Imperial College London
Department of Computing

Separation Logic Reasoning for Resources with Arity

by

Ana Armas

Submitted in part fulfilment of the requirements for the
MSc Degree in Computing (Theory)

September 2011

Abstract

Separation Logic has been developed over the last few years as a way to reason about programs in terms of the resources they use, specially when these resources can be shared. The most widely studied example of such resources is a shared memory heap. Most approaches to this model consider an ideal memory heap, with additional memory always available for allocation. Here, we explore a slightly different model where, as is the case when working with physical memory, it is possible to run out of space. Through this model, we introduce the notion of *arity* of a state, which is also present in another quite different model: the partial hydrocarbon model, which provides a simplified representation of interaction between organic chemistry partial molecules.

Acknowledgements

I want to thank my supervisor, Philippa Gardner, for being always supportive, enthusiastic and extremely approachable. For having given me a lot of self confidence. For being interested in my work and helping me give the best of myself. I have to thank her also for assigning me such an amazing second supervisor, which leads me to the next paragraph...

I want to thank James Brotherston, whose guidance and support have been invaluable. Thanks for solving my doubts, silly as they may have been sometimes, with such patience. For brainstorming with me, for reading so many proofs and explanations and commenting on them with such detail and interest. For having such interesting ideas. For treating this project like part of his own work. For pushing me a little further right until the end.

I also want to thank my family: my parents, for making this possible, for always encouraging me to follow my dreams and believing in me, whatever I decide to do. My sister, for being there when I have needed her. My boyfriend, for being there all the time and really helping me through the hardest moments. My grandparents, because they will always be a part of anything good that I do.

Thanks as well to Joaquín Hernández and Carlos Gregorio, for they are guilty that I ended up so crazy about Mathematics and Theoretical Computing. I really would not be here had it not been because of them and for that I will be eternally grateful.

Thanks as well to my friends, the ones that are far, and especially the ones that have been closer to me during the year.

Finally, thanks to the Fundación Mutua Madrileña, for honouring me with a wonderful scholarship that allowed me to focus on what was important over the academic year: learning.

Contents

1 Introduction **5**

2 Background **7**

2.1 The Memory Heap Model . . . 7

2.1.1 Syntax and Semantics of Commands . . . 7

2.1.2 Syntax and Semantics of Program Assertions . . . 9

2.1.3 Hoare Reasoning with Separation Logic . . . 10

2.1.4 Frame Rule: Soundness . . . 13

2.1.5 Frame Rule: Completeness . . . 14

3 Resources with arity **17**

3.1 The Bounded Heap Model . . . 18

3.1.1 Assertion Language . . . 18

3.1.2 Tight Specifications . . . 22

3.1.3 Example: Proof of a program . . . 23

3.1.4 Results . . . 26

3.1.5 Alternative Approach . . . 35

3.2 The Bounded Heap Model: allocating contiguous cells . . . 36

3.3 The Partial Hydrocarbon Model . . . 44

3.3.1 Commands and operational semantics . . . 50

3.3.2 Assertion Language . . . 53

3.3.3 Locality Conditions. Soundness of the Frame Rule. . . . 53

3.3.4 Tight Specifications. . . . 58

4 Conclusions and Future Work **64**

1 Introduction

Separation logic was introduced a few years ago as a powerful tool to reason about programs in terms of the resources they use, especially when these resources are shared. This is the case for dynamic memory, where one memory address can be 'pointed at' by several variables or by other memory addresses. A logic was needed that allowed to give tight specifications of programs, mentioning only the resources actually used by them and ensuring that anything not mentioned remains intact. This kind of reasoning is meant to be automated, so it was developed in a syntactic fashion, in the style of Hoare logic [4, 5], with axioms and rules that are used to infer new statements from them. The rules in separation logic follow the style of Hoare logic rules, but instead of using only classical logic to express the pre and postconditions in Hoare triples, they use the logic BI of bunched implications [12, 13]. This logic has, in particular, a new connective: the separating conjunction $*$. The formula $A * A'$ denotes those heaps that can be split into two disjoint parts satisfying A and A' respectively. This new connective makes it possible to perform spatial reasoning and express restrictions in the sharing of the dynamic memory heap, and to do so in a concise and scalable way.

The separating conjunction also makes it possible to write the main rule of separation logic: the Frame Rule. Having a small specification of some command, of the form $\{A\}\ C\ \{A'\}$, the Frame Rule allows us, under some reasonable conditions, to infer a specification $\{A*D\}\ C\ \{A'*D\}$ for the same command with a bigger initial state. The meaning of the Frame Rule is the following: if we have a command that acts locally on a portion of the memory, then, if we consider a bigger portion of the memory, the extended parts will stay intact. This rule being so important, one can see how something crucial when developing this logic, or extending it to various other resource models, is to prove soundness of this rule. Yang and O'Hearn managed to isolate two locality conditions for the commands that are necessary and sufficient to guarantee soundness of the Frame Rule [2]. These two conditions are Safety Monotonicity and the Frame Property, and we will become more familiar with them in what follows. Completeness of the Frame Rule, although not as vital as soundness, is also an important issue. It is proved too in [2].

In [9] Pym, O'Hearn and Yang explore a variety of resource models that can be described with BI, such as Petri nets, mobile processes or money. In [3], Calcagno, O'Hearn and Yang present abstract separation logic, which reasons about *separation algebras*: partial, cancellative, commutative monoids. Separation algebras work as an abstract resource model that generalises over a set of various other models. They prove that the conditions of Safety Monotonicity and Frame Property still guarantee soundness of the Frame Rule in this asbtract setting.

In this paper we look into some different variants of the standard memory model. While it is usually considered to have a finite amount af allocated cells but an infinite global set of addresses, we will explore a couple of approaches with a finite global set of addresses, as it is in any actual physical memory. In the standard approach, memory allocation never leads to a memory fault because a fresh memory address is always assumed to be available. Furthermore, in [1] Reynolds assumes that there is always a set of arbitrary size of contiguous unallocated cells, in order to be able to use unrestricted address arithmetic. We give in section 3.1 a simple extension of the model that allows allocation of single cells in a bounded memory setting, and in section 3.2 we will see how it is far from trivial to give a sensible operational semantics for allocation of multiple contiguous cells if we consider a bounded heap.

Raza looks into the idea of a finite memory heap too in his PhD thesis [6], although his approach is quite different to ours. He treats the set of non-allocated addresses as a "unique, atomic piece of resource", while we split it into pieces at will, obtaining the advantage of a finer control over the use of inactive memory cells.

Through the bounded memory model, we will introduce the notion of *arity* of a state, and we will see how memory allocation commands, in particular, need states to have a certain arity in order not to lead to a memory fault.

In our main approach to the bounded memory model (in section 3.1) we regard arity as simply the number of cells that can still be allocated. In this approach we will not be able to allocate blocks of contiguous cells at once (but we will attempt to fix this in a later approach). A command that wants to allocate a new cell will need to act on a state where at least one cell is free. In order to prevent framing new portions of memory from being resonsible for memory faults (for otherwise we would lose soundness of the Frame Rule), some information about free cells must be included explicitly in the state. With this purpose, we will consider that a certain amount of reserved, inactive (i.e. non-allocated) cells are also part of our states. We define our new states in such a way that extending a state to a bigger one will not affect the number of cells we had already reserved, and so arity will be preserved by framing as needed. Safety Monotonicity and the Frame Property are proved here for this model, and so is soundness of the Frame Rule — of course with respect to a set of assertions with a syntax, slightly different from the usual, that makes them capable of giving information about the arity of states. We also provide tight specifications for the allocation and deallocation commands and prove their soundness as well as soundness of all other rules in the logic. Hence, we provide a setting were we can prove specifications of programs that perform complex allocation/deallocation tasks (as long as cells are allocated and deallocated one by one) in a setting where the available heap memory is limited.

One may wonder why we only take a number of reserved cells, instead of a set of specific such cells. In section 3.1.5 we look into that alternative and expose a big flaw: a naive adaptation of the usual Frame Property fails. We make an attempt to overcome this problem by giving an alternative Frame Property that still guarantees soundness of the Frame Rule, in terms of an equivalence relation between states. However, we see that it would be necessary to completely renounce to address arithmetic and even modify the semantics of arithmetic and boolean expressions, which would be a very unnatural thing to do.

As we already announced, our first approach to the bounded memory model will not allow allocation of several contiguos cells, but this is fixed in 3.2, where instead of reserving just some number of cells with no predefined shape, we reserve a set of disjoint contiguous blocks of inactive cells — again with no specific address names. The representation of these gaps is much more complex to handle than a single natural number, and the task of giving a sensible operational semantics for memory allocation becomes challenging. Here, we suggest one and prove that it is sound.

