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] University of Birmingham and University of Illinois at Urbana-Champaign

Abstract. Reynolds, Ishtiaq and O'Hearn have recently developed an approach to reasoning about mutable data structures using an assertion language with spatial conjunction and implication connectives. In this paper we study 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

Reynolds, Ishtiaq and O'Hearn have recently developed an approach to reasoning about mutable data structures using an assertion language with spatial connectives [6, 4]. 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 [6, 4] 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 of the co-halting problem for a two-stack machine. It hinges on the ability to describe list-like structures, even without the aid of recursive definitions.

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 decidabilty 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 $f : X \rightharpoonup_{fin} Y$, 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 $g(x) = y$.

For a natural number $n \geq 2$, the syntax of expressions E and assertions P for n-ary heap cells is given by the following grammar:

$$E ::= x, y \ldots \mid \mathsf{nil}$$
$$P ::= (E \mapsto E_1, \ldots, E_n) \mid E = E \mid \mathsf{false} \mid P \Rightarrow P \mid \forall x.\, P \mid \mathsf{emp} \mid P * P \mid P -\!\!* P$$

Expressions are either variables or the constant nil. Assertions include equality, usual connectives from first-order classical logic, and spatial connectives. The predicate $(E \mapsto E_1, \ldots, E_n)$ asserts that E is the only allocated cell and it points to a n-ary 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.

Expressions and assertions for n-ary heap cells are interpreted in the following model:

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

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 n-ary 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 (Semantic Consequence). *We say that Q is a semantic consequence of P, written $P \models Q$, if $s, h \models P$ implies $s, h \models Q$ for all the states (s, h).*

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

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

Table 1. Semantics of Expressions and Assertions

For the sake of simplicity, Section 3 assumes that each heap cell has five fields, and Section 4 and 5 consider binary heap cells. However, the results in the paper are equally applicable to other cases as long as each heap cell is assumed to have no less than 2 fields.

3 Undecidability

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

Theorem 1. *Deciding semantic consequence in the assertion language is not recursively enumerable even when assertions are restricted by the following context free grammar:*

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

where $(E \hookrightarrow E_1, E_2, E_3, E_4, E_5)$ *is* $(E \mapsto E_1, E_2, E_3, E_4, E_5) * \mathsf{true}$.

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 $(l_1, \ldots, l_5)$ without requiring that l is the only allocated heap cell. The meaning of $(E \hookrightarrow E_1, \ldots, E_5)$ is that a heap cell E is allocated and contains $(E_1, \ldots, E_5)$ but it need not be the only allocated cell.

We prove the theorem by reducing the co-halting problem of a two stack machine to the problem of deciding semantic consequence. Then, the conclusion follows because the co-halting problem of a two stack machine is not recursively enumerable. In the proof, we use two kinds of variables, variables in the assertion language and *meta* variables, which range over states of the two-stack machine and assertions including even variables in the assertion language. To avoid confusion, variables are written in the sans serif font, such as x, y, and meta variables in the italic font, such as x, y, in the section.

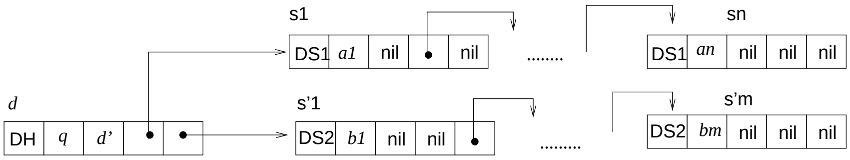
Consider a two stack machine, whose alphabet is $\{0, 1\}$ and set of states is $\{Q_0, \ldots, Q_e\}$ with Q_0 the initial state and Q_e the unique final state. We pick up $e+6$ distinct variables t, f, q_0, …, q_e, DH, $\mathsf{DS1}$ and $\mathsf{DS2}$, which are supposed to hold different values. Variables t and f represent elements in the alphabet, 1 and 0, respectively. And q_i represents the state Q_i. The remaining three variables are used to denote a role that each heap cell plays in the encoding: when a cell is tagged with DH, it is used as a head node for encoding a snapshot of the machine during execution; and when a cell is tagged with $\mathsf{DS}n$, it is used to encode the content of the n-th stack. When we use a meta variable to denote a state or a symbol in the alphabet, the same meta variable is also used to denote the corresponding variable. For instance, a meta variable a denoting 0 is also used to denote the variable f.

We first encode a status of the machine during execution. Let $(q, a_1 \ldots a_n, b_1 \ldots b_m)$ be an instantaneous description of a two-stack machine, where q denotes the current state, and $a_1 \ldots a_n$ and $b_1 \ldots b_m$ are the current contents of the two stacks with a_1 and b_1 their top elements. Suppose that d' denotes the head node of the encoding of an instantaneous description at the previous step; if there is no previous step, d' denotes nil. We encode this fact — $(q, a_1 \ldots a_n, b_1 \ldots b_m)$ is the current status of the machine with d' as its previous status — by the following assertion:

$$\begin{aligned}
&Desc(d', d, (q, a_1 \ldots a_n, b_1 \ldots b_m)) \stackrel{\Delta}{=} \\
&\quad \exists \mathsf{s}_1, \ldots, \mathsf{s}_{n+1}, \mathsf{s}'_1, \ldots, \mathsf{s}'_{m+1}. \\
&\qquad (d' \neq d) \wedge (d \hookrightarrow \mathsf{DH}, q, d', \mathsf{s}_1, \mathsf{s}'_1) \\
&\qquad \wedge \left(\textstyle\bigwedge_{1 \leq j \leq n} (\mathsf{s}_j \hookrightarrow \mathsf{DS1}, a_j, \mathsf{nil}, \mathsf{s}_{j+1}, \mathsf{nil})\right) \wedge (\mathsf{s}_{n+1} = \mathsf{nil}) \\
&\qquad \wedge \left(\textstyle\bigwedge_{1 \leq j \leq m} (\mathsf{s}'_j \hookrightarrow \mathsf{DS2}, b_j, \mathsf{nil}, \mathsf{nil}, \mathsf{s}'_{j+1})\right) \wedge (\mathsf{s}'_{m+1} = \mathsf{nil})
\end{aligned}$$

Recall that DH, $\mathsf{DS1}$ and $\mathsf{DS2}$ are assumed to denote different values. So, for a state satisfying the assertion, any two variables of $d, d', \mathsf{s}_1, \ldots, \mathsf{s}_{n+1}, \mathsf{s}'_1, \ldots, \mathsf{s}'_{m+1}$

must denote different values. The assertion indicates that the following structure of cells exists on the heap:

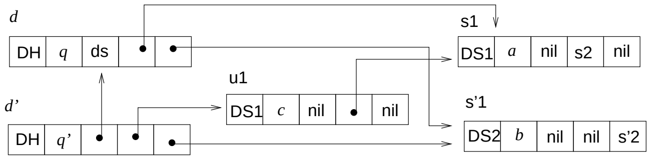


In the above picture, the first field of each heap cell denotes what role the cell plays in the encoding. The second field of the head node, which is pointed to by d in the picture, stores the current state q using the variable q_i corresponding to q. The third field of the head node points to either another head node encoding the instantaneous description just one step before or nil so indicating that the current head node describes the initial status of the machine. The last two fields in the head node denote two stacks, which are just two singly linked lists. Each node for two stacks uses the second field to store the contents of the stacks, which are sequences of t and f.

