1) \cup FV(\phi_2)|$, $h\#h_1$ and $(s, h_1) \models \phi_1$ we have $s, h * h_1 \models \phi_2$. So assume we have $h_1 \in H_q$ such that $h\#h_1$ and $(s, h_1) \models \phi_1$. By Lemma 4 there exists $\mathbf{b}'$ such that $vh_q(\mathbf{b}') = h_1$. Since $h\#h_1$ we know that $h * h_1 \in H_{p+q}$. By Lemma 10 we know $h * h_1 = vh_p(\mathbf{b}) * vh_q(\mathbf{b}') = vh_{p+q}(\mathbf{b} \bullet \mathbf{b}')$, and so, by Lemma 5 $[\mathbf{B} \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}, \mathbf{B}' \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}'] \models heap(\mathbf{B} \bullet \mathbf{B}')$. By induction we know

$$[\mathbf{B}' \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}', \mathbf{X} \; \mathbf{Z}{\Rightarrow} \; vs(s)] \models tran'(\phi_1, \mathbf{B}')$$

It follows then that

$$[\mathbf{B} \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}, \mathbf{B}' \; \mathbf{Z}{\Rightarrow} \; \mathbf{b}', \mathbf{X} \; \mathbf{Z}{\Rightarrow} \; vs(s)] \models tran'(\phi_2, \mathbf{B} \bullet \mathbf{B}')$$

And by induction $(s, h * h_1) \models \phi_2$ as required.