We will also approach the notion of arity of a resource from a quite different point of view: that of a model of (partial) hydrocarbon molecules. This model represents objects with a structure much more complex than the structure of the memory model: partial hydrocarbon molecules can be viewed as graphs with dangling edges. The arity of such a molecule will be given by its set of dangling edges, since they are the connection points that we can use to create new bonds between atoms. We give here a syntax for assertions describing this model, define a simple language of commands for manipulating such molecules and provide tight specifications for them and prove soundness of these specifications. We also prove Safety Monotonicity, Frame Property and soundness of the Frame Rule.

The structure of the remainder of this document is the following: in section 2 we introduce the basic notions of separation logic (reasoning about the standard heap memory model), using [1, 2] as main references. In section 3 we explore the three approaches to the bounded memory model previously introduced (sections 3.1, 3.1.5 and 3.2) and the already mentioned partial hydrocarbon model (section 3.3). To finish, we will present some ideas for future work.

2 Background

In this subsection we present the basic notions given in [1, 2] about separation logic based on the standard heap memory resource. We define the memory model with its combination operation, and give a few commands to manipulate the memory, together with an operational semantics for them. Then we give the syntax of the assertions used to describe the memory and reason about it, and also their semantics. After this, we introduce Hoare reasoning with separation logic, and with it some important notions such as *safety* and *correctness*. We present the Frame Rule and consider the issues of its soundness and completeness. In particular, we introduce Safety and Termination Monotonicity and the Frame Property, properties they ensure the *local* behaviour that we need from our commands and that are the key to soundness of the Frame Rule.

2.1 The Memory Heap Model

Definition 2.1 (Dynamic Memory Model)**.** We consider stores or stacks that assign a value (an element in $Values$) to each of the elements in a fixed set *Variables* of variables (considered as infinite, for simplicity).

$$Stacks \stackrel{def}{=} Variables \to Values$$

We have chosen to use here $Values = \mathbb{Z} \cup \{\mathbf{nil}\}$, as opposed to $Values = \mathbb{Z}, \mathbf{nil} \in \mathbb{Z}$ used by Reynolds [1]. Results in both [1] and [2] work just the same, but we find that it is better to keep the value **nil** outside $\mathbb{Z}$ to prevent a strange behaviour of evaluation of arithemtic expressions.

Let *Addresses* be the set of cell addresses and *Atoms* the set of atomic values, with

$$Atoms \cup Addresses \subseteq \mathbb{Z}$$

$$Atoms \cap Addresses = \emptyset$$

We consider in this section that *Addresses* is an infinite subset of $\mathbb{Z}$ with unrestricted address arithmetic allowed on its elements. We also consider, as does [1], a set *Addresses* such that, whichever addresses are already allocated, we can always find an arbitrarily big set of contiguous unallocated cells.

We consider heaps as partial, finite mappings from the set of addressable memory cells into values.

$$Heaps \stackrel{def}{=} Addresses \rightharpoonup_{fin} Values$$

We will write $dom(h)$ for the domain of heap h, and e for the heap with empty domain.

Finally, memory states will be considered as some stack, together with some heap:

$$States \stackrel{def}{=} Stacks \times Heaps$$

The combination operation $\cdot$ between heaps is the following: given $h, h' \in Heaps$, $h \cdot h'$ is the union of both heaps when their domain is disjoint. This combination operation is commutative and associative.

2.1.1 Syntax and Semantics of Commands

We will use a small language of commands that manipulate our states. Reynolds [1] considers the simple imperative language originally axiomatized by Hoare ([4, 5]), plus a few more commands which enable us to perform standard memory manipulating tasks like lookup, update, allocation and disposal of memory. Their syntax is given in Figure 1, together with the syntax and semantics of arithmetic and boolean expressions.

$$
\begin{array}{rcl}
C & ::= & x := E \mid x := [E] \mid x := \mathbf{new}(E_1, \ldots, E_n) \mid [E] := E' \mid \mathbf{dispose}(E) \mid C;C \\
 & & \mid \mathbf{while}\ B\ \mathbf{do}\ C \mid \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C'
\end{array}
$$

$$E ::= x, y, ... \mid 0 \mid 1 \mid E + E' \mid E - E' \mid E \times E'$$

$$B ::= \mathbf{false} \mid B \rightarrow B' \mid E = E' \mid E < E'$$

$$
\begin{array}{rcll}
[\![x]\!]s & = & s(x) \text{ for any variable } x & \\
[\![0]\!]s & = & 0 & \\
[\![1]\!]s & = & 1 & \\
[\![E\ op_a\ E']\!]s & = & [\![E]\!]s\ \overline{op_a}\ [\![E']\!]s & \text{for any arithmetic operator } op_a \text{, with } \overline{op_a} \text{ being the corresponding semantic operator} \\
 & & & \\
[\![\mathbf{false}]\!]s & = & \mathbf{false} & \\
[\![B \rightarrow B']\!]s & = & \begin{cases} \mathbf{true} \text{ if whenever } [\![B]\!]s = \mathbf{true} \text{ also } [\![B']\!]s = \mathbf{true} \\ \mathbf{false} \text{ otherwise} \end{cases} & \\
[\![E\ op_b\ E']\!]s & = & [\![E]\!]s\ \overline{op_b}\ [\![E']\!]s & \text{for any boolean operator } op_b \text{, with } \overline{op_b} \text{ being the corresponding semantic operator}
\end{array}
$$

Figure 1: Syntax of commands, arithmetic and boolean expressions, and semantics of arithmetic and boolean expressions for the standard memory heap model

Arithmetic expressions are interpreted as elements of $Stacks \rightharpoonup Values$, that is, partial functions mapping stacks to values. We consider them partial because the presence of the special value **nil** can cause some arithmetic expression not to be interpretable as values under certain stacks. Boolean expressions are elements of $Stacks \rightharpoonup \{\mathbf{true}, \mathbf{false}\}$ (partial too, since they sometimes require evaluation of arithmetic expressions). Note that the semantics of both arithmetic an boolean expressions does not depend on the heap, but solely on the stack.

Let *configurations* be objects of the form $\langle C, (s, h)\rangle$ (a state together with a command), $\langle (s', h')\rangle$ or `fault`, with the last two being *terminal configurations*. The terminal configuration `fault` denotes a memory failure.

The behaviour of the commands can be observed in their operational semantics (Figure 2). It is a one step operational semantics, given as a binary relation $\rightsquigarrow$ between configurations.

Notation: given a mapping $f : A \longrightarrow B$, $f[x \mapsto v]$ denotes the mapping with domain $A \cup \{x\}$ that assigns v to x and the same value as f to any other element in A. When $A \subseteq A'$, $f\rceil_{A'}$ denotes the mapping that results from restricting the domain of f to A'.

It is stressed in [1] that, even though the same := notation is used in some of them, none of the new commands is an instance of the original assignment command $x := E$: they involve interaction with the heap, while $x := E$ only interacts with the stack.

A particular feature of this programming language is that everything, except for allocation of new memory addresses, takes place in a deterministic way: the memory address to be consulted, mutated or deallocated is mentioned explicitly. Hence, even if the heap is extended, this will never have a negative effect on the computation — that is, it will not cause a memory fault. This *local* behaviour of the commands is something very desirable. On the other hand, allocation of memory happens in a nondeterministic way: any inactive memory address can be chosen. We will see, later in this section, how this is important.

