Computability and Complexity Results for a Spatial Assertion Language for Data Structures

Cristiano Calcagno[1,2], Hongseok Yang[3], and Peter W. O'Hearn[1]

[1] Queen Mary, University of London
[2] DISI, University of Genova
[3] ROPAS, KAIST

Abstract. This paper studies a recently developed an approach to reasoning about mutable data structures, which uses an assertion language with spatial conjunction and implication connectives. We investigate computability and complexity properties of a subset of the language, which allows statements about the shape of pointer structures (such as "there is a link from x to y") to be made, but not statements about the data held in cells (such as "x is a prime number"). We show that validity, even for this restricted language, is not r.e., but that the quantifier-free sublanguage is decidable. We then consider the complexity of model checking and validity for several fragments.

1 Introduction

This paper studies a recently developed an approach to reasoning about mutable data structures [9, 5]. The assertion language includes spatial conjunction and implication connectives alongside those of classical logic, in the style of the logic of Bunched Implications [8]. The conjunction $P * Q$ is true just when the current heap can be split into disjoint components, one of which makes P true and the other of which makes Q true. The implication $P \mathrel{-\!\!*} Q$ says that whenever P is true for a new or fresh piece of heap, Q is true for the combined new and old heap. In addition, there is an atomic formula, the points-to relation $E \mapsto F, G$, which says that E points to a cons cell holding F in its car and G in its cdr.

As a small example of $*$,

$$(x \mapsto a, y) * (y \mapsto b, x)$$

describes a two-element circular linked list, with a and b in the data fields. The conjunction $*$ here requires x and y to be pointers to distinct and non-overlapping cells. For an example of $\mathrel{-\!\!*}$,

$$(x \mapsto a, b) * ((x \mapsto c, b) \mathrel{-\!\!*} P)$$

says that x points to a cell holding (a, b), and that P will hold if we update the car to c.

The logic of [9, 5, 7] can be used to structure arguments in a way that leads to pleasantly simple proofs of pointer algorithms. But the assertion language that the logic uses to describe pre and postconditions is itself new, and its properties have not been studied in detail. The purpose of this paper is to study computability and complexity problems for the language.

We consider a pared down sublanguage, which includes the points-to relation and equality as atomic predicates, but not arithmetic or other expressions or atomic predicates for describing data. We do this to separate out questions about the shapes of data structures themselves from properties of the data held in them. This also insulates us from decidability questions about the data. In our language we can write a formula that says that x points to a linked list with two nodes, but not a formula that says that the list is sorted.

Our first result is that, even with these restrictions, the question of validity is not r.e. The spatial connectives are not needed for this negative result. This result might seem somewhat surprising, given the sparseness of the language; decidability would obtain immediately were we to omit the points-to relation. The proof goes by reduction from a well-known non-r.e. problem of finite model theory: deciding whether a closed first-order logic formula holds for all nonempty *finite* structures.

This result has two consequences. The first is that it tells us that we cannot hope to find an axiomatic description of $\mapsto$, adequate to the whole language. The second is that we should look to sublanguages if we are to find a decidability result.

Our second result is that the quantifier-free sublanguage is decidable. The main subtlety in the proof is the treatment of $-\!\!*$, whose semantics uses a universal quantification over heaps. This is dealt with by a bounding result, which restricts the number of heaps that have to be considered to verify or falsify a formula.

We then consider the complexity of model checking and validity. For the quantifier-free fragment and several sublanguages both questions are shown to be PSPACE-complete. One fragment is described where the former is NP-complete and the latter Π_2^{P}-complete. We also remark on cases where (like in propositional calculus) model checking is linear and validity coNP-complete.

2 The Model and the Assertion Language

In this section we present a spatial assertion language and its semantics. The other sections study properties of fragments of this language.

Throughout the paper we will use the following notation. A finite map f from X to Y is written $f\colon X \rightharpoonup_{fin} Y$, and $dom(f)$ indicates the domain of f. The notation $f\#g$ means that f and g have disjoint domains, and in that case $f * g$ is defined by $(f * g)(x) = y$ iff $f(x) = y$ or $g(x) = y$.