A transition t is encoded using aliased pointers. We only show the encoding of the case $t = (q, a, b) \to (q', ca, b)$; the other four cases can be handled similarly.

$$
\begin{array}{l}
\mathit{Trans}(d, d', (q, a, b) \xrightarrow{t} (q', ca, b)) \stackrel{\Delta}{=} \\
\exists \mathsf{ds}, \mathsf{s}_1, \mathsf{s}_2, \mathsf{s}'_1, \mathsf{s}'_2, \mathsf{u}_1. \\
(\mathsf{u}_1 \neq \mathsf{s}_1) \wedge (d \neq d') \wedge (d \hookrightarrow \mathsf{DH}, q, \mathsf{ds}, \mathsf{s}_1, \mathsf{s}'_1) \wedge (d' \hookrightarrow \mathsf{DH}, q', d, \mathsf{u}_1, \mathsf{s}'_1) \\
\wedge (\mathsf{s}'_1 \hookrightarrow \mathsf{DS2}, b, \mathsf{nil}, \mathsf{nil}, \mathsf{s}'_2) \wedge (\mathsf{s}_1 \hookrightarrow \mathsf{DS1}, a, \mathsf{nil}, \mathsf{s}_2, \mathsf{nil}) \wedge (\mathsf{u}_1 \hookrightarrow \mathsf{DS1}, c, \mathsf{nil}, \mathsf{s}_1, \mathsf{nil})
\end{array}
$$

Note that $d, \mathsf{s}_1, \mathsf{s}'_1, d', \mathsf{u}_1$ denote different locations. Therefore, the body part of the existential quantification indicates that the following structure of cells exists on the heap:



Notice also that the encoding specifies that d' is the next step of d by making the third field of d' point back to d.

Lemma 1. *Let δ be the set of transitions of the two-stack machine, and suppose that (q, α, β) is an instantaneous description of the machine. For all the states (s, h), if*

$$s, h \models \left(\bigvee_{t \in \delta} \mathit{Trans}(\mathsf{ds}_0, \mathsf{ds}_1, t) \right) \wedge \mathit{Desc}(\mathsf{ds}', \mathsf{ds}_0, (q, \alpha, \beta))$$

then there exist t and (q', α', β') such that

- *(q', α', β') is obtained from (q, α, β) by applying t; and*
- *$s, h \models \mathit{Desc}(\mathsf{ds}_0, \mathsf{ds}_1, (q', \alpha', \beta'))$.*

Proof. Let h be a heap satisfying the condition in the lemma. Then, by the interpretation of $\vee$, there exists a transition $t \in \delta$ such that

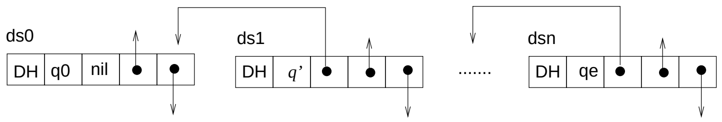
$$s, h \models \mathit{Trans}(\mathsf{ds}_0, \mathsf{ds}_1, t) \wedge \mathit{Desc}(\mathsf{ds}', \mathsf{ds}_0, (q, \alpha, \beta)).$$

Consider the case when $t = (q, a, b) \to (q', ca, b)$. Since $s, h \models \mathit{Trans}(\mathsf{ds}_0, \mathsf{ds}_1, t)$, the first stack of the instantaneous description encoded by nodes from ds_0 should contain a as a top element and the second stack b as a top element. So, we can rewrite α and β as $a\alpha_0$ and $b\beta_0$ for appropriate α_0 and β_0. Then, $(q', ca\alpha_0, b\beta_0)$ can be obtained from (q, α, β) by applying t. Then, the nodes from ds_1 encode the instantaneous description $(q', ca\alpha_0, b\beta_0)$ because of the definitions of $\mathit{Desc}(\mathsf{ds}_0, \mathsf{ds}_1, (q', ca\alpha_0, b\beta_0))$ and $\mathit{Trans}(\mathsf{ds}_0, \mathsf{ds}_1, (q, a, b) \to (q', ca, b))$. That is, we have

$$s, h \models \mathit{Desc}(\mathsf{ds}_0, \mathsf{ds}_1, (q', ca\alpha_0, b\beta_0)).$$

All the other cases can be handled similarly. □

The last step of the encoding is to handle a computation sequence. For this purpose, we've already made the third field of each head cell point back to a previous step. The following picture shows heap cells representing a terminating computation sequence:



For each instantaneous description which appears in the middle of an accepting computation, either the description denotes termination or there is another description for the next step. *Next* captures this. *Next* says that for each head node which denotes a computation step, either it is in the unique final state q_e or there is another head node for the next step.

$$\begin{aligned}\mathit{Next} \triangleq\ & \forall \mathsf{ds}_0, \mathsf{ds}', \mathsf{q}, \mathsf{s}_1, \mathsf{s}_1'.\\ & (\mathsf{ds}_0 \hookrightarrow \mathsf{DH}, \mathsf{q}, \mathsf{ds}', \mathsf{s}_1, \mathsf{s}_1') \Rightarrow (\mathsf{q} = \mathsf{q}_e) \vee \exists \mathsf{ds}_1.\, (\textstyle\bigvee_{t \in \delta} \mathit{Trans}(\mathsf{ds}_0, \mathsf{ds}_1, t))\end{aligned}$$

Notice the local nature of *Next*; they only mention each step as opposed to a whole sequence of computation, which is usually the case in the encodings with a list data type or a graph data type in [2, 3]. We need one more formula which says that all $e+6$ variables t, f, q_0, …, q_e, DH, $\mathsf{DS1}$, $\mathsf{DS2}$ denote distinct values:

$$\mathit{Disjoint} \triangleq \bigwedge_{v, w \in \{\mathsf{t}, \mathsf{f}, \mathsf{q}_0, \ldots, \mathsf{q}_e, \mathsf{DH}, \mathsf{DS1}, \mathsf{DS2}\}, v \not\equiv w} (v \neq w)$$

where we use $\equiv$ to mean that two variables are the same as symbols.