$$\overline{\langle x := E, (s,h)\rangle \rightsquigarrow (s[x \mapsto [\![E]\!]s], h)}$$

$$\frac{[\![E]\!]s = n \in dom(h) \quad h(n) = m}{\langle x := [E], (s,h)\rangle \rightsquigarrow \langle (s[x \mapsto m], h)\rangle} \quad \frac{[\![E]\!]s \notin dom(h)}{\langle x := [E], (s,h)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{[\![E]\!]s = n \in dom(h)}{\langle [E] := E', (s,h)\rangle \rightsquigarrow \langle (s, h[n \mapsto [\![E']\!]s])\rangle} \quad \frac{[\![E]\!]s \notin dom(h)}{\langle [E] := E', (s,h)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{m, \dots, m+n-1 \notin dom(h) \quad v_1 = [\![E_1]\!]s, \dots, v_n = [\![E_n]\!]s}{\langle x := \mathbf{new}(E_1, \dots, E_n), (s,h)\rangle \rightsquigarrow \langle (s[x \mapsto m], h[m \mapsto v_1, \dots, m+n-1 \mapsto v_n])\rangle}$$

$$\frac{[\![E]\!]s = n \in dom(h)}{\langle \mathbf{dispose}(E), (s,h)\rangle \rightsquigarrow \langle (s, h\rceil_{dom(h)\setminus\{n\}})\rangle} \quad \frac{[\![E]\!]s \notin dom(h)}{\langle \mathbf{dispose}(E), (s,h)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{\langle C, (s,h)\rangle \rightsquigarrow \langle C'', (s',h')\rangle}{\langle C; C', (s,h)\rangle \rightsquigarrow \langle C''; C', (s',h')\rangle} \quad \frac{\langle C, (s,h)\rangle \rightsquigarrow \langle (s',h')\rangle}{\langle C; C', (s,h)\rangle \rightsquigarrow \langle C', (s',h')\rangle}$$

$$\frac{\langle C, (s,h)\rangle \rightsquigarrow \texttt{fault}}{\langle C; C', (s,h)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{[\![B]\!]s = true}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s,h)\rangle \rightsquigarrow \langle C, (s,h)\rangle} \quad \frac{[\![B]\!]s = false}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s,h)\rangle \rightsquigarrow \langle C', (s,h)\rangle}$$

$$\frac{[\![B]\!]s = false}{\langle \mathbf{while}\ B\ C, (s,h)\rangle \rightsquigarrow \langle (s,h)\rangle} \quad \frac{[\![B]\!]s = true}{\langle \mathbf{while}\ B\ C, (s,h)\rangle \rightsquigarrow \langle C; \mathbf{while}\ B\ C, (s,h)\rangle}$$

Figure 2: Operational semantics of commands with respect to the memory model

Something else that can be observed in the operational semantics is how running a program that refers to a memory address that is not active — i.e. not in the domain of the current state's heap — will cause the program to fault. However, if every needed memory cell is active, the program will never fault. Remember that we assume an infinite number of cells always available, and even a set of inactive contiguous cells as big as we want. Therefore, memory allocation can never lead to `fault` in this setting.

2.1.2 Syntax and Semantics of Program Assertions

We introduce here a language of assertions for reasoning about properties of memory states. Syntax and semantics of the assertion language for the model are given below. The main new operator introduced is $*$ (separating conjunction). Its purpose is to express in an explicit way disjointness between two portions of resource. Some properties of $*$ are commutativity, associativity and having **emp** as its neutral element.

$$\begin{array}{rcl} A & ::= & \top \mid \bot \mid B \mid \neg A \mid A \wedge A' \mid A \vee A' \mid A \rightarrow A' \mid \exists x.\ A \mid \forall x.\ A \mid \\ & & \mathbf{emp} \mid E \mapsto E' \mid A * A' \mid A \mathrel{-\!\!*} A' \end{array}$$

$$[\![A]\!] \stackrel{def}{=} \{(s,h) \mid (s,h) \models A\}$$

The relation denoted by $\models$ between a state and an assertion, with $(s,h) \models A$ denoting satisfaction of the assertion A by the state (s,h), is given in Figure 3.

Satisfaction relation

$$
\begin{array}{rcl}
(s,h) \models \top & \text{always} & \\
(s,h) \models \bot & \text{never} & \\
(s,h) \models B & \Leftrightarrow & [\![B]\!]s = \mathit{true} \\
(s,h) \models \neg A & \Leftrightarrow & (s,h) \not\models A \\
(s,h) \models A \wedge A' & \Leftrightarrow & (s,h) \models A \text{ and } (s,h) \models A' \\
(s,h) \models A \vee A' & \Leftrightarrow & (s,h) \models A \text{ or } (s,h) \models A' \\
(s,h) \models A \rightarrow A' & \Leftrightarrow & (s,h) \models A \text{ implies } (s,h) \models A' \\
(s,h) \models \exists x.\ A & \Leftrightarrow & \text{for some } v \in \mathit{Values}\ (s[x \mapsto v], h) \models A \\
(s,h) \models \forall x.\ A & \Leftrightarrow & \text{for every } v \in \mathit{Values}\ (s[x \mapsto v], h) \models A \\
(s,h) \models \mathbf{emp} & \Leftrightarrow & \mathit{dom}(h) = \emptyset \\
(s,h) \models E \mapsto E' & \Leftrightarrow & \mathit{dom}(h) = \{[\![E]\!]s\} \text{ and } h([\![E]\!]s) = [\![E']\!]s \\
(s,h) \models A * A' & \Leftrightarrow & \exists h_1, h_2.\ h = h_1 \cdot h_2,\ (s,h_1) \models A \text{ and } (s,h_2) \models A' \\
(s,h) \models A \mathrel{-\!\!*} A' & \Leftrightarrow & \forall h'.\ \text{if } (s,h') \models A \text{ and } h \perp h' \text{ then } h \cdot h' \models A'
\end{array}
$$

Figure 3: semantics and satisfaction relation for assertions

2.1.3 Hoare Reasoning with Separation Logic

Since the goal of developing Separation Logic is to be able to reason about *correctness* of programs, it is compulsory that we define the notion of correctness that will be considered. In the heap model explored in [1, 2] two notions of correctness are given: *partial* and *total* correctness.

Let $\leadsto^*$ be the transitive closure of $\leadsto$ in what follows.

Definition 2.2. A configuration $\langle C, (s,h)\rangle$ is *safe* iff $\langle C, (s,h)\rangle \not\leadsto^* \texttt{fault}$.

A configuration $\langle C, (s,h)\rangle$ *must terminate normally* iff it is safe and there is no infinite $\leadsto$-sequence starting from $\langle C, (s,h)\rangle$.

Definition 2.3 (Partial Correctness)**.** $\{A\}\ C\ \{A'\}$ holds iff

$$
\begin{array}{l}
\forall (s,h) \in \mathit{States}.\ (s,h) \models A \text{ implies} \\
\quad \langle C, (s,h)\rangle \text{ is safe and} \\
\quad \forall (s',h') \in \mathit{States}.\ \langle C, (s,h)\rangle \leadsto \langle (s',h')\rangle \text{ implies } (s',h') \models A'
\end{array}
$$

Definition 2.4 (Total Correctness)**.** $[A][C][A']$ holds iff

$$
\begin{array}{l}
\forall (s,h) \in \mathit{States}.\ (s,h) \models A \text{ implies} \\
\quad \langle C, (s,h)\rangle \text{ must terminate normally and} \\
\quad \forall (s',h') \in \mathit{States}.\ \langle C, (s,h)\rangle \leadsto \langle (s',h')\rangle \text{ implies } (s',h') \models A'
\end{array}
$$

We will focus here on partial correctness.

One thing important here, as observed in [2], is that we can be sure that, if $\{A\}\ C\ \{A'\}$ holds, then in every state satisfying A the heap has enough resource for C to run safely. We use a *fault-avoiding* notion of correctness. So, in particular, nothing that is not explicitly mentioned in A will be needed by C, and we can be sure that there are no side effects of C — we do not need to say which cells remain the same, they all do, unless otherwise stated. That is: any cell in the initial state that is not explicitly mentioned will remain unchanged. This is a very nice feature for scalability.

The example given in [2] for this is very simple and clear: suppose we have the triple

$$\{x \hookrightarrow 5\}\ C\ \{x \hookrightarrow 6\}$$

and we know that C does not modify variables. Then no memory cell in the initial heap other than the one with addres x will be modified by C for, in particular, the state where the heap contains one single cell at address x with contents 5 is an acceptable starting state, and trying to dereference any other (inactive) memory address would result in faulting. Hence, only x and memory addresses allocated by C can be modified by C, according to this specification. Note that this is not only a consequence of our notion of correctness, but also of the chosen set of commands: it would not be true anymore if we had some command that could deallocate every allocated cell without mentioning it explicitly.

Also, since this notion of correctness avoids faulting, we can be sure that if $\{A\}\ C\ \{A'\}$ holds, it will never do something like dereferencing a cell that has previously been deallocated.

Finally, let us remark that this fault-avoiding condition refers to *all* possible computations of C on *any* initial state satisfying A. In this setting, as we already said, a command will never fault if it has the memory resource it needs — it will surely fault if it does not have it. Although there is nondeterminism in the allocation of new memory and computations can sometimes happen in various different ways, we need all of these possible computations to behave nicely. We are interested, as pointed out in [2], in *classes of computations*.