The syntax of expressions E and assertions P for binary heap cells is given by the following grammar:

$$
\begin{aligned}
E &::= x, y \ldots \mid \mathsf{nil} \\
P &::= (E \mapsto E, E) \mid E = E \mid \mathsf{false} \mid P \Rightarrow P \mid \forall x.\, P \mid \mathsf{emp} \mid P * P \mid P -\!\!* P
\end{aligned}
$$

Expressions are either variables or the constant nil. Assertions include equality, usual connectives from first-order classical logic, and spatial connectives. The predicate $(E \mapsto E_1, E_2)$ asserts that E is the only allocated cell and it points to a binary heap cell containing E_i in the i-th component. The assertion emp says that the heap is empty. The assertion $P_1 * P_2$ means that it is possible to split the current heap in disjoint sub-heaps making the two assertions true. The assertion $P_1 \mathrel{-\!\!*} P_2$ means that for each new heap disjoint from the current one and making P_1 true, the combined new and old heap makes P_2 true.

The other logical connectives are expressible as usual as derived notation:

$$\neg P \triangleq P \Rightarrow \mathsf{false} \qquad P_1 \wedge P_2 \triangleq \neg(P_1 \Rightarrow \neg P_2) \qquad \exists x. P \triangleq \neg(\forall x. \neg P)$$

Expressions and assertions for binary heap cells are interpreted in the following model:

$$\begin{aligned} Val &\triangleq Loc \cup \{nil\} \\ Stack &\triangleq Var \to Val \\ Heap &\triangleq Loc \rightharpoonup_{fin} Val \times Val \\ State &\triangleq Stack \times Heap \end{aligned}$$

Values are either locations or nil, and a state is composed of a stack and a heap. The heap is a finite map from locations to binary heap cells, whose domain indicates the locations that are allocated at the moment. The semantics of expressions and assertions is given in Table 1.

Definition 1 (Validity). *We say that P is valid, written $\models P$, if $s, h \models P$ for all the states (s, h).*

$$\begin{aligned} [\![x]\!]s &\triangleq s(x) \\ [\![\mathsf{nil}]\!]s &\triangleq nil \end{aligned}$$

$$\begin{array}{ll} s, h \models (E \mapsto E_1, E_2) & \text{iff } dom(h) = \{[\![E]\!]s\} \text{ and } h([\![E]\!]s) = ([\![E_1]\!]s, [\![E_2]\!]s) \\ s, h \models E_1 = E_2 & \text{iff } [\![E_1]\!]s = [\![E_2]\!]s \\ s, h \models \mathsf{false} & \quad \text{never} \\ s, h \models P_1 \Rightarrow P_2 & \text{iff if } s, h \models P_1 \text{ then } s, h \models P_2 \\ s, h \models \mathsf{emp} & \text{iff } dom(h) = \emptyset \\ s, h \models P_1 * P_2 & \text{iff there exist } h_1 \text{ and } h_2 \text{ such that} \\ & \qquad h_1 \# h_2;\ h_1 * h_2 = h;\ s, h_1 \models P_1;\ s, h_2 \models P_2 \\ s, h \models P_1 \mathrel{-\!\!*} P_2 & \text{iff for all } h_1 \text{ such that } h \# h_1 \text{ and } (s, h_1) \models P_1, \\ & \qquad (s, h * h_1) \models P_2 \\ s, h \models \forall x. P & \text{iff for any } v \text{ in } Val, s[x \mapsto v], h \models P \end{array}$$

Table 1. Semantics of Expressions and Assertions

3 Undecidability

The main result in this section is that the validity problem is not recursively enumerable even when the spatial connectives, $*$, emp and $\mathrel{-\!\!*}$, do not appear in assertions.

Theorem 1. *Deciding whether an assertion is valid is not recursively enumerable even when the assertion language is restricted as follows:*

$$P ::= (E \hookrightarrow E, E) \mid E = E \mid \mathsf{false} \mid P \Rightarrow P \mid \forall x.\, P$$

where $(E \hookrightarrow E_1, E_2)$ *is* $(E \mapsto E_1, E_2) * \mathsf{true}$.

Note that the theorem uses an intuitionistic variant $\hookrightarrow$ of the predicate $\mapsto$ because only with the $\mapsto$ predicate, we can not express that a heap cell l is allocated and contains (v_1, v_2) without requiring that l is the only allocated heap cell. The meaning of $(E \hookrightarrow E_1, E_2)$ is that a heap cell E is allocated and contains (E_1, E_2) but it need not be the only allocated cell.