Lemma 2. *Let (q_0, α_0, β_0) be the initial instantaneous description of the two stack machine. For all the states (s, h), if*

$$\begin{aligned} s, h \models\ & \forall \mathsf{t}, \mathsf{f}, \mathsf{q}_0, \ldots, \mathsf{q}_e, \mathsf{DH}, \mathsf{DS1}, \mathsf{DS2}.\\ & \mathit{Disjoint} \Rightarrow \exists \mathsf{ds}_0.\, \mathit{Desc}(\mathsf{nil}, \mathsf{ds}_0, (q_0, \alpha_0, \beta_0)) \wedge \mathit{Next}\end{aligned}$$

then there exists n such that

$$
\begin{aligned}
s, h \models\ & \forall \mathsf{t}, \mathsf{f}, \mathsf{q}_0, \ldots, \mathsf{q}_e, \mathsf{DH}, \mathsf{DS1}, \mathsf{DS2}. \\
& \quad \mathit{Disjoint} \Rightarrow \\
& \qquad \exists \mathsf{ds}_0 \ldots \mathsf{ds}_n. (\textstyle\bigwedge_{0 \le i < n} (\bigvee_{t \in \delta} \mathit{Trans}(\mathsf{ds}_i, \mathsf{ds}_{i+1}, t))) \\
& \qquad\qquad \wedge \mathit{Desc}(\mathsf{nil}, \mathsf{ds}_0, (q_0, \alpha_0, \beta_0)) \wedge (\exists \mathsf{ds}', \mathsf{s}_1, \mathsf{s}_1'. (\mathsf{ds}_n \hookrightarrow \mathsf{DH}, \mathsf{q}_e, \mathsf{ds}', \mathsf{s}_1, \mathsf{s}_1'))
\end{aligned}
$$

Proof. Suppose a state s, h satisfies the formula in the assumption. Then, the heap must contain a head node l encoding the initial status of the machine (q_0, α_0, β_0). Since *Next* holds, it is possible to find a sequence $l_0, l_1, l_2, \ldots$ of head nodes such that $l_0 = l$, l_{i+1} denotes a next step of l_i in the machine (that is, $\bigvee_{t \in \delta} \mathit{Trans}(\mathsf{ds}_i, \mathsf{ds}_{i+1}, t)$ holds in a state $(s[\mathsf{ds}_i \mapsto l_i, \mathsf{ds}_{i+1} \mapsto l_{i+1}], h))$ and l_m is the end of the sequence only when its second field contains the final state q_e. So, it suffices to show that the sequence is finite. Suppose that the sequence is infinite. Since there are only finite number of allocated cells in h, the sequence must have a repetition. However, that can not become the case because the third link of each l_{i+1} points back to l_i and $l_0 = \mathsf{nil}$; note that a connected graph with n nodes and $n-1$ links can not have any cycle. □

Now the main fact in this encoding is the following:

Proposition 1. *The two stack machine accepts (q_0, α_0, β_0) iff there exists a state (s, h) such that*

$$
\begin{aligned}
s, h \models\ & \forall \mathsf{t}, \mathsf{f}, \mathsf{q}_0, \ldots, \mathsf{q}_e, \mathsf{DH}, \mathsf{DS1}, \mathsf{DS2}. \\
& \quad \mathit{Disjoint} \Rightarrow \exists \mathsf{ds}_0. \mathit{Desc}(\mathsf{nil}, \mathsf{ds}_0, (q_0, \alpha_0, \beta_0)) \wedge \mathit{Next}
\end{aligned}
$$

Proof. It is easy to see that the only-if direction holds because we can build a heap which only contains heap cells for encoding an accepting computation of (q_0, α_0, β_0). And Lemma 1 and 2 give the other direction. □

The following corollary shows the reduction.

Corollary 1. *The two stack machine does not accept (q_0, α_0, β_0) iff*

$$
\begin{pmatrix}
\forall \mathsf{t}, \mathsf{f}, \mathsf{q}_0, \ldots, \mathsf{q}_e, \mathsf{DH}, \mathsf{DS1}, \mathsf{DS2}. \\
\quad \mathit{Disjoint} \Rightarrow \exists \mathsf{ds}_0. \mathit{Desc}(\mathsf{nil}, \mathsf{ds}_0, (q_0, \alpha_0, \beta_0)) \wedge \mathit{Next}
\end{pmatrix} \models \mathsf{false}
$$

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 2. *Deciding semantic consequence in the assertion language is algorithmically decidable as long as the assertions are instances of the following grammar:*

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

The theorem is equivalent to the existence of an algorithm which takes an assertion following the grammar in the theorem and answers whether the assertion holds for all states. We show that such an algorithm exists. 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 2.

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:*

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

The size of P intuitively determines how many heap cells, which are not directly pointed to by variables in P, are relevant to the truth of P. For instance, the size of $(\neg\mathsf{emp}) * (\neg\mathsf{emp})$ is 2 and indicates that if at least two heap cells are already allocated on the heap, allocating more heap cells does not affect the truth of the assertion. Note that $(\neg\mathsf{emp}) * (\neg\mathsf{emp})$ holds for a state when at least two cells are allocated on the heap.

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

Proposition 2. *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, $(s, h) \models Q -\!\!* R$ 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 $-\!\!*$. 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 $(s, h'_1 * h) \models R$. The formal proof appears in the appendix.

Corollary 2. *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 $s, [] \models (\neg P) -\!\!* \mathsf{false}$. □

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 3. *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 2.

Corollary 3. *Given an assertion P, checking $(s, h) \models P$ for all the states (s, h) is decidable.*

[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.

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
$\mathcal{L}$	$P ::= (E \mapsto E, E) \mid (E \not\mapsto -) \mid E = E \mid \mathsf{false}$ $\mid P \wedge P \mid P \vee P \mid \mathsf{emp}$	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
$\mathcal{L}^{-*}$	$P ::= \mathcal{L} \mid P \mathrel{-\!\!*} P$	PSPACE	PSPACE
$\mathcal{L}^{\neg * -*}$	$P ::= \mathcal{L} \mid \neg P \mid P * P \mid P \mathrel{-\!\!*} P$	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 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 space reasons. 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 4. *$MC(\mathcal{L}^*)$ 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 $tr(-)$:*

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

Proposition 5. *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.

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*

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

deciding whether it is true.

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

$$tr(x) \stackrel{\Delta}{=} (x \mapsto \mathsf{nil}, \mathsf{nil}) * \mathsf{true} \qquad tr(\neg x) \stackrel{\Delta}{=} (x \not\mapsto -)$$
$$tr(F_1 \wedge F_2) \stackrel{\Delta}{=} tr(F_1) \wedge tr(F_2) \qquad tr(F_1 \vee F_2) \stackrel{\Delta}{=} tr(F_1) \vee tr(F_2)$$
$$tr(\exists y_i . G) \stackrel{\Delta}{=} ((y_i \mapsto \mathsf{nil}, \mathsf{nil}) \vee \mathsf{emp}) * tr(G)$$
$$tr(\forall x_i . G) \stackrel{\Delta}{=} \neg\, (((x_i \mapsto \mathsf{nil}, \mathsf{nil}) \vee \mathsf{emp}) * \neg\, tr(G))$$

Proposition 6. *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, $dom(h_0) = \{l_1, \ldots, l_n\}$ 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 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 $tr(-)$:

$$
\begin{aligned}
tr(x) &\triangleq (x_t \hookrightarrow -) & tr(\neg x) &\triangleq (x_f \hookrightarrow -) \\
tr(F_1 \wedge F_2) &\triangleq tr(F_1) \wedge tr(F_2) & tr(F_1 \vee F_2) &\triangleq tr(F_1) \vee tr(F_2)
\end{aligned}
$$

$$
\begin{aligned}
tr(\forall x_i.\exists y_i.G) \triangleq\ & (OK_{\{x_i\}} \wedge I_{\{x_i\}^{op}}) -\!\!* \\
& \sim((OK_{\{\leq x_i\}} \wedge I_{\{\geq y_i\}}) \wedge \sim(OK_{\{\leq y_i\}} \wedge I_{\{\geq x_{i+1}\}} \wedge tr(G)))
\end{aligned}
$$

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

Proposition 7. *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 [5] to maintain decidability, and also recursive definitions or special atomic predicates [1] for describing paths through the heap.