The original command-specific inference rules of Hoare Logic, such as

$$\frac{}{\{A[E/x]\}\ x := E\ \{A\}} \quad (\textit{Assignment})$$

($A[E/x]$ denotes substitution of any occurrence of x in A for E)

$$\frac{\{A\}\ C_1\ \{D\} \quad \{D\}\ C_2\ \{F\}}{\{A\}\ C_1; C_2\ \{F\}} \quad (\textit{Sequential Composition})$$

are still sound in this setting, and also some others like Consequence and Substitution:

$$\frac{[\![A']\!] \subseteq [\![A]\!] \quad \{A\}\ C\ \{D\} \quad [\![D]\!] \subseteq [\![D']\!]}{\{A'\}\ C\ \{D'\}} \quad (\textit{Consequence})$$

$$\frac{\{A\}\ C\ \{A'\}}{(\{A\}\ C\ \{A'\})[E_1/x_1, \ldots, E_n/x_n]} \quad (\textit{Substitution})$$

(where $x_1, \ldots, x_n$ are the variables occurring free in A, C or A', and if x_i is modified by C then E_i is a variable that does not occur free in any other x_j.)

(Note: It is worth mentioning that, despite remaining sound in this setting, the rule

$$\frac{}{\{A[E/x]\}\ x := E\ \{A\}}$$

does not match the spirit of separation logic, for it is not *tight*. The following rule will be used instead:

$$\frac{}{\{x \doteq y\}\ x := E\ \{x \doteq E[y/x]\}}$$

with $x \doteq y$ being an abbreviation for $x = y \wedge \mathbf{emp}$.)

However, the following rule

$$\frac{\{A\}C\{D\}}{\{A \wedge F\}C\{D \wedge F\}}$$

is not sound any more. Here is a counterexample that reflects this fact:

$$\frac{\{x \mapsto -\}[x] := 3\{x \mapsto 3\}}{\{x \mapsto - \wedge \neg(x \mapsto 3)\}[x] := 3\{x \mapsto 3 \wedge \neg(x \mapsto 3)\}}$$

It is clear that the postcondition in the conclusion can never hold, even though the precondition might do.

In spite of losing soundness for this rule, this setting allows a new, much more powerful rule, that allows us to extend local specifications and is the true heart of Separation Logic: the Frame Rule

$$\frac{\{A\}\ C\ \{A'\}}{\{A * D\}\ C\ \{A' * D\}}$$

Let us consider the following refinement of this basic version of the rule, given in [2]:

$$\frac{\{A\}\ C\ \{A'\}}{\{A * D\}\ C\ \{A' * D\}}\ Modifies(C)\#D$$

where $Modifies(C)$ is the set of variables updated by C and the relation $\#$ between a set of variables X and an assertion A is defined so that $X\#A$ iff the semantics of the assertion A is independent from the values assigned to the variables in X. That is $X\#A$ iff for any $s, s' \in Stacks$ with $s\rceil_{Variables \setminus X} = s'\rceil_{Variables \setminus X}$ we have $(s,h) \models A \Rightarrow (s',h) \models A$.

The Frame Rule enables us to extend any specification $\{A\}\ C\ \{A'\}$ — where A might describe a limited portion of memory — to a bigger initial state $A * D$, and explicitly state that the extended part of the heap (disjoint with the original heap) remains unchanged — as long as C does not change variables free in D.

This restriction is necessary because, even though the heaps for A and D (and for A' and D) are disjoint, the variable stack is shared completely. If there was some variable free in D whose value is changed by C, then the heap would be changed, and we want to avoid that.

So we can give *tight* specifications of commands in terms, exclusively, of their *footprint* — the precise variables and portions of the heap that the command does use — and it is implicit that everything else will remain unchanged. The Frame Rule, combined with others, will then allow us to obtain specifications of larger programs (made by combining our basic commands) and extend them to the whole memory heap in which we will actually run the programs.

We can see an example of these tight specifications in the inference rule for mutation proposed in [1]:

$$\frac{}{\{e \mapsto -\}\ [e] := e'\ \{e \mapsto e'\}}$$

Thanks to the Frame Rule, any such specification can then be extended to a state with a bigger heap, in this case

$$\frac{}{\{(e \mapsto -) * A\}[e] := e'\{(e \mapsto e') * A\}}$$

or even given in form of backwards reasoning, in terms of a postcondition D, in this case by taking $A = (e \mapsto e') -\!\!* D$ and using the valid implication $F * (F -\!\!* D) \Rightarrow D$:

$$\frac{}{\{(e \mapsto -) * ((e \mapsto e') -\!\!* D)\}[e] := e'\{D\}}$$

Figure 4 contains a whole set of axioms and rules we can use to reason about program specifications in the standard memory model.

$$\overline{\{x \doteq y\}\ x := E\ \{x \doteq E[y/x]\}}$$

$$\overline{\{x = y \wedge (E \mapsto z)\}\ x := [E]\ \{x = z \wedge (E[y/x] \mapsto z)\}}\ \ x, y, z \text{ distinct}$$

$$\overline{\{\exists x.\ (E \mapsto x)\}\ [E] := F\ \{E \mapsto F\}}$$

$$\overline{\{x \doteq y\}\ x := new(E)\ \{x \mapsto E[y/x]\}}$$

$$\overline{\{\exists x.\ (E \mapsto x)\}\ dispose(E)\ \{\mathbf{emp}\}}$$

$$\frac{\{A\}\ C_1\ \{A''\} \quad \{A''\}\ C_2\ \{A'\}}{\{A\}\ C_1; C_2\ \{A'\}}$$

$$\frac{[\![A_1]\!] \subseteq [\![A'_1]\!] \quad \{A'_1\}\ C\ \{A'_2\} \quad [\![A'_2]\!] \subseteq [\![A_2]\!]}{\{A_1\}\ C\ \{A_2\}}$$

$$\frac{\{A \wedge B\}\ C_1\ \{A'\} \quad \{A \wedge \neg B\}\ C_2\ \{A'\}}{\{A\}\ \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \{A'\}}$$

$$\frac{\{A \wedge B\}\ C\ \{A\}}{\{A\}\ \mathbf{while}\ B\ \mathbf{do}\ C\ \{A \wedge \neg B\}}$$

Figure 4: Rules for the Standard Memory Heap Model

Now that we know how the Frame Rule looks and works, we can understand why it is so important that memory allocation is done in a nondeterministic way. Imagine this was not the case, and the portion of memory to be allocated by a program C executed from a state described by A was deterministically fixed. Suppose $\{A\}C\{A'\}$ and C does not deallocate any memory. Then the memory to be allocated and the memory heap in A must be disjoint. Let D describe a state in which the heap is disjoint with the heap in A, but not with the portion of memory that C is going to allocate. $A * D$ would hold in the initial state, but $A' * D$ would not hold in the final state, since the heaps that A' and D describe are not disjoint. This is explained in very simple and clear way in [2]: "An address that is allocated during an execution from a small state cannot be allocated starting in a bigger state where it is already active".

2.1.4 Frame Rule: Soundness

As we announced previously, there are two properties in this setting that are really important and have been isolated in [2] as responsible for soundness of the Frame Rule: Safety and Termination Monotonicity and the Frame Property. Since the Frame Rule is what allows us to use tight specifications when working on correctness of programs, it is very important that it is sound, so these properties have a central role. We will speak of Safety Monotonicity only,

here, instead of Safety and Termination Monotonicity, since we are focusing only on partial correctness.

The properties are formulated like this:

Lemma 2.1 (Safety Monotonicity).
If $\langle C, (s, h)\rangle$ *is safe and* $h \cdot h' \in Heaps$ *then* $\langle C, (s, h \cdot h')\rangle$ *is safe.*

Lemma 2.2 (Frame Property).
If $\langle C, (s, h_0)\rangle$ *is safe and* $\langle C, (s, h_0 \cdot h_1)\rangle \rightsquigarrow^* \langle (s', h')\rangle$*, then there exists some* h'_0 *with* $\langle C, (s, h_0)\rangle \rightsquigarrow^* \langle (s', h'_0)\rangle$ *and* $h' = h'_0 \cdot h_1$*.*

We omit here the details of the proofs, which are given in [2]. So is the following theorem:

Theorem 2.1 (Soundness of the Frame Rule). *The Frame Rule is sound for both partial and total correctness.*

There is an interesting remark in [2] about how the Frame Property is formulated. It could have been (wrongly) formulated like this:

If $\langle C, (s, h_0)\rangle \rightsquigarrow^* (s', h'_0)$ *and* $h_0 \perp h_1$*, then* $\langle C, h_0 \cdot h_1\rangle \rightsquigarrow^* (s', h'_0 \cdot h_1)$*.*

This actually does not hold because C, running on (s, h_0), might have allocated memory that is already active in h_1. We have already seen how this would be a problem. Hence, we cannot go from any computation on a small state to one on a bigger state, because issues like this might arise. What the Frame Property says is slightly different. It says that "if a command is safe in a given state, then the result of executing it in a larger state can be tracked to *some* execution computation on the little state" ([2]). Hence, instead of going from a small state to a bigger one, what we need is to be able to go from the bigger one back to the small, as long as it is safe.