We prove the theorem by reducing the validity on nonempty finite structures of closed first-order logic formulas to validity for our restricted language. Then, the conclusion follows by a standard result from finite model theory [3]:

Theorem 2 (Trakhtenbrot). *Even if a signature consists only of one binary relation, the set of closed first-order logic formulas valid on all nonempty finite structures is not recursively enumerable.*

The reduction goes by translating a first-order logic formula with a single binary relation R to an assertion. Let φ be a first-order logic formula, which is not necessarily closed. The translation $rd(\varphi)$ is given as follows:

$$rd(\varphi) \triangleq (\exists x.\, (x \hookrightarrow \mathsf{nil}, \mathsf{nil})) \Rightarrow prd(\varphi)$$

$$\begin{aligned}
prd(R(x,y)) &\triangleq (\exists z.\, (z \hookrightarrow x, y)) \wedge (x \hookrightarrow \mathsf{nil}, \mathsf{nil}) \wedge (y \hookrightarrow \mathsf{nil}, \mathsf{nil}) \\
prd(\varphi \Rightarrow \psi) &\triangleq prd(\varphi) \Rightarrow prd(\psi) \\
prd(\mathsf{false}) &\triangleq \mathsf{false} \\
prd(x = y) &\triangleq (x = y) \wedge (x \hookrightarrow \mathsf{nil}, \mathsf{nil}) \\
prd(\exists x.\, \varphi) &\triangleq \exists x.\, ((x \hookrightarrow \mathsf{nil}, \mathsf{nil}) \wedge prd(\varphi))
\end{aligned}$$

Intuitively, the translation encodes the relation $R^{\mathcal{A}}$ and the universe $|\mathcal{A}|$ of a nonempty finite structure $\mathcal{A}$ by heap cells: each element in $|\mathcal{A}|$ is encoded as an allocated cell containing (nil, nil), and a related pair (a_1, a_2) in $R^{\mathcal{A}}$ is encoded as an allocated cell containing (x, y) where x and y are encodings of a_1 and a_2, respectively. Note that the guard $(\exists x.\, (x \hookrightarrow \mathsf{nil}, \mathsf{nil}))$ in the definition of $rd(\varphi)$ models the fact that the universe of a finite structure must be nonempty.

The reduction becomes complete once we prove that for *closed* first-order logic formulas, the translation preserves and reflects validity. To show that validity is reflected, we prove a lemma which implies that for all nonempty finite structures $\mathcal{A}$ and environments η (mapping variables to elements of $|\mathcal{A}|$), it is always possible to find a state (s,h) so that

$$\mathcal{A},\eta \models \varphi \iff (s,h) \models rd(\varphi) \quad \text{for all closed first-order formulas } \varphi.$$

Lemma 1. *Let $\mathcal{A}$ be a nonempty finite structure for the signature $\{R\}$, where R is a binary relation. For all heaps h and sets B, C of locations such that*

- *$\{B, C\}$ is a partition of $dom(h)$;*
- *γ is a bijection from $|\mathcal{A}|$ to B such that $h(\gamma(a)) = (nil, nil)$ for all $a \in |\mathcal{A}|$; and*
- *δ is a bijection from $R^{\mathcal{A}}$ to C such that $h(\delta(a_1,a_2)) = (\gamma(a_1),\gamma(a_2))$ for all $(a_1,a_2) \in R^{\mathcal{A}}$,*

we have

$$\mathcal{A},\eta \models \varphi \iff \gamma \circ \eta, h \models rd(\varphi)$$

for all first-order formulas φ and environments η.

Proof. Since the universe $|\mathcal{A}|$ is not empty, the guard $(\exists x.\,(x \hookrightarrow \mathsf{nil},\mathsf{nil}))$ holds for the state (s,h). So, it suffices to prove the following claim: for all first-order formulas φ,

$$\mathcal{A},\eta \models \varphi \iff \gamma \circ \eta, h \models prd(\varphi)$$

It is straightforward to show the claim using induction over the structure of φ. □

Before showing that validity is preserved by the translation, we note that when φ is closed, so is $rd(\varphi)$; so, $rd(\varphi)$ is valid if and only if for all heaps h, there is some stack s with $(s,h) \models rd(\varphi)$. Let φ be a closed first-order formula and let h be a heap. When h does not have any cells containing (nil, nil), the guard $(\exists x.\,(x \hookrightarrow nil, nil))$ of $rd(\varphi)$ always becomes false; consequently, $(s,h) \models rd(\varphi)$ for all stacks s. The key idea to handle the other case, where a heap h has at least one cell containing (nil, nil), is to build a nonempty finite structure $\mathcal{A}$ and a stack s such that φ holds in $\mathcal{A}$ if and only if $(s,h) \models rd(\varphi)$. We construct such a stack simply by mapping all variables to the address of allocated cells in h containing (nil, nil); then, the following lemma shows how to construct the needed structure.