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. S. A. Cook and D.C. Oppen. An assertion language for data structures. In *Principles of Programming Languages*, pages 160–166. ACM, 1975.
3. S. Ishtiaq. BI is strongly expressive. available at http://www.dcs.qmw.ac.uk/˜ si/, November 2000.
4. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Principles of Programming Languages*, January 2001.
5. 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).
6. J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*. Palgrave, 2000.

A Appendix: Proofs of Proposition 2 and 3

In the appendix, we prove Proposition 2 and 3, which are important in showing that the fragment in Section 4 is decidable. All assertions in the section are assumed to be instances of the grammar of Theorem 2.

Proposition. *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 $(s, h * h_1) \models R$.

Note that the only-if direction follows from the definition of $\mathrel{-\!\!*}$. So, we concentrate on the other direction. In order to show the if direction, we partition heaps so that no assertion with certain conditions, which both Q and R also satisfy, can tell apart two heaps in the same partition; then, we show that for every heap h_1' with $h \# h_1'$ and $s, h \models Q$, it is possible to find a heap h_1 such that h_1 satisfies all the conditions in the proposition and $h * h_1$ and $h * h_1'$ are in the same partition; the conclusion follows from such arguments. For a stack s, a natural number n and a finite set X of variables, we define an equivalence relation $\sim_{s,n,X}$ between heaps, whose equivalence classes formalize the partitioning of heaps just described. Intuitively, two heaps are related if they are essentially the same as far as the variables in X are concerned, and they have the same amount of "garbage heap cells" upto n.

Definition 9 **($=_S$).** *Given a set of values $S \subseteq Val$, $=_S$ is a relation between pairs of values such that for any $(v_0, v_1), (v_0', v_1') \in Val \times Val$, $(v_0, v_1) =_S (v_0', v_1')$ iff for $i = 0, 1$,*

$$\text{if } v_i \in S \cup \{nil\}, \text{ then } v_i = v_i'; \text{ and if } v_i \notin S \cup \{nil\}, \text{ then } v_i' \notin S \cup \{nil\}.$$

Definition 10 **($\sim_{s,n,X}$).** *Given a stack s, a natural number n and a set X of variables, $\sim_{s,n,X}$ is a relation between heaps such that $h \sim_{s,n,X} h'$ iff*

- *$s(X) \cap dom(h) = s(X) \cap dom(h')$;*
- *for all $l \in s(X) \cap dom(h)$, $h(l) =_{s(X)} h'(l)$;*
- *if $|dom(h) - s(X)| < n$, then $|dom(h) - s(X)| = |dom(h') - s(X)|$; and*
- *if $|dom(h) - s(X)| \geq n$, then $|dom(h') - s(X)| \geq n$.*

Intuitively, $h \sim_{s,n,X} h'$ holds if h and h' are the "same" as far as variables in X are concerned; moreover, the number of heap cells which are not pointed to by any variable in X is the same upto n for h and h'.

The equivalence relation is preserved by splitting a heap and adding a new heap.

Lemma 4. *If $m = n_1 + n_2$ and $h \sim_{s,m,X} h'$ and $h = h_1 * h_2$, then there exist h_1', h_2' such that*

1. *$h_1' \# h_2'$;*
2. *$h' = h_1' * h_2'$;*
3. *$h_1 \sim_{s,n_1,X} h_1'$; and*
4. *$h_2 \sim_{s,n_2,X} h_2'$.*

Proof. Let $\{A, B\}$ be a partition of $dom(h') - s(X)$ such that

- when $|dom(h) - s(X)| < m$, $|A| = |dom(h_1) - s(X)|$ and $|B| = |dom(h_2) - s(X)|$;
- when $|dom(h) - s(X)| \geq m$ and $|dom(h_2) - s(X)| \geq n_2$, $|A| = \min(|dom(h_1) - s(X)|, n_1)$; and
- when $|dom(h) - s(X)| \geq m$ and $|dom(h_2) - s(X)| < n_2$, $|B| = |dom(h_2) - s(X)|$.

Notice that the first case is possible because when $|dom(h) - s(X)| < m$, $|dom(h') - s(X)| = |dom(h) - s(X)|$ and $|dom(h) - s(X)| = |dom(h_1) - s(X)| + |dom(h_2) - s(X)|$. Also, notice that the second and the third cases are also possible because when $|dom(h) - s(X)| \geq m$, $|dom(h') - s(X)| \geq m \geq max(n_1, n_2)$.

Let h_A' and h_B' be restrictions of h' to $A \cup (dom(h_1) \cap s(X))$ and $B \cup (dom(h_2) \cap s(X))$, respectively. We show that h_A' and h_B' satisfy the required properties.

1. $h'_A \# h'_B$:

$$
\begin{aligned}
& dom(h'_A) \cap dom(h'_B) \\
= \quad & \\
& (A \cup (dom(h_1) \cap s(X))) \cap (B \cup (dom(h_2) \cap s(X))) \\
= \quad & \\
& (A \cap B) \cup (dom(h_1) \cap dom(h_2) \cap s(X)) \\
& \cup (dom(h_1) \cap s(X) \cap B) \cup (dom(h_2) \cap s(X) \cap A) \\
= \quad & \qquad \because A \cap B = \emptyset \wedge dom(h_1) \cap dom(h_2) = \emptyset \\
& (dom(h_1) \cap s(X) \cap B) \cup (dom(h_2) \cap s(X) \cap A) \\
= \quad & \qquad \because B \cap s(X) \subseteq (dom(h') - s(X)) \cap s(X) = \emptyset \\
& dom(h_2) \cap s(X) \cap A \\
= \quad & \qquad \because A \cap s(X) \subseteq (dom(h') - s(X)) \cap s(X) = \emptyset \\
& \emptyset
\end{aligned}
$$

2. $h' = h'_A * h'_B$:
Since $h'_A \# h'_B$ and h'_A and h'_B are restrictions of h', it suffices to show that $dom(h'_A) \cup dom(h'_B) = dom(h')$.

$$
\begin{aligned}
& dom(h'_A) \cup dom(h'_B) \\
= \quad & \\
& (A \cup (dom(h_1) \cap s(X))) \cup (B \cup (dom(h_2) \cap s(X))) \\
= \quad & \\
& (A \cup B) \cup ((dom(h_1) \cup dom(h_2)) \cap s(X)) \\
= \quad & \qquad \because h = h_1 * h_2 \\
& (A \cup B) \cup (dom(h) \cap s(X)) \\
= \quad & \qquad \because h \sim_{s,m,X} h' \\
& (A \cup B) \cup (dom(h') \cap s(X)) \\
= \quad & \qquad \because A \cup B = dom(h') - s(X) \\
& (dom(h') - s(X)) \cup (dom(h') \cap s(X)) \\
= \quad & \\
& dom(h')
\end{aligned}
$$

3. $h_1 \sim_{s,n_1,X} h'_A$:
To handle this condition and the last condition, we rely on the simplification of $\sim$ for heaps obtained by restricting two already related heaps:

> **Fact:** Suppose that $h \sim_{s,n,X} h'$ and that h_0 and h'_0 are restrictions of h and h', respectively. If $s(X) \cap dom(h_0) = s(X) \cap dom(h'_0)$, then $h_0(l) =_{s(X)} h'_0(l)$ for all $l \in s(X) \cap dom(h_0)$.