2.1.5 Frame Rule: Completeness

One way of understanding completeness of the Frame Rule is that, whenever a specification statement for some command is a semantic consequence of another (for the same command), it is possible to derive the first from the second using the Rule of Consequence and the Frame Rule. Even though we do not look into completeness in the following sections (although it will definitely be something interesting to study in the future), we give here an overall idea of how it is formally enunciated by Yang and O'Hearn in [2]. We will not go into details about how they also prove it.

Instead of reasoning in terms of standard Hoare triples, Yang and O'Hearn use for this purpose Hoare triples with an unspecified command, $\{A\} - \{A'\}$, and a fixed set X of variables, which are taken to be the variables modified by the "ghost" command. In this setting, the Frame Rule looks like this

$$\frac{\{A\} \ - \ \{A'\}}{\{A * D\} \ - \ \{A' * D\}} \ X \# D$$

(we already defined the relation # in subsection 2.1.3 when we introduced the Frame Rule).

Enunciating completeness requires first of all defining a notion of semantic consequence between specification statements. This is done using *predicate transformers* (defined below), and in particular a subclass of predicate tranformers which verify a locality property — like our commands did.

To avoid changing the name of *predicate transformers* — since we are talking here in terms of assertions and their interpretations and not in terms of predicates — let us consider here the following definition:

Definition 2.5 (Predicates)**.** Let the set of all predicates be

$$\texttt{Pred} \stackrel{def}{=} \mathcal{P}(States)$$

(the powerset of $States$).

Note that, given any assertion A, $[\![A]\!] \in \mathcal{P}(States)$ is a predicate.

Definition 2.6 (Predicate Transformers)**.** A predicate transformer is a monotone mapping from predicates to predicates:

$$\texttt{PT} \stackrel{def}{=} \texttt{Pred} \rightarrow_{monotone} \texttt{Pred}$$

$t \in \texttt{PT}$ monotone means that if $P \subseteq Q$ then $t(P) \subseteq t(Q)$.

A correspondence is established in [2] between predicate transformers and the commands in the considered programming language. We will not give here the formal correspondence, but we explain how it works. Let us (in a small abuse of notation) refer momentarily with the same identifier t to some predicate transformer and its corresponding command. Then if we have $\{A\}\ t\ \{A'\}$ (for the *command* t), that is equivalent to $[\![A]\!] \subseteq t([\![A']\!])$ (for the *predicate transformer* t). As we can see, predicate transformers modify a postcondition into a precondition.

Not every predicate transformer can be seen as corresponding to one of our commands. In fact, some of these predicate transformers do not even have the nice locality properties that all our commands had. We are going to consider only a subset of PT: only those predicate transformers verifying a certain locality condition with respect to our fixed set X of "modified" variables. We want our predicate transformers to behave in such a way that having them act on some heap gives us at least all the states that we get by restricting them to act on a smaller part of the heap, and then extending the heap again with the portion that we had removed (as long as the part removed is independent of all variables affected by the predicate transformer).

Given $P, R \in \texttt{Pred}$, let $P * R = \{(s, h * h') \mid (s, h) \in P \text{ and } (s, h') \in R\}$.

In an abuse of notation, we will # to represent a relation between sets of variables and predicates — remember that we use the same notation to represent a relation (very similar to this one) between sets of variables and assertions. Given $P \in \texttt{Pred}$, we will have $X\#P$ iff $(s, h) \in P$ does not depend on the value that s assigns to variables in X, that is

$$X\#P \text{ iff } (s,h) \in P \text{ implies } \forall s'.\ (s'\rceil_{dom(s')\setminus X} = s\rceil_{dom(s)\setminus X} \Rightarrow (s', h) \in P)$$

Definition 2.7 (Locality for X)**.** We will say that the predicate transformer t satisfies locality for X iff

$$\forall P, Q \in \texttt{Pred}.\ X\#P \Rightarrow t(Q) * P \subseteq t(Q * P)$$

The subset of all local predicates satisfying this locality condition for X is $\texttt{LPT}(X)$.

This locality condition is equivalent to saying that (the predicate tranformer) t satisfies the Frame Rule, given in terms of predicate transformers as

$$\forall P, Q, R \in \texttt{Pred}.\ X\#R \wedge P \subseteq t(Q) \Rightarrow P * R \subseteq t(Q * R)$$

With the aid of $\texttt{LPT}(X)$, Yang and O'Hearn define a semantic consequence relation between pairs of specification statements:

$$\{A\} - \{D\} \models_X \{A'\} - \{D'\} \quad \text{iff} \quad \begin{array}{l} \text{for every } t \in \texttt{LPT(X)}, \\ t \models_X \{A\} - \{D\} \text{ implies } t \models_X \{A'\} - \{D'\} \end{array}$$

where $t \models_X \{A\} - \{D\}$ iff $[\![A]\!] \subseteq t([\![D]\!])$.

And, finally, let

$$\{A\} - \{D\} \vdash_X \{A'\} - \{D'\} \quad \text{iff} \quad \begin{array}{l} \{A'\} - \{D'\} \text{ can be derived from } \{A\} - \{D\} \\ \text{using the rule of Consequence and the Frame Rule} \end{array}$$

With all the necessary elements now introduced, we have the following formal enunciation of completeness:

Theorem 2.2 (Completeness of the Frame Rule)**.**

$$\textit{If } \{A\} - \{D\} \models_X \{A'\} - \{D'\} \textit{ then } \{A\} - \{D\} \vdash_X \{A'\} - \{D'\}$$

And even soundness can be re-enunciated as its converse:

Theorem 2.3 (Soundness of the Frame Rule)**.**

$$\textit{If } \{A\} - \{D\} \vdash_X \{A'\} - \{D'\} \textit{ then } \{A\} - \{D\} \models_X \{A'\} - \{D'\}$$

3 Resources with arity

The main feature of all the resources that can be reasoned about in a separation logic style is their combination operation. Whether it is partial, total, deterministic or nondeterministic, it is what makes it possible for us to take a large resource instance, split it into smaller portions and focus on these when we do our reasoning, instead of dealing with the whole big instance. It also allows us to take some resource instance and make it bigger, either by framing another piece of resource onto it, or following instructions from some programming command — like the memory allocation command $x := \mathbf{new}(E_1, \ldots, E_n)$ in the standard memory model presented in section 2.

As we mentioned in the introduction, in the original memory model it is assumed that the set *Addresses* is infinite. However, if we suppose that it is finite, then we might have to face the case when all of them have already been allocated, and in this case any command that tried to allocate new memory would lead to `fault`. We will refer to the ability of a memory state to be extended as its *arity*, since it can be described numerically. We will see later how this notion can be extended to a different kind of resource.

Even within the bounded memory model we can approach arity in different ways. For example, in our first approach to arity within this model, we will regard it as just the amount of cells that are yet to be allocated, with allocation taking place for one cell at a time. As we just said, in order for $x := \mathbf{new}(E)$ not to lead to `fault`, we need to make sure that there will be at least one cell that has not been allocated yet. We need to include this information in the states in such a way that safety is preserved by framing (remember the definition of safety given in section 2: a configuration $\langle C, (s, h)\rangle$ is safe iff it can never lead to `fault`). That is we need to make sure that Safety Monotonicity holds, otherwise the Frame Rule cannot possibly be sound. With this aim, we will add a third component to our states, which will be a natural number and will represent a number of *reserved* cells. This is, a number of inactive cells over which we have permission, and which we can safely allocate, regardless of their precise addresses. Since memory allocation takes place in a nondeterministic fashion, it seems natural that we should not care about the addresses of the cells held reserved. On the contrary, we treat all free cells as balls in a big bag, and our only concern is that nobody else takes too many balls from that bag. As long as they leave enough balls for us — at least as many as we had reserved — we do not care which ones they take. Furthermore, when we look into the alternative approach of taking this new third component as a properly specified set of cells addresses, instead of just a number, we realize that it brings problems with the Frame Property. We will see why this happens, later in this section.

The first approach just described, with only a natural number as the new component of states, is a very natural extension of the standard states and is quite easy to work with. However, it does not allow us to allocate several contiguous cells (with a command like $x := \mathbf{new}(E_1, \ldots, E_n)$), for information just about some amount of cells is not enough information to know their relative positions inside the global memory heap. We attempt to fix this in yet another approach, taking the new third component of states as a set of natural numbers, which represent the sizes of some disjoint contiguous gaps in the memory heap — not necessarily describing all the gaps, just some gaps we have permission to use. We give here an operational semantics for allocation and deallocation of memory in this model, and prove that the one given for allocation is sensible, which is not a trivial task. Due to time restrictions, Safety Monotonicity and Frame Property have only been conjectured here.