Lemma 2. *For all heaps h and stacks s such that $h(s(x))$ is defined and equal to (nil, nil) for all variables x, let $\mathcal{A}$ be a structure for the signature $\{R\}$ given by:*

- *$|\mathcal{A}| = \{l \in dom(h) \mid h(l) = (nil, nil)\}$; and*
- *$(l_1,l_2) \in R^{\mathcal{A}}$ iff l_1, l_2 are in $|\mathcal{A}|$ and $h(l) = (l_1,l_2)$ for some $l \in dom(h)$.*

Then, $\mathcal{A}, s \models \varphi$ iff $s, h \models rd(\varphi)$.

Proof. Note that the structure $|\mathcal{A}|$ can not be empty because in h, at least a single allocated heap cell must contain (nil, nil); otherwise, no stack would satisfy the condition in the lemma. One consequence of this fact is that $(s,h) \models rd(\varphi)$ iff $(s,h) \models prd(\varphi)$. So, it suffices to show that

$$\mathcal{A}, s \models \varphi \Longleftrightarrow s, h \models prd(\varphi),$$

which can be easily proved using induction over φ. □

4 Decidable Fragment

The undecidability result in the previous section indicates that in order to obtain a decidable fragment of the assertion language, either quantifiers must be taken out in the fragment or they should be used in a restricted manner. In this section, we consider the quantifier-free fragment of the assertion language, including spatial connectives, emp, $*$ and $-\!\!*$. The main result in the section is:

Theorem 3. *Deciding the validity of assertions is algorithmically decidable as long as the assertions are instances of the following grammar:*

$$P ::= (E \mapsto E, E) \mid E = E \mid \mathsf{false} \mid P \Rightarrow P \mid \mathsf{emp} \mid P * P \mid P -\!\!* P$$

To prove the theorem, we need to show that there is an algorithm which takes an assertion following the grammar in the theorem and answers whether the assertion holds for all states. The main observation is that each assertion determines a finite set of states so that if the assertion holds for all states in the set, it indeed holds for all the states. The proof proceeds in two steps: first we consider the case that an assertion P and a state s, h are given so that an algorithm is supposed to answer whether $s, h \models P$; then, we construct an algorithm which, given an assertion P, answers whether $s, h \models P$ holds for all the states (s, h). In the remainder of the section, we assume that all the assertions follow the grammar given in Theorem 3.

The problem of algorithmically deciding whether $s, h \models P$ holds given P, s, h as inputs is not as straightforward as it seems because of $-\!\!*$: when P is of the form $Q -\!\!* R$, the interpretation of $s, h \models P$ involves quantification over all heaps, which might require to check infinite possibilities. So, the decidability proof is mainly for showing that there is a finite boundary algorithmically determined by Q and R. We first define the *size* of an assertion, which is used to give an algorithm to determine the boundary.

Definition 2 (size of P). *For an assertion P, we define* size *of P, $|P|$, as follows:*

$$\begin{array}{rcl@{\qquad}rcl}
|(E \mapsto E_1, E_2)| &=& 1 & |E_1 = E_2| &=& 0 \\
|\mathsf{false}| &=& 0 & |P \Rightarrow Q| &=& max(|P|, |Q|) \\
|P * Q| &=& |P| + |Q| & |P -\!\!* Q| &=& |Q| \\
|\mathsf{emp}| &=& 1 & & &
\end{array}$$

The size of P determines a bound on the number of heap cells we have to consider to determine whether P is true or not. For instance, the size of $(x \mapsto \mathsf{nil}, \mathsf{nil}) * (y \mapsto \mathsf{nil}, \mathsf{nil})$ is 2; and to decide whether it or its negation is true, or whether it is merely satisfiable, requires us only to look at heaps of size upto 2.

The following proposition claims that there is a bound number of heaps to check in the interpretation of $s, h \models Q \mathrel{-\!\!*} R$; the decidability result is just an immediate corollary. Let ord be an effective enumeration of Loc.

Proposition 1. *Given a state (s, h) and assertions Q, R, let X be $FV(Q) \cup FV(R)$ and B a finite set consisting of the first $max(|Q|, |R|)$ locations in $Loc - (dom(h) \cup s(X))$ where the ordering is given by ord. Pick a value $v \in Val - s(X) - \{nil\}$. Then, $(s, h) \models Q \mathrel{-\!\!*} R$ holds iff for all h_1 such that*

- *$h \# h_1$ and $(s, h_1) \models Q$;*
- *$dom(h_1) \subseteq B \cup s(X)$; and*
- *for all $l \in dom(h_1)$, $h_1(l) \in (s(X) \cup \{nil, v\}) \times (s(X) \cup \{nil, v\})$*

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

To see why the proposition implies the decidability result, notice that there are only finitely many h_1's satisfying the conditions because both $B \cup s(X)$ and $s(X) \cup \{nil, v\}$ are finite. Since all the other cases of P only involve finitely many ways to satisfy $s, h \models P$, the exhaustive search gives the decision algorithm.