Therefore, it suffices to show that

- $dom(h_1) \cap s(X) = dom(h'_A) \cap s(X)$;
- if $|dom(h_1) - s(X)| < n_1$, then $|dom(h_1) - s(X)| = |dom(h'_A) - s(X)|$; and
- if $|dom(h_1) - s(X)| \geq n_1$, then $|dom(h'_A) - s(X)| \geq n_1$.

For the first bullet,

$$
\begin{array}{cl}
& dom(h'_A) \cap s(X) \\
= & \\
& (A \cup (dom(h_1) \cap s(X))) \cap s(X) \\
= & \\
& (A \cap s(X)) \cup (dom(h_1) \cap s(X)) \\
= & \quad \because A \subseteq dom(h') - s(X) \\
& dom(h_1) \cap s(X)
\end{array}
$$

For the other two bullets, we note that $dom(h'_A) - s(X) = A$ because:

$$
\begin{array}{cl}
& dom(h'_A) - s(X) \\
= & \\
& (A \cup (dom(h_1) \cap s(X))) - s(X) \\
= & \\
& A - s(X) \\
= & \quad \because A \subseteq dom(h') - s(X) \\
& A
\end{array}
$$

We divide the proof into two cases, when $|dom(h) - s(X)| < m$ and when $|dom(h) - s(X)| \geq m$. In the first case, $|A| = |dom(h_1) - s(X)|$ holds by the choice of A; therefore, the requirement is satisfied. For the second, we consider two subcases depending on whether $|dom(h_2) - s(X)| \geq n_2$ or not. If $|dom(h_2) - s(X)| \geq n_2$, $|A| = \min(|dom(h_1) - s(X)|, n_1)$ by the choice of A. Therefore, it meets the requirement. The only remaining case is when $|dom(h) - s(X)| \geq m$ and $|dom(h_2) - s(X)| < n_2$. In this case, $|B|$ is smaller than n_2 by the choice of B. Therefore,

$$
\begin{aligned}
|A| &= |dom(h) - s(X)| - |B| \\
&\geq n_1 + n_2 - |B| \\
&\geq n_1
\end{aligned}
$$

Since $|dom(h_1) - s(X)|$ is greater than or equal to n_1 in this case, the requirement is met, too.

4. $h_2 \sim_{s,n_2,X} h'_B$:
Since h_2 and h'_B are restrictions of already related heaps, it suffices to show that
   - $dom(h_2) \cap s(X) = dom(h'_B) \cap s(X)$;
   - if $|dom(h_2) - s(X)| < n_2$, then $|dom(h_2) - s(X)| = |dom(h'_B) - s(X)|$; and
   - if $|dom(h_2) - s(X)| \geq n_2$, then $|dom(h'_B) - s(X)| \geq n_2$.

For the first bullet,

$$
\begin{array}{cl}
& dom(h'_B) \cap s(X) \\
= & \\
& (B \cup (dom(h_2) \cap s(X))) \cap s(X) \\
= & \\
& (B \cap s(X)) \cup (dom(h_2) \cap s(X)) \\
= & \quad \because B \subseteq dom(h') - s(X) \\
& dom(h_2) \cap s(X)
\end{array}
$$

For the other two, we notice that $dom(h'_B) - s(X) = B$.

$$
\begin{array}{ll}
 & dom(h'_B) - s(X) \\
= & \\
 & (B \cup (dom(h_2) \cap s(X))) - s(X) \\
= & \\
 & B - s(X) \\
= & \quad \because B \subseteq dom(h') - s(X) \\
 & B
\end{array}
$$

When $|dom(h) - s(X)| < m$ or $|dom(h_2) - s(X)| < n_2$, $|B| = |dom(h_2) - s(X)|$. Therefore, the two requirements are satisfied. The only remaining case is when $|dom(h) - s(X)| \geq m$ and $|dom(h_2) - s(X)| \geq n_2$. We show that $|B| \geq n_2$. Note that $|A| = min(|dom(h_1) - s(X)|, n_1)$ by the choice of A in this case. Therefore,

$$
\begin{aligned}
|B| &= |dom(h) - s(X)| - |A| \\
&\geq m - |A| \\
&\geq m - n_1 \\
&\geq n_2
\end{aligned}
$$

□

Lemma 5. *If $h \sim_{s,n,X} h'$, $h \# h_1$ and $h' \# h_1$, then $h * h_1 \sim_{s,n,X} h' * h_1$.*

Proof. We check each condition in the definition of $h * h_1 \sim_{s,n,X} h' * h_1$.

1. $s(X) \cap dom(h * h_1) = s(X) \cap dom(h' * h_1)$:

$$
\begin{array}{ll}
 & s(X) \cap dom(h * h_1) \\
= & \\
 & (s(X) \cap dom(h)) \cup (s(X) \cap dom(h_1)) \\
= & \quad \because h \sim_{s,n,X} h' \\
 & (s(X) \cap dom(h')) \cup (s(X) \cap dom(h_1)) \\
= & \\
 & s(X) \cap dom(h' * h_1)
\end{array}
$$

2. for all $l \in s(X) \cap dom(h * h_1)$, $(h * h_1)(l) =_{s(X)} (h' * h_1)(l)$:
   For such l, $l \in s(X) \cap dom(h)$ or $l \in s(X) \cap dom(h_1)$. In the first case, $(h * h_1)(l) = h(l)$ and $(h' * h_1)(l) = h'(l)$, and $h(l) =_{s(X)} h'(l)$ since $h \sim_{s,n,X} h'$. Therefore, $(h * h_1)(l) =_{s(X)} (h' * h_1)(l)$. In the second case, $(h * h_1)(l) = h_1(l) = (h' * h_1)(l)$. Since $=_{s(X)}$ is reflexive, $(h * h_1)(l) =_{s(X)} (h' * h_1)(l)$ also holds.
3. if $|dom(h * h_1) - s(X)| < n$, then $|dom(h * h_1) - s(X)| = |dom(h' * h_1) - s(X)|$:
   When $|dom(h * h_1) - s(X)| < n$, $|dom(h) - s(X)| < n$. Since $h \sim_{s,n,X} h'$, $|dom(h) - s(X)| = |dom(h') - s(X)|$. The conclusion easily follows from this

as shown below:

$$
\begin{array}{ll}
& |dom(h * h_1) - s(X)| \\
= & \\
& |dom(h) - s(X)| + |dom(h_1) - s(X)| \\
= & \\
& |dom(h') - s(X)| + |dom(h_1) - s(X)| \\
= & \\
& |dom(h' * h_1) - s(X)|
\end{array}
$$

4. if $|dom(h * h_1) - s(X)| \geq n$, then $|dom(h' * h_1) - s(X)| \geq n$:
   There are two cases: $|dom(h) - s(X)| < n$ or $|dom(h) - s(X)| \geq n$. In the first case, since $h \sim_{s,n,X} h'$, $|dom(h' * h_1) - s(X)| = |dom(h * h_1) - s(X)| \geq n$. In the second case, $|dom(h' * h_1) - s(X)| \geq |dom(h') - s(X)| \geq n$ because $h \sim_{s,n,X} h'$.