As for other kinds of resources, to finish, we will look into a partial hydrocarbon model, which provides a simplified representation of partial hydrcarbon molecules by means of graphs with labelled nodes representing atoms (labels being `H` for hydrogen and `C` for carbon) and a restricted

amount of edges attached to each node, in accordance to its label. These edges can either link two nodes, or be dangling, as if we had taken a static picture of some intermediate moment during a chemical reaction, when some bonds between atoms have been broken and others are yet to be formed. It is fairly easy to imagine the notion of arity considered here: it is but the number of dangling edges coming out of nodes of each kind. A mechanism to preserve a certain arity when framing — again, to ensure, not Safety Monotonicity, but Liveness Monotonicity, a similar property suitable for this setting — is used here as well: distinguishing between simply free edges, and reserved ones.

3.1 The Bounded Heap Model

As we announced, our first approach to the bounded memory model, and to dealing with arity as one of its features, will be adding a new component to our states: a natural number that will represent a number of inactive cells held reserved for manipulation with commands.

In this approach, *Stacks* and *Heaps* (and *Addresses*) are taken as defined in section 2, only now *Addresses* can be a finite set, with $\mathtt{MAX} = |Addresses|$. *States* have now the following shape:

$$States \stackrel{def}{=} \{(s,h,n) \in Stacks \times Heaps \times \mathbb{N} \text{ such that } |dom(h)| + n \leq \mathtt{MAX}\}$$

The combination operation is now

$$(s,h,n) \cdot (s,h',n') = (s, h \cdot h', n + n')$$

whenever $(s, h \cdot h', n + n') \in States$ (in particular $h \cdot h' \in Heaps$), and undefined otherwise. In order to combine two states, we need the two heaps to be disjoint, but we also need the total amount of cells in the state not to exceed the number of cells in *Addresses*.

The commands will be exactly the ones presented in the introduction, except that $x := \mathbf{new}(E_1, \ldots, E_n)$ now becomes just $\mathbf{new}(E)$ — we will only allocate new memory cells one by one. Arithmetic and boolean expressions will be the same as in section 2, and so will their semantics.

The revised operational semantics is in Figure 5. It reflects how the memory allocation command will now `fault` whenever running on a state that does not possess some free cell. It also shows how allocation and deallocation of memory affect our new third component.

3.1.1 Assertion Language

To perform reasoning tasks with this model we need to make some changes to the assertion language, in order for it to be able to express information about the free cells being held.

$$\begin{array}{rcl} A & ::= & \top \mid \bot \mid B \mid \neg A \mid A \wedge A' \mid A \vee A' \mid A \rightarrow A' \mid \exists x.\ A \mid \forall x.\ A \mid \\ & & \mathbf{emp} \mid E \mapsto E' \mid E \mapsto E', E'' \mid A * A' \mid A \mathrel{-\!\!*} A' \mid A^E \end{array}$$

The satisfaction relation between states and assertions is given in Figure 6. We can see in that figure how the annotation E in the basic construct A^E indicates some lower bound to the number of free cells held reserved in the state.

There is a close relation between this new assertion language and the one given in section 2. This relation is expressed here in terms of a mapping $\lfloor\cdot\rfloor$, which turns assertions of the kind

$$\frac{}{\langle x := E, (s,h,n)\rangle \rightsquigarrow (s[x \mapsto [\![E]\!]s], h, n)}$$

$$\frac{[\![E]\!]s = n \in dom(h) \quad h(n) = m}{\langle x := [E], (s,h,n)\rangle \rightsquigarrow \langle (s[x \mapsto m], h, n)\rangle} \qquad \frac{[\![E]\!]s \notin dom(h)}{\langle x := [E], (s,h,n)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{[\![E]\!]s = k \in dom(h)}{\langle [E] := E', (s,h,n)\rangle \rightsquigarrow \langle (s, h[k \mapsto [\![E']\!]s], n)\rangle} \qquad \frac{[\![E]\!]s \notin dom(h)}{\langle [E] := E', (s,h,n)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{n > 0 \quad m \in Addresses \setminus dom(h) \quad v = [\![E]\!]s}{\langle x := \mathbf{new}(E), (s,h,n)\rangle \rightsquigarrow \langle (s[x \mapsto m], h[m \mapsto v], n-1)\rangle} \qquad \frac{n = 0}{\langle x := \mathbf{new}(E), (s,h,n)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{[\![E]\!]s = k \in dom(h)}{\langle \mathbf{dispose}(E), (s,h,n)\rangle \rightsquigarrow \langle (s, h\rceil_{dom(h)\setminus\{k\}}, n+1)\rangle} \qquad \frac{[\![E]\!]s \notin dom(h)}{\langle \mathbf{dispose}(E), (s,h,n)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{\langle C, (s,h,n)\rangle \rightsquigarrow \langle C'', (s',h',n')\rangle}{\langle C;C', (s,h,n)\rangle \rightsquigarrow \langle C'';C', (s',h',n')\rangle} \qquad \frac{\langle C, (s,h)\rangle \rightsquigarrow \langle (s',h',n')\rangle}{\langle C;C', (s,h,n)\rangle \rightsquigarrow \langle C', (s',h',n')\rangle}$$

$$\frac{\langle C, (s,h,n)\rangle \rightsquigarrow \texttt{fault}}{\langle C;C', (s,h,n)\rangle \rightsquigarrow \texttt{fault}}$$

$$\frac{[\![B]\!]s = true}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s,h,n)\rangle \rightsquigarrow \langle C, (s,h,n)\rangle} \qquad \frac{[\![B]\!]s = false}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s,h,n)\rangle \rightsquigarrow \langle C', (s,h,n)\rangle}$$

$$\frac{[\![B]\!]s = false}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s,h,n)\rangle \rightsquigarrow \langle (s,h,n)\rangle} \qquad \frac{[\![B]\!]s = true}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s,h,n)\rangle \rightsquigarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, (s,h,n)\rangle}$$

Figure 5: Operational semantics of commands with respect to the bounded heap model

just defined into asssertions with the original syntax — those that described states consisting just of a stack and a heap — by simply deleting every annotation in the expression.

We will need the following auxiliary lemma:

Lemma 3.1. *If every annotation occurring in A is 0 then either $\forall n.\ (s,h,n) \models A$ or $\forall n.\ (s,h,n) \not\models A$.*

Proof. We apply structural induction on A, not considering the cases $\bot, \vee, \rightarrow, \forall$, since they can be expressed in terms of $\top, \neg, \wedge, \exists$.

– $\top$

  Straightforwardly, $\forall n.\ (s,h,n) \models \top$.

– B

  Suppose there is at least one $n \in \mathbb{N}$ such that $(s,h,n) \models B$. Then $[\![B]\!]s = \mathbf{true}$ and so $(s,h,n') \models B$ for every $n' \in \mathbb{N}$.

– $\neg A$

  Suppose there is at least one $n \in \mathbb{N}$ such that $(s,h,n) \models \neg A$. Then $(s,h,n) \not\models A$. By induction hypothesis then we have that for every $n' \in \mathbb{N}$ it is $(s,h,n') \not\models A$, and so $\forall n.\ (s,h,n) \models \neg A$.