The interesting direction of the proposition is "if" because the only-if direction follows from the interpretation of $\mathrel{-\!\!*}$. Intuitively, the if direction of the proposition holds because the following three changes of heap cells do not affect the truth of either Q or R: relocating "garbage" heap cells (those not in $s(X)$); de-allocating redundant garbage heap cells when there are more than $max(|Q|, |R|)$ of them; overwriting "uninteresting values" (those not in $s(X) \cup \{nil\}$) by another uninteresting value (v). Then, for every heap h'_1 with $h \# h'_1$, there is a sequence of such changes which transforms h'_1 and $h * h'_1$ to h_1 and $h * h_1$, respectively, such that h_1 satisfies the last two conditions in the proposition. The proposition follows because each step in the sequence preserves the truth of both Q and R; so, $(s, h'_1) \models Q$ implies $(s, h_1) \models Q$, and $(s, h_1 * h) \models R$ implies $(s, h'_1 * h) \models R$.

Corollary 1. *Given a stack s and an assertion P, checking $(s, h) \models P$ for all h is decidable.*

Proof. The corollary holds because $s, h \models P$ for all h iff $s, [] \models (\neg P) \mathrel{-\!\!*} \mathsf{false}$. □

For the decidability of checking $(s, h) \models P$ for all states (s, h), we observe that the actual values of variables are not relevant to the truth of an assertion as long as the "relationship" of the values remains the same. We define a relation $\approx_X$ to capture this "relationship" formally. Intuitively, two states are related by $\approx_X$ iff the relationship of the values, which are stored in variables in X or in heap cells, are the same in the two states.

Definition 3 **($\approx_X$).** *For states (s, h) and (s', h') and a subset X of Var, $(s, h) \approx_X (s', h')$ iff there exists a bijection r from Val to Val such that $r(nil) = nil$; $r(s(x)) = s'(x)$ for all $x \in X$; and $(r \times r)(h(l)) = h'(r(l))$ for all $l \in Loc$.* [1]

Proposition 2. *For all the states (s, h) and (s', h') and all assertions P such that $(s, h) \approx_{FV(P)} (s', h')$, if $(s, h) \models P$, then $(s', h') \models P$.*

Lemma 3. *Given a state (s, h) and an assertion P, let B be the set consisting of the first $|FV(P)|$ locations in Loc, where the ordering is given by* ord. *Then, there exists a state (s', h') such that $s'(Var - FV(P)) \subseteq \{nil\}$; $s'(FV(P)) \subseteq B \cup \{nil\}$; and $(s, h) \approx_{FV(P)} (s', h')$.*

The decidability result follows from the above lemma. To see the reason, we note that because of the lemma, for all assertions P, there is a finite set of stacks such that if for all stacks s in the set and all heaps h, $(s, h) \models P$, then P holds for all states whose stack is not necessarily in the set. Therefore, a decision algorithm is obtained by exhaustively checking for each stack s in the finite set whether $(s, h) \models P$ holds for all heaps h using the algorithm in Corollary 1.

Corollary 2. *Given an assertion P, checking $(s, h) \models P$ for all the states (s, h) is decidable.*

5 Complexity

In this section we study the complexity of model checking for some fragments of the decidable logic of Section 4.

We consider the following fragments, where $(E \not\mapsto -)$ means that E is not allocated $(s, h \models (E \not\mapsto -)$ iff $[\![E]\!]s \notin dom(h))$:

Language		MC	VAL
$\mathcal{L}$	$P ::= (E \mapsto E, E) \mid (E \not\mapsto -) \mid E = E \mid E \neq E \mid \mathsf{false}$ $\mid P \wedge P \mid P \vee P \mid \mathsf{emp}$	P	coNP
$\mathcal{L}^*$	$P ::= \mathcal{L} \mid P * P$	NP	Π_2^{P}
$\mathcal{L}^{\neg *}$	$P ::= \mathcal{L} \mid \neg P \mid P * P$	PSPACE	PSPACE
$\mathcal{L}^{-\!*}$	$P ::= \mathcal{L} \mid P -\!\!* P$	PSPACE	PSPACE
$\mathcal{L}^{\neg * -\!*}$	$P ::= \mathcal{L} \mid \neg P \mid P * P \mid P -\!\!* P$	PSPACE	PSPACE

Given a fragment $\mathcal{L}^c$, the corresponding model-checking problem $MC(\mathcal{L}^c)$ is deciding whether $s, h \models P$ holds given a state (s, h) and an assertion $P \in \mathcal{L}^c$. The validity problem asks whether a formula is true in all states. In the above table the second-last column reports the complexity of model checking and the last the complexity of validity.