□

The equivalence relation is also preserved by relocating "garbage heap cells", removing some of them and overwriting an "uninteresting value" of each heap cell by another "uninteresting value". Given a subset V of Val and v in $Val - V - \{nil\}$, we use $prun_{V,v}$ to denote a function from $Val \times Val$ to $Val \times Val$ which overwrites "uninteresting values", i.e., values not in $V \cup \{nil\}$, by another "uninteresting value" v:

$$prun_{V,v}(v_0, v_1) = (v'_0, v'_1)$$

where for $i = 0, 1$, if $v_i \in (V \cup \{nil\})$, then $v'_i = v_i$, and otherwise, $v'_i = v$.

Lemma 6. *For all $(s, h) \in State$, $m \in \mathbf{Nat}$, $v \in Val - s(X) - \{nil\}$ and all finite subsets B of Loc such that $B \cap s(X) = \emptyset$ and $|B| = m$, there exists h' with the following properties:*

- *$h \sim_{s,m,X} h'$;*
- *$dom(h') - s(X) \subseteq B$;*
- *for all $l \in dom(h') - s(X)$, $h'(l) = (v, v)$; and*
- *for all $l \in dom(h') \cap s(X)$, $prun_{s(X),v}(h(l)) = h'(l)$*

Proof. Pick up a subset B_0 of B such that $|B_0| = min(|dom(h) - s(X)|, m)$. Then, define h' such that

$$h'(l) = \begin{cases} (v, v) & \text{when } l \in B_0 \\ prun_{s(X),v}(h(l)) & \text{when } l \in dom(h) \cap s(X) \\ \textsf{undefined} & \text{otherwise} \end{cases}$$

Notice that h' is well-defined because B_0 and $(dom(h) \cap s(X))$ are disjoint. We show that all conditions for $h \sim_{s,m,X} h'$ are satisfied.

- $dom(h) \cap s(X) = dom(h') \cap s(X)$:
  It holds since $B_0 \cap s(X) = \emptyset$.
- for all $l \in dom(h) \cap s(X)$, $h(l) =_{s(X)} h'(l)$:
  By the definition of h', $prun_{s(X),v}(h(l)) = h'(l)$ for all such l. $h(l) =_{s(X)} h'(l)$ follows from this since $prun_{s(X),v}(h(l)) =_{s(X)} h(l)$.

- if $|dom(h) - s(X)| < m$, then $|dom(h) - s(X)| = |dom(h') - s(X)|$; and if $|dom(h) - s(X)| \geq m$, then $|dom(h') - s(X)| \geq m$:

  Since B_0 is defined to be $min(|dom(h) - s(X)|, m)$, it suffices to show that $dom(h') - s(X) = B_0$, which is, in fact, the case by the construction of h'.

$\square$

The most important property of the equivalence relation is that no assertion can tell apart two equivalent heaps:

Lemma 7. *Given heaps h, h', a stack s, a predicate P and a set X of variables with $FV(P) \subseteq X$, if $h \sim_{s,|P|,X} h'$ and $(s, h) \models P$, then $(s, h') \models P$.*

Proof. We use induction on the structure of P.

- $P \equiv (E \mapsto E_1, E_2)$:

$$
\begin{array}{ll}
& (s,h) \models (E \mapsto E_1, E_2) \\
\Longrightarrow & \\
& dom(h) = \{[\![E]\!](s)\} \wedge h([\![E]\!](s)) = ([\![E_1]\!](s), [\![E_2]\!](s)) \\
\Longrightarrow & \qquad \because [\![E]\!](s) \in s(X) \wedge [\![E_i]\!](s) \in s(X) \cup \{nil\} \wedge h \sim_{s,1,X} h' \\
& dom(h') = \{[\![E]\!](s)\} \wedge h'([\![E]\!](s)) = ([\![E_1]\!](s), [\![E_2]\!](s)) \\
\Longrightarrow & \\
& (s,h') \models (E \mapsto E_1, E_2)
\end{array}
$$

- $P \equiv (E_1 = E_2)$:

$$
\begin{array}{ll}
& (s,h) \models (E_1 = E_2) \\
\Longrightarrow & \\
& [\![E_1]\!](s) = [\![E_2]\!](s) \\
\Longrightarrow & \\
& (s,h') \models (E_1 = E_2)
\end{array}
$$

- $P \equiv \mathsf{false}$:

  In this case, $(s, h) \models \mathsf{false}$ never holds.

- $P \equiv Q \Rightarrow R$:

$$
\begin{array}{ll}
& (s,h) \models Q \Rightarrow R \\
\Longrightarrow & \\
& (s,h) \models Q \text{ implies } (s,h) \models R \\
\Longrightarrow & \qquad \because \text{the induction hypothesis and } \sim_{s,|P|,X} \text{ is symmetric} \\
& (s,h') \models Q \text{ implies } (s,h') \models R \\
\Longrightarrow & \\
& (s,h') \models Q \Rightarrow R
\end{array}
$$

- $P \equiv Q * R$:

$$
\begin{array}{ll}
& (s,h) \models Q * R \\
\Longrightarrow & \\
& \exists h_0, h_1.\ \ h_0 \# h_1\ h = h_0 * h_1,\ (s,h_0) \models Q \text{ and } (s,h_1) \models R \\
\Longrightarrow & \qquad \because |P| = |Q| + |R| \wedge \text{Lemma 4} \\
& \exists h'_0, h'_1, h_0, h_1. \\
& \quad h'_0 \# h'_1 \wedge h' = h'_0 * h'_1 \\
& \quad \wedge\ h_0 \sim_{s,|Q|,X} h'_0 \wedge h_1 \sim_{s,|R|,X} h'_1 \\
& \quad \wedge\ (s,h_0) \models Q \wedge (s,h_1) \models R \\
\Longrightarrow & \qquad \because \text{the induction hypothesis} \\
& \exists h'_0, h'_1.\ \ h'_0 \# h'_1,\ h' = h'_0 * h'_1,\ (s,h'_0) \models Q \text{ and } (s,h'_1) \models R \\
\Longrightarrow & \\
& (s,h') \models Q * R
\end{array}
$$