– $A \wedge A'$

  Suppose there is at least one $n \in \mathbb{N}$ such that $(s,h,n) \models A \wedge A'$. Then $(s,h,n) \models A$ and $(s,h,n) \models A'$, and so by structural induction we have $\forall n.\ (s,h,n) \models A$ and $\forall n.\ (s,h,n) \models A'$ and so $\forall n.\ (s,h,n) \models A \wedge A'$.

$$
\begin{array}{lcl}
(s,h,n) \models \top \text{ always} & & \\
(s,h,n) \models \bot \text{ never} & & \\
(s,h,n) \models B & \Leftrightarrow & [\![B]\!]s = \textbf{true} \\
(s,h,n) \models \neg A & \Leftrightarrow & (s,h,n) \not\models A \\
(s,h,n) \models A \wedge A' & \Leftrightarrow & (s,h,n) \models A \text{ and } (s,h,n) \models A' \\
(s,h,n) \models A \vee A' & \Leftrightarrow & (s,h,n) \models A \text{ or } (s,h,n) \models A' \\
(s,h,n) \models A \rightarrow A' & \Leftrightarrow & (s,h,n) \models A \text{ implies } (s,h,n) \models A' \\
(s,h,n) \models \exists x.\ A & \Leftrightarrow & \text{for some } v \in \mathit{Values} \text{ we have } (s[x \mapsto v],h,n) \models A \\
(s,h,n) \models \forall x.\ A & \Leftrightarrow & \text{for every } v \in \mathit{Values} \text{ we have } (s[x \mapsto v],h,n) \models A \\
(s,h,n) \models \mathit{emp} & \Leftrightarrow & \mathit{dom}(h) = \emptyset \\
(s,h,n) \models E \mapsto E' & \Leftrightarrow & \mathit{dom}(h) = [\![E]\!]s,\ \ h([\![E]\!]s) = [\![E']\!]s \\
(s,h,n) \models A * A' & \Leftrightarrow & \begin{array}{l}(s,h,n) = (s, h_1 \cdot h_2, n_1 + n_2) \text{ for some } h_1,h_2,n_1,n_2 \\ \text{with } (s,h_1,n_1) \models A \text{ and } (s,h_2,n_2) \models A'\end{array} \\
(s,h,n) \models A -\!\!* A' & \Leftrightarrow & \begin{array}{l}\forall h',n' \text{ if } (s,h',n') \models A \text{ and } (s,h\cdot h',n+n') \in \mathit{States} \\ \ \text{then } (s,h\cdot h',n+n') \models A'\end{array} \\
(s,h,n) \models A^E & \Leftrightarrow & (s,h,n) \models A \text{ and } n \geq [\![E]\!]s
\end{array}
$$

Figure 6: Semantics of assertions for the Bounded Heap model

– $\exists x.\ A$

Suppose there is at least one $n \in \mathbb{N}$ such that $(s,h,n) \models \exists x.\ A$. Then there is $v \in \mathit{Values}$ such that $(s[x \mapsto v],h,n) \models A$ and by structural induction we have $\forall n.\ (s[x \mapsto v],h,n) \models A$ and so $\forall n.\ (s,h,n) \models \exists x.\ A$.

– **emp**

Suppose there is at least one $n \in \mathbb{N}$ such that $(s,h,n) \models \textbf{emp}$. Then $\mathit{dom}(h) = \emptyset$ and so $(s,h,n') \models \textbf{emp}$ for every $n' \in \mathbb{N}$.

– $E \mapsto E'$

Suppose there is at least one $n \in \mathbb{N}$ such that $(s,h,n) \models E \mapsto E'$. Then $\mathit{dom}(h) = [\![E]\!]s$ and $h([\![E]\!]s) = [\![E']\!]s$ and $(s,h,n) \models E \mapsto E'$ for every $n \in \mathbb{N}$.

– $A * A'$

Suppose there is at least one $n \in \mathbb{N}$ such that $(s,h,n) \models A * A'$. Then for some h_1,h_2,n_1,n_2 it is $(s,h,n) = (s,h_1 \cdot h_2, n_1 + n_2)$ with $(s,h_1,n_1) \models A$ and $(s,h_2,n_2) \models A'$. But then by induction hypothesis $\forall n_1.\ (s,h_1,n_1) \models A$ and $\forall n_2.\ (s,h_2,n_2) \models A'$. This implies that $\forall n.\ (s,h,n) \models A * A'$ because given any $n' \in \mathbb{N}$ it is, for instance, $n' = n' + 0$ and we have $(s,h_1,n') \models A$ and $(s,h_2,0) \models A'$, so we have $(s,h,n') \models A * A'$.

– $A -\!\!* A'$

Suppose there is at least one $n \in \mathbb{N}$ with $(s,h,n) \models A -\!\!* A'$. Then for any h',n' with $(s,h \cdot h',n+n') \in \mathit{States}$ and $(s,h',n') \models A$ we have $(s,h\cdot h',n+n') \models A'$. Now take any other $m \in \mathbb{N}$ and suppose there are h',m' with $(s,h \cdot h',m+m') \in \mathit{States}$ and $(s,h',m') \models A$. Because $(s,h,n) \models A -\!\!* A'$ we know that $(s,h\cdot h',n+m') \models A'$. But then by induction hypothesis it must also be true that $(s,h \cdot h',m+m') \models A'$. Hence, $(s,h,m) \models A -\!\!* A'$ and. since m was arbitrary, $\forall m.\ (s,h,m) \models A -\!\!* A'$.

– A^E

Because by hypothesis every annotation in A is 0, it must be $E = 0$.

Suppose there is at least one $n \in \mathbb{N}$ with $(s,h,n) \models A^0$. Then it must be $n \geq 0$ and $(s,h,n) \models A$. By induction hypothesis we have $\forall n.\ (s,h,n) \models A$ and since for every $n \in \mathbb{N}$ it is $n \geq 0$, we have $\forall n.\ (s,h,n) \models A^0$.

□

And this is the mentioned relation:

Lemma 3.2. *If* `MAX` $= \infty$*, every annotation occurring in* A *is* 0 *and* $(s,h,0) \in States$ *then*

$$(s,h,0) \models A \quad \textit{iff} \quad (s,h) \models \lfloor A \rfloor$$

Proof. We will reason by stuctural induction on A, again omitting the cases $\bot, \rightarrow, \vee, \forall$.

Throughout all the proof, we will use that `MAX` $= \infty$ implies $(s,h,0) \in States$ for any $h \in Heaps$.

- $\top$

  $\lfloor \top \rfloor = \top$.
  We always have $(s,h,0) \models \top$, as well as $(s,h) \models \top$.

- B

  $\lfloor B \rfloor = B$.
  We have $(s,h,0) \models B$ iff $[\![B]\!]s = \mathbf{true}$ iff $(s,h) \models B$.

- $\neg A$

  $\lfloor \neg A \rfloor = \neg \lfloor A \rfloor$.
  $(s,h,0) \models \neg A$ iff $(s,h,0) \not\models A$ iff (by induction hypothesis) $(s,h) \not\models \lfloor A \rfloor$ iff $(s,h) \models \neg \lfloor A \rfloor$.

- $A \wedge A'$

  $\lfloor A \wedge A' \rfloor = \lfloor A \rfloor \wedge \lfloor A' \rfloor$.
  $(s,h,0) \models A \wedge A'$ iff both $(s,h,0) \models A$ and $(s,h,0) \models A'$. By induction hypothesis, this is true if and only if $(s,h) \models \lfloor A \rfloor$ and $(s,h) \models \lfloor A' \rfloor$, which is equivalent to $(s,h) \models \lfloor A \rfloor \wedge \lfloor A' \rfloor$.

- $\exists x.\ A$

  $\lfloor \exists x.\ A \rfloor = \exists x.\ \lfloor A \rfloor$.
  $(s,h,0) \models \exists x.\ A$ iff, for some $v \in Values$, it is $(s[x \mapsto v], h, m) \models A$. By induction hypothesis, this is true iff $(s[x \mapsto v], h) \models \lfloor A \rfloor$ or equivalently $(s,h) \models \exists x.\ \lfloor A \rfloor$.

- **emp**

  $\lfloor \mathbf{emp} \rfloor = \mathbf{emp}$.
  $(s,h,0) \models \mathbf{emp}$ iff $dom(h) = \emptyset$ iff $(s,h) \models \mathbf{emp}$.

- $E \mapsto E'$

  $\lfloor E \mapsto E' \rfloor = E \mapsto E'$.
  $(s,h,0) \models E \mapsto E'$ iff $dom(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![E']\!]s$ iff $(s,h) \models E \mapsto E'$.

- $A * A'$

  $\lfloor A * A' \rfloor = \lfloor A \rfloor * \lfloor A' \rfloor$.
  $(s,h,0) \models A * A'$ iff $(s,h,0) = (s, h_1 \cdot h_2, n_1 + n_2)$ with $(s,h_1,n_1) \models A$ and $(s,h_2,n_2) \models A'$. $n_1, n_2 \in \mathbb{N}$ and $n_1 + n_2 = 0$ implies $n_1 = n_2 = 0$, so we have $(s,h_1,0) \models A$ and

$(s, h_2, 0) \models A'$. By induction hypothesis we then have $(s, h_1) \models \lfloor A \rfloor$ and $(s, h_2) \models \lfloor A' \rfloor$, and so $(s, h) \models \lfloor A \rfloor * \lfloor A' \rfloor$.

Now suppose $(s, h) \models \lfloor A \rfloor * \lfloor A' \rfloor$. Then there are $h_1, h_2 \in \mathit{Heaps}$ with $h = h_1 \cdot h_2$, $(s, h_1) \models \lfloor A \rfloor$ and $(s, h_2) \models \lfloor A' \rfloor$. By induction hypothesis $(s, h_1, 0) \models A$ and $(s, h_2, 0) \models A'$, and also $(s, h_1 \cdot h_2, 0) \models A * A'$.