The easy fragment is $\mathcal{L}$. Clearly $MC(\mathcal{L})$ can be solved in linear time by the obvious algorithm arising from the semantic definitions, and it is not difficult to

[1] the equality in $(r \times r)(h(l)) = h'(r(l))$ means that if one side of the equation is defined, the other side is also defined and they are equal.

show that the validity is coNP-complete. As soon as we add $*$, model checking bumps up to NP-complete. The validity problem for $\mathcal{L}^*$ is Π_2^{P}-complete; we show the former but consideration of the latter is omitted for brevity. It is possible to retain linear model checking when $*$ is restricted so that one conjunct is of the form $(E \mapsto E_1, E_2)$. The fragment $\mathcal{L}^{\neg * -\!*}$ is the object of the decidability result of Section 4; a consequence of our results there is that model checking and validity can be decided in polynomial space. Below we show PSPACE-hardness for model checking for the two fragments $\mathcal{L}^{\neg *}$ and $\mathcal{L}^{-\!*}$. It is a short step to show PSPACE-hardness for validity.

5.1 $MC(\mathcal{L}^*)$ is NP-complete

In this section we show directly that $MC(\mathcal{L}^*)$ belongs to NP, and give a reduction from an NP-complete problem to it.

Proposition 3. *$MC(\mathcal{L}^*)$ is in NP.*

Proof. The only interesting part is deciding whether $s, h \models P * Q$ holds. The algorithm proceeds by choosing non-deterministically a set $D \subseteq dom(h)$, determining a splitting of h in two heaps h_1 and h_2 obtained by restricting h to D and to $dom(h) - D$ respectively. □

Definition 4. *The problem SAT is, given a formula F from the grammar*

$$F ::= x \mid \neg x \mid F \wedge F \mid F \vee F$$

deciding whether it is satisfiable, i.e. whether there exists an assignment of boolean values to the free variables of F making F true.

Definition 5. *The translation from formulas F to assertions P of $\mathcal{L}^*$ is defined by a function $tr(-)$:*[2]

$$tr(x) \stackrel{\Delta}{=} (x \mapsto \mathsf{nil}, \mathsf{nil}) * \mathsf{true} \qquad tr(\neg x) \stackrel{\Delta}{=} (x \not\mapsto -)$$
$$tr(F_1 \wedge F_2) \stackrel{\Delta}{=} tr(F_1) \wedge tr(F_2) \qquad tr(F_1 \vee F_2) \stackrel{\Delta}{=} tr(F_1) \vee tr(F_2)$$

Proposition 4. *A formula F with variables $\{x_1, \ldots, x_n\}$ is satisfiable if and only if $s_0, h_0 \models tr(F) * \mathsf{true}$ holds, where s_0 maps distinct variables x_i to distinct locations l_i, $dom(h_0) = \{l_1, \ldots, l_n\}$ and $h_0(l_i) = (nil, nil)$ for $i = 1, \ldots, n$.*

Proof. The truth of a boolean variable x is represented by its being allocated; in the initial state (s_0, h_0) all the variables are allocated. The formula $tr(F) * \mathsf{true}$ is true if and only if there exists a subheap h' making $tr(F)$ true, and subheaps correspond to assignments of boolean values to the variables in F. □

Since the translation and construction of (s_0, h_0) can be performed in polynomial time, an immediate consequence is NP-hardness of $MC(\mathcal{L}^*)$, hence NP-completeness.

[2] true can be expressed by $\mathsf{nil} = \mathsf{nil}$ in $\mathcal{L}^*$.

5.2 $MC(\mathcal{L}^{\neg *})$ is PSPACE-complete

In this section PSPACE-hardness of $MC(\mathcal{L}^{\neg *})$ is proved by reducing a PSPACE-complete problem to it. Completeness follows from the fact that $MC(\mathcal{L}^{\neg * -\!*})$ is in PSPACE and that $\mathcal{L}^{\neg *}$ is a sub-fragment of $\mathcal{L}^{\neg * -\!*}$.