- $P \equiv Q \mathrel{-\!\!*} R$:
For all heaps h'_1 such that $(s,h'_1) \models Q$ and $h'_1 \# h'$, we have to show that $(s, h' * h'_1) \models Q$. Given such h'_1, let's choose a subset B of $Loc - dom(h) \cup dom(h') \cup s(X)$ such that $|B| = max(|Q|,|R|)$, which is possible because Loc is a countably infinite set and $dom(h) \cup dom(h') \cup s(X)$ is finite. By Lemma 6, there exists h_1 such that $h_1 \sim_{s,max(|Q|,|R|),X} h'_1$ and $dom(h_1) - s(X) \subseteq B$. Note that $(dom(h_1) - s(X)) \cap (dom(h) \cup dom(h')) = \emptyset$ by the choice of B, which we use in the following:

$$
\begin{array}{ll}
& dom(h_1) \cap (dom(h) \cup dom(h')) \\
= & \\
& ((dom(h_1) \cap s(X)) \cup (dom(h_1) - s(X))) \cap (dom(h) \cup dom(h')) \\
= & \qquad \because (dom(h_1) - s(X)) \cap (dom(h) \cup dom(h')) = \emptyset \\
& (dom(h_1) \cap s(X)) \cap (dom(h) \cup dom(h')) \\
= & \qquad \because h_1 \sim_{s,max(|Q|,|R|),X} h'_1 \\
& (dom(h'_1) \cap s(X)) \cap (dom(h) \cup dom(h')) \\
= & \\
& (dom(h'_1) \cap s(X) \cap dom(h)) \cup (dom(h'_1) \cap s(X) \cap dom(h')) \\
= & \qquad \because h \sim_{s,|P|,X} h' \\
& (dom(h'_1) \cap s(X) \cap dom(h')) \cup (dom(h'_1) \cap s(X) \cap dom(h')) \\
= & \qquad \because h' \# h'_1 \\
& \emptyset
\end{array}
$$

That is, $h \# h_1$ and $h' \# h_1$. Therefore, we have $h * h_1 \sim_{s,|R|,X} h' * h_1$ by Lemma 5. We also have $h' * h_1 \sim_{s,|R|,X} h' * h'_1$ by the same lemma because $h_1 \sim_{s,|R|,X} h'_1$, $h' \# h_1$ and $h' \# h'_1$. So, $h * h_1 \sim_{s,|R|,X} h' * h'_1$. Now, for obtaining the goal, which is to show $(s, h' * h'_1) \models R$, we only need to prove that $(s, h * h_1) \models R$; then, the conclusion follows by induction. Since $h_1 \sim_{s,|Q|,X} h'_1$, by the induction hypothesis, $(s,h_1) \models Q$. Now, it follows from $(s,h) \models Q \mathrel{-\!\!*} R$ that $(s, h * h_1) \models R$.

– $P \equiv \mathsf{emp}$:

$$
\begin{array}{ll}
 & (s,h) \models \mathsf{emp} \\
\Longrightarrow & \\
 & dom(h) = \emptyset \\
\Longrightarrow & \qquad \because h \sim_{s,1,X} h' \\
 & dom(h') \cap s(X) = \emptyset \wedge |dom(h') - s(X)| = 0 \\
\Longrightarrow & \\
 & dom(h') = \emptyset \\
\Longrightarrow & \\
 & (s,h') \models \mathsf{emp}
\end{array}
$$

□

We have developed enough properties of the equivalence relation $\sim_{s,n,X}$ to show Proposition 2, which says that in the interpretation of $(s,h) \models Q \mathrel{-\!\!*} R$, it suffices to consider heaps only in a certain finite set. We show that for all h_0 with $h \# h_0$ and $(s,h_0) \models Q$, there is a heap h_1 in the finite set, i.e. satisfying the conditions in the proposition, such that $h_1 * h$ is related to $h_0 * h$ by $\sim_{s,|R|,X}$. Then, Lemma 7 gives the conclusion: $(s, h * h_1) \models R$ by the assumption, so $(s, h * h_0) \models R$ because $h_0 * h \sim_{s,|R|,X} h_1 * h$ holds (Lemma 7). We use Lemma 6 to get h_1 which is related to h_0 by $\sim_{s,max(|Q|,|R|),X}$ and satisfies the last two conditions in the proposition, which are $dom(h_1) - s(X) \subseteq B$ and $h_1(l) \in (s(X) \cup \{nil, v\}) \times (s(X) \cup \{nil, v\})$ for all $l \in dom(h_1)$. Note that h_1 is related to h_0 by both $\sim_{s,|Q|,X}$ and $\sim_{s,|R|,X}$ since both $|Q|$ and $|R|$ are less than or equal to $max(|Q|,|R|)$. So, $(s,h_1) \models Q$ by Lemma 7. So, in order to show that h_1 is in the finite set, it remains to prove that $h_1 \# h$. Note again that the additional requirement, $h_0 * h \sim_{s,|R|,X} h_1 * h$, also follows from $h_1 \# h$ because $h_0 \sim_{s,|R|,X} h_1$ and $h_0 \# h$ (Lemma 5). The following shows that $h_1 \# h$ holds:

$$
\begin{array}{ll}
 & dom(h_1) \cap dom(h) \\
= & \\
 & \Big((dom(h_1) - s(X)) \cup (dom(h_1) \cap s(X))\Big) \cap dom(h) \\
= & \qquad \because h_0 \sim_{s,|Q|,X} h_1 \\
 & \Big((dom(h_1) - s(X)) \cup (dom(h_0) \cap s(X))\Big) \cap dom(h) \\
\subseteq & \qquad \because dom(h_1) - s(X) \subseteq B \wedge h_0 \# h \\
 & B \cap dom(h) \\
= & \qquad \because B \subseteq Loc - (dom(h) \cup s(X)) \\
 & \emptyset
\end{array}
$$

Before showing Proposition 3, we observe different ways to express that a heap h' is obtained from h by renaming locations according to the bijection $r\colon Val \to Val$ with $r(nil) = nil$. For two functions $f\colon A \to B$ and $g\colon C \to D$, let $f \times g$ be a function from $A \times C$ to $B \times D$ given by $(f \times g)(a,b) = (f(a), g(b))$. Then, all the following three indicate that a heap h' is obtained from h simply by renaming locations:

– $(r \times r) \circ h = h' \circ (r|_{Loc})$;
– $h = (r^{-1} \times r^{-1}) \circ h' \circ (r|_{Loc})$; and

– $h \circ (r|_{Loc})^{-1} = (r^{-1} \times r^{-1}) \circ h'$

Note that the first is precisely the last requirement in the definition of $\approx_X$.

Lemma 8. *Given states $(s,h),(s',h')$ and an expression E, if $(s,h) \approx_Y (s',h')$ by a bijection $r\colon Val \to Val$ and $Y \supseteq FV(E)$, then $r([\![E]\!](s)) = [\![E]\!](s')$.*

Proposition. *For all 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 $(s',h') \models P$.*

Proof. Let r be the bijection in the definition of $(s,h) \approx_{FV(P)} (s',h')$. We use induction on the structure of P.