Definition 6. *The problem QSAT is, given a closed formula G from the grammar*

$$F ::= x \mid \neg x \mid F \wedge F \mid F \vee F, \qquad G ::= \forall x_1.\exists y_1.\dots.\forall x_n.\exists y_n.F$$

deciding whether it is true.

Definition 7. *The translation from formulas G to assertions P of $\mathcal{L}^{\neg *}$ is defined by a function $tr(-)$:*

$$\begin{array}{rlrl} tr(x) & \triangleq (x \mapsto \mathsf{nil},\mathsf{nil}) * \mathsf{true} & tr(\neg x) & \triangleq (x \not\mapsto -) \\ tr(F_1 \wedge F_2) & \triangleq tr(F_1) \wedge tr(F_2) & tr(F_1 \vee F_2) & \triangleq tr(F_1) \vee tr(F_2) \\ tr(\exists y_i.G) & \triangleq ((y_i \mapsto \mathsf{nil},\mathsf{nil}) \vee \mathsf{emp}) * tr(G) & & \\ tr(\forall x_i.G) & \triangleq \neg\,(((x_i \mapsto \mathsf{nil},\mathsf{nil}) \vee \mathsf{emp}) * \neg\, tr(G)) & & \end{array}$$

Proposition 5. *A closed formula G is true if and only if $s_0, h_0 \models tr(G)$ holds, where s_0 maps distinct variables x_i to distinct locations l_i, $dom(h_0) = \{l_1, \dots, l_n\}$ and $h_0(l_i) = (nil, nil)$ for $i = 1, \dots, n$.*

Proof. The truth of a boolean variable x is represented by its being allocated; in the initial state (s_0, h_0) all the variables are allocated. The only interesting cases are the quantifiers. The invariant is that $tr(\exists y_i.G)$ is checked in a state where y_i is allocated, thus $((y_i \mapsto \mathsf{nil},\mathsf{nil}) \vee \mathsf{emp}) * tr(G)$ holds iff $tr(G)$ holds either for the current state or for the state obtained by de-allocating y_i. In other words, G either holds for y_i true or for y_i false. The translation of $(\forall x_i.-)$ is essentially $\neg(\exists x_i.\neg -)$. □

Observing that the translation and construction of (s_0, h_0) can be performed in polynomial time, we have shown PSPACE-hardness of $MC(\mathcal{L}^{\neg *})$.

5.3 $MC(\mathcal{L}^{-\!*})$ is PSPACE-complete

In analogy with the previous section, a translation from QSAT to $MC(\mathcal{L}^{-\!*})$ is presented.

This case is more complicated, since $-\!*$ provides a natural way of representing universal quantifiers, but there is no immediate way to represent existentials. Our solution is to use two variables x_t and x_f to represent a boolean variable x. There are three admissible states:

- initial, when neither x_t nor x_f is allocated;
- true, when x_t is allocated and x_f is not;
- false, when x_f is allocated and x_t is not.

We use some auxiliary predicates:

$$
\begin{aligned}
(x \hookrightarrow -) &\triangleq ((x \mapsto \mathsf{nil}, \mathsf{nil}) {-\!\!*}\, \mathsf{false}) \wedge (x \neq \mathsf{nil}) \\
I_x &\triangleq (x_t \not\hookrightarrow -) \wedge (x_f \not\hookrightarrow -) \\
OK_x &\triangleq ((x_t \hookrightarrow -) \wedge (x_f \not\hookrightarrow -)) \vee ((x_f \hookrightarrow -) \wedge (x_t \not\hookrightarrow -))
\end{aligned}
$$

The meaning of $(x \hookrightarrow -)$ is that it is not possible to extend the current heap with x pointing to $(\mathsf{nil}, \mathsf{nil})$, i.e. x is allocated; I_x means that x is in an initial state, and OK_x means that x is either in state true or in state false.