– $P \equiv (E \mapsto E_1, E_2)$:

$$
\begin{array}{l}
\quad (s,h) \models (E \mapsto E_1, E_2) \\
\Longrightarrow \\
\quad dom(h) = \{[\![E]\!](s)\} \wedge h([\![E]\!](s)) = ([\![E_1]\!](s), [\![E_2]\!](s)) \\
\Longrightarrow \qquad \because (s,h) \approx_{FV(P)} (s',h') \\
\quad dom(h') = \{r([\![E]\!](s))\} \wedge h'(r([\![E]\!](s))) = (r([\![E_1]\!](s)), r([\![E_2]\!](s))) \\
\Longrightarrow \qquad \because \text{Lemma 8} \\
\quad dom(h') = \{[\![E]\!](s')\} \wedge h'([\![E]\!](s')) = ([\![E_1]\!](s'), [\![E_2]\!](s'))
\end{array}
$$

– $P \equiv (E_1 = E_2)$:

$$
\begin{array}{l}
\quad (s,h) \models E_1 = E_2 \\
\Longrightarrow \\
\quad [\![E_1]\!](s) = [\![E_2]\!](s) \\
\Longrightarrow \\
\quad r([\![E_1]\!](s)) = r([\![E_2]\!](s)) \\
\Longrightarrow \qquad \because \text{Lemma 8} \\
\quad [\![E_1]\!](s') = [\![E_2]\!](s')
\end{array}
$$

– $P \equiv \mathsf{false}$:
$(s,h) \models \mathsf{false}$ never holds.

– $P \equiv Q \Rightarrow R$:

$$
\begin{array}{l}
\quad (s,h) \models Q \Rightarrow R \\
\Longrightarrow \\
\quad (s,h) \models Q \text{ implies } (s,h) \models R \\
\Longrightarrow \qquad \because \text{the induction hypothesis and the symmetry of } \approx_{FV(Q)} \\
\quad (s',h') \models Q \text{ implies } (s',h') \models R \\
\Longrightarrow \\
\quad (s',h') \models Q \Rightarrow R
\end{array}
$$

– $P \equiv \mathsf{emp}$:
In this case, we should show that $dom(h') = \emptyset$. Since $(s,h) \approx_{FV(P)} (s',h')$ by r, it suffices show that $dom((r \times r) \circ h \circ (r|_{Loc})^{-1}) = \emptyset$. Notice that $r \times r$ and $(r|_{Loc})^{-1}$ are bijective. Therefore, the above is equivalent to $dom(h) = \emptyset$, which is the case since $(s,h) \models \mathsf{emp}$.

– $P \equiv Q * R$:
Since $(s,h) \models Q * R$, there exist h_1, h_2 such that $h_1 \# h_2$, $h = h_1 * h_2$, $(s,h_1) \models Q$ and $(s,h_2) \models R$. Let h'_i be $(r \times r) \circ h_i \circ (r|_{Loc})^{-1}$ for $i = 1,2$. Then, it suffices to show that $h'_1 \# h'_2$, $h = h'_1 * h'_2$, $(s,h_1) \approx_{FV(Q)} (s', h'_1)$ by r and $(s,h_2) \approx_{FV(R)} (s', h'_2)$ by r. The conclusion follows by the induction hypothesis.

- $h'_1 \# h'_2$:

$$
\begin{array}{ll}
 & dom(h'_1) \cap dom(h'_2) \\
= & \\
 & dom((r \times r) \circ h_1 \circ (r|_{Loc})^{-1}) \cap dom((r \times r) \circ h_2 \circ (r|_{Loc})^{-1}) \\
= & \quad \because r \times r \text{ is total} \\
 & dom(h_1 \circ (r|_{Loc})^{-1}) \cap dom(h_2 \circ (r|_{Loc})^{-1}) \\
= & \quad \because r \text{ is bijective} \\
 & r(dom(h_1)) \cap r(dom(h_2)) \\
= & \quad \because r \text{ is bijective} \\
 & r(dom(h_1) \cap dom(h_2)) \\
= & \\
 & \emptyset
\end{array}
$$

- $h = h'_1 * h'_2$:
  Note that $dom(h'_1) \cup dom(h'_2) = r(dom(h_1) \cup dom(h_2))$ by the similar calculation as above, and so, $dom(h'_1) \cup dom(h'_2) = r(dom(h)) = dom(h')$. Therefore, it suffices to show that for all $l \in dom(h'_1)$, $h'_1(l) = h'(l)$ and for all $l \in dom(h'_2)$, $h'_2(l) = h'(l)$. We only show the first; the other is similar.

$$
\begin{array}{rcl}
h'_1(l) & = & (r \times r)(h_1((r|_{Loc})^{-1}(l))) \\
 & = & (r \times r)(h((r|_{Loc})^{-1}(l))) \\
 & = & h'(l)
\end{array}
$$

- $(s,h_1) \approx_{FV(Q)} (s', h'_1)$ by r:
  Since $(s,h) \approx_{FV(Q*R)} (s', h')$, the conditions on stacks hold. And by definition of h'_1, $h'_1 = (r \times r) \circ h_1 \circ (r|_{Loc})^{-1}$.
- $(s,h_2) \approx_{FV(Q)} (s', h'_2)$ by r:
  The case is similar to the previous one.

– $P \equiv Q \mathrel{-\!\!*} R$:
Let h'_1 be a heap such that $h' \# h'_1$ and $(s', h'_1) \models Q$. We should show that $(s', h' * h'_1) \models Q$. Define h_1 as $(r^{-1} \times r^{-1}) \circ h'_1 \circ (r|_{Loc})$. Then, $(s', h'_1) \approx_{FV(Q)} (s, h_1)$ by r and so $(s, h_1) \models Q$ by the induction hypothesis. Note that $h \# h_1$

holds because

$$
\begin{array}{ll}
 & dom(h) \cap dom(h_1) \\
= & \\
 & dom((r^{-1} \times r^{-1}) \circ h' \circ (r|_{Loc})) \cap dom((r^{-1} \times r^{-1}) \circ h'_1 \circ (r|_{Loc})) \\
= & \quad \because r^{-1} \times r^{-1} \text{ is total} \\
 & dom(h' \circ (r|_{Loc})) \cap dom(h'_1 \circ (r|_{Loc})) \\
= & \quad \because r^{-1} \text{ is bijective} \\
 & r^{-1}(dom(h')) \cap r^{-1}(dom(h'_1)) \\
= & \quad \because r^{-1} \text{ is bijective} \\
 & r^{-1}(dom(h') \cap dom(h'_1)) \\
= & \\
 & \emptyset
\end{array}
$$

So, we also know that $(s, h * h_1) \models R$. Now, it suffices to show $(s, h * h_1) \approx_{FV(R)} (s', h' * h'_1)$ by r; then, the conclusion follows by the induction hypothesis. We note that all requirements on stacks hold since $(s, h) \approx_{FV(Q -\!\!* R)} (s', h')$ by r by assumption. So, the only step which we need to show is that

$$h' * h'_1 = (r \times r) \circ (h * h_1) \circ (r|_{Loc})^{-1}$$

The following shows it:

$$
\begin{array}{ll}
 & h' * h'_1 \\
= & \\
 & \Big((r \times r) \circ h \circ (r|_{Loc})^{-1}\Big) * \Big((r \times r) \circ h_1 \circ (r|_{Loc})^{-1}\Big) \\
= & \quad \because (r \times r) \text{ is total} \\
 & (r \times r) \circ \Big((h \circ (r|_{Loc})^{-1}) * (h_1 \circ (r|_{Loc})^{-1})\Big) \\
= & \quad \because (r|_{Loc})^{-1} \text{ is bijective} \\
 & (r \times r) \circ (h * h_1) \circ (r|_{Loc})^{-1}
\end{array}
$$

$\square$