Definition 8. *Given a closed formula $\forall x_1.\exists y_1.\ldots.\forall x_n.\exists y_n.F$, define the ordered set $V \triangleq \{x_1 < y_1 < \ldots < x_n < y_n\}$. Write S^{op} for $V - S$ when $S \subseteq V$. Define $\{\leq x\} \triangleq \{x' \in V | x' \leq x\}$. The predicates are extended as follows:*

$$
I_S \triangleq \bigwedge_{x \in S} I_x \qquad\qquad OK_S \triangleq \bigwedge_{x \in S} OK_x
$$

The translation is defined by a function $tr(-)$:

$$
\begin{aligned}
tr(x) &\triangleq (x_t \hookrightarrow -) \\
tr(\neg x) &\triangleq (x_f \hookrightarrow -) \\
tr(F_1 \wedge F_2) &\triangleq tr(F_1) \wedge tr(F_2) \\
tr(F_1 \vee F_2) &\triangleq tr(F_1) \vee tr(F_2) \\
tr(\forall x_i.G) &\triangleq (OK_{\{x_i\}} \wedge I_{\{x_i\}^{op}}) {-\!\!*}\, tr(G) \\
tr(\exists y_i.G) &\triangleq \sim((OK_{\{\leq x_i\}} \wedge I_{\{\geq y_i\}}) \wedge \sim(OK_{\{\leq y_i\}} \wedge I_{\{\geq x_{i+1}\}} \wedge tr(G)))
\end{aligned}
$$

where $\sim P$ is short for $P {-\!\!*}\, \mathsf{false}$.

Intuitively, the translation of x says that x is in state true, and the translation of $\neg x$ says that x is in state false. For $(\forall x_i.G)$, the invariant is that $OK_{\{\leq y_{i-1}\}}$ and $I_{\geq x_i}$ hold, and the translation says that G holds after extending the current heap with any new heap containing only x_i in an OK state (i.e. true or false). For $(\exists y_i.G)$, the invariant is that $OK_{\{\leq x_i\}}$ and $I_{\geq y_i}$ hold. The formula $\sim(P \wedge Q)$ implies that when P holds in a new heap, the heap does not satisfy Q; in particular, if P is $OK_{\{\leq x_i\}} \wedge I_{\{\geq y_i\}}$ and P holds in the current heap, $\sim(P \wedge Q)$ implies that inverting the boolean value of x makes Q false. This case is the most complicated of the translation, and involves a double negation. In words, it says that given an initial heap h_0, inverting the boolean values of variables in $\{\leq x_i\}$ leads to a heap h_1 which makes the following false: for every heap h_2 obtained from h_1 by inverting again the boolean values of variables in $\{\leq x_i\}$ and by assigning some boolean value to y_i, G does not hold in h_2.

Proposition 6. *A closed formula G is true if and only if $s_0, [] \models tr(G)$ holds, where s_0 maps distinct variables x_i to distinct locations l_i, and $[]$ is the empty heap.*

To obtain the PSPACE-hardness result, observe that the translation can be performed in polynomial time, since the size of each I_S and OK_S is linear in the number of variables.

6 Future Work

Possible directions for future work include incorporating heap variables that allow us to take snapshots of the heap, with suitable restrictions [6] to maintain decidability, and also recursive definitions or special atomic predicates [1] for describing paths through the heap. We also plan to investigate the relation of our approach to work on model checking mobile ambients [2]. Finally, it would be useful to integrate our results on counter-models with the recently developed tableaux proof theory for Bunched Implications [4].

Acknowledgments

We would like to thank the anonymous referees for their valuable comments that suggested interesting improvements to the paper. Yang was supported by the US NSF under grant INT-9813854 and by Creative Research Initiatives of the Korean Ministry of Science and Technology. Calcagno and O'Hearn were supported by the EPSRC under the "Local Reasoning about State" projects.

References

1. M. Benedikt, T. Reps, and M. Sagiv. A decidable logic for describing linked data structures. In *ESOP '99: European Symposium on Programming*, pages 2–19. Lecture Notes in Computer Science, Vol. 1576, S.D. Swierstra (ed.), Springer-Verlag, New York, NY, 1999.
2. W. Charatonik, S. Dal Zilio, A. Gordon, S. Mukhopadhyay, and J.M. Talbot. The complexity of model checking mobile ambients. In *FoSSaCS*, April 2001.
3. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995. ISBN 3-540-60149-X.
4. D. Galmiche and D. Méry. Proof-search and countermodel generation in propositional BI logic. In *TACS*, 2001. LNCS to appear.
5. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Principles of Programming Languages*, January 2001.
6. J. Jenson, M. Jorgensen, N. Klarkund, and M. Schwartzback. Automatic verification of pointer programs using monadic second-order logic. In *Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation*, pages 225–236, 1997. SIGPLAN Notices 32(5).
7. P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *Proceedings of 15th Annual Conference of the European Association for Computer Science Logic: CSL 2001*, pages 1–19. Springer-Verlag, 2001. LNCS 2142.
8. P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 99.
9. J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*. Palgrave, 2000.