– $A -\!\!* A'$

$\lfloor A -\!\!* A' \rfloor = \lfloor A \rfloor -\!\!* \lfloor A' \rfloor$.
Taking $(s, h, 0) \in \mathit{States}$ we have $(s, h, 0) \models A -\!\!* A'$ iff whenever $(s, h', n') \models A$ and it is the case that $(s, h \cdot h', n') \in \mathit{States}$, we also have $(s, h \cdot h', n') \models A'$. Suppose $(s, h') \models \lfloor A \rfloor$ and $h \cdot h'$ is defined. By induction hypothesis, we have $(s, h', 0) \models A$. This implies that $(s, h \cdot h', 0) \models A'$, so again by induction hypothesis we have $(s, h \cdot h') \models \lfloor A' \rfloor$ and so $(s, h) \models \lfloor A \rfloor -\!\!* \lfloor A' \rfloor$.

Now suppose $(s, h) \models \lfloor A \rfloor -\!\!* \lfloor A' \rfloor$. Then whenever $(s, h') \models \lfloor A \rfloor$ and $h \cdot h'$ is defined we have $(s, h \cdot h') \models \lfloor A' \rfloor$. Suppose $(s, h', n') \models A$. Because every annotation in A must be 0, applying Lemma 3.1 we have $(s, h', n'') \models A$ for every $n'' \in \mathbb{N}$, and in particular $(s, h', 0) \models A$. By induction hypothesis, we know then that $(s, h') \models \lfloor A \rfloor$, and so because $(s, h) \models \lfloor A \rfloor -\!\!* \lfloor A' \rfloor$ it is $(s, h \cdot h') \models \lfloor A' \rfloor$. But then, by induction hypothesis we have $(s, h \cdot h', 0) \models A'$, and applying again Lemma 3.1 we have $(s, h \cdot h', n') \models A'$. Hence, $(s, h, 0) \models A -\!\!* A'$.

– A^E

By hypothesis, it must be $E = 0$. $\lfloor A^0 \rfloor = \lfloor A \rfloor$.
$(s, h, 0) \models A^0$ implies, in particular, $(s, h, 0) \models A$, so by induction hypothesis $(s, h) \models \lfloor A \rfloor$.

Now suppose $(s, h) \models \lfloor A \rfloor$. Then by induction hypothesis $(s, h, 0) \models A$, but also $0 \geq 0$, so $(s, h, 0) \models A^0$.

□

Let us explain why we needed to impose the restrictions $\mathtt{MAX} = \infty$ and all annontations in A being 0 to get the previous result.

Suppose we had $\mathtt{MAX} \in \mathbb{N}$ and the assertion $\mathbf{true}^{\mathtt{MAX}+1}$. While every *old* state (s, h) would satisfy $\mathbf{true} = \lfloor \mathbf{true}^{\mathtt{MAX}+1} \rfloor$, no $(s, h, n) \in \mathit{States}$, with our current definition of States, would satisfy $\mathbf{true}^{\mathtt{MAX}+1}$.

Now suppose that, even with $\mathtt{MAX} = \infty$, we had to consider the assertion $\mathbf{true} -\!\!* \mathbf{true}^7$. The state (with the new definition of States) $(s, h, 0)$ does not satisfy $\mathbf{true} -\!\!* \mathbf{true}^7$ because, taking $(s, h', 2) \in \mathit{States}$ with $h \cdot h' \in \mathit{Heaps}$, we have $(s, h', 2) \models \mathbf{true}$ and $(s, h \cdot h', 2) \in \mathit{States}$ but $(s, h \cdot h', 2) \not\models \mathbf{true}^7$ because $2 < 7$. However, $(s, h) \models \lfloor \mathbf{true} -\!\!* \mathbf{true}^7 \rfloor = \mathbf{true} -\!\!* \mathbf{true}$ because in the old model $\mathbf{true} -\!\!* \mathbf{true} \equiv \mathbf{true}$.

3.1.2 Tight Specifications

Most of the command specific rules used in the original memory model remain the same in this one. Only the axioms for allocation and deallocation of memory have been changed. The modified axioms are the following

$$\frac{}{\{x = y \wedge \mathbf{emp}^1\}\ x := \mathit{new}(E)\ \{x \mapsto E[y/x]\}}$$

$$\overline{\{\exists x.\ (E \mapsto x)\}\ dispose(E)\ \{\mathbf{emp}^1\}}$$

We are going to prove soundness of these axioms, and of all the unmodified rules in this setting, and we will do it with respect to the same notions of correctness and safety as in the original heap model (we will only look into partial correctness here). Here they are, adapted to our new definition of states:

Definition 3.1. A configuration $\langle C, (s,h)\rangle$ is *safe* iff $\langle C, (s,h)\rangle \not\rightsquigarrow^* \texttt{fault}$.

Definition 3.2 (Partial Correctness)**.** $\{A\}\ C\ \{A'\}$ holds iff

$$\begin{array}{l} \forall (s,h) \in States.(s,h) \models A \text{ implies} \\ \quad \langle C, (s,h)\rangle \text{ is safe and} \\ \quad \forall (s',h') \in States.\langle C, (s,h)\rangle \rightsquigarrow \langle (s',h')\rangle \text{ implies } (s',h') \models A' \end{array}$$

All technical results about this model are stated and proved in subsection 3.1.4. These results include Safety Monotonicity, Frame Property and Soundness of the Frame Rule, as well as soundness of all other rules and axioms.

First, however, let us see an example of a proof of a program in this setting.

3.1.3 Example: Proof of a program

One of the main data structures used in programming is lists. Since we cannot allocate contiguous cells in this model, we will work, for the time being, with linked lists without contents. We will use the following abbreviations to talk about list segments.

$$(s,h,n) \models ls(x,y,m) \Leftrightarrow (s,h,n) \models (m = 0 \wedge x = y \wedge \mathbf{emp}) \vee (m > 0 \wedge \exists z.\ [x \mapsto z * ls(z,y,m-1)])$$

That is, $ls(x,y,m)$ describes those heaps that form a singly linked list segment of length m from x to y.

Observations:

- $(s,h,n) \models ls(x,y,m)$ and $(s,h,n) \models m > 0$ implies $(s,h,n) \models \neg x \neq \mathbf{nil}$ (because **nil** is not an address and so it is impossible for us to have $(s,h,n) \models \exists z.\ x \mapsto z$ if $x = \mathbf{nil}$).
- $(s,h,n) \models x = \mathbf{nil}$ implies $(s,h,n) \models m = 0 \wedge x = y \wedge \mathbf{emp}$ (same reason as above).
- $(s,h,n) \models x \mapsto y$ iff $(s,h,n) \models ls(x,y,1)$.

We will use these in our example, as well as the following result:

Lemma 3.3 (Transitivity of lists)**.**

$$(s,h,n) \models ls(x,y,m) * ls(y,z,m') \textit{ implies } (s,h,n) \models ls(x,z,m+m')$$

Proof. Induction on m:

- $m = 0$

  Let $(s,h,n) \models ls(x,y,0) * ls(y,z,m')$, then $(s,h,n) \models x = y \wedge \wedge \mathbf{emp} * ls(y,z,m')$, and that implies $(s,h,n) \models ls(x,z,m')$.

- $m \geq 1$

  Let $(s,h,n) \models ls(x,y,m) * ls(y,z,m')$, then it must also be the case that $(s,h,n) \models \exists w.\ [x \mapsto w * ls(w,y,m-1)] * ls(y,z,m')$ and so $(s,h,n) \models \exists w.\ [x \mapsto w * ls(w,y,m-1) * ls(y,z,m')]$ giving, by induction hypothesis, $(s,h,n) \models \exists w.\ [x \mapsto w * ls(w,z,m-1+m')]$, and finally $(s,h,n) \models ls(x,z,m+m')$.

$\square$

We will test our logic with the following example: a program that makes a disjoint copy of a given list. The input to the program is a pointer x to the head of the list.

The program is the following:

$$
\begin{array}{lll}
C & : & y := \mathbf{nil}; \\
 & & i := x; \\
 & & j := \mathbf{nil}; \\
 & & \mathbf{while}\ i \neq \mathbf{nil}\ \mathbf{do} \quad \begin{array}{l} y := \mathbf{new}(j); \\ i := [i]; \\ j := y \end{array}
\end{array}
$$

We want to prove that this program satisfies the following specification:

$$\{ls(x,\mathbf{nil},m)^m\}\ C\ \{ls(x,\mathbf{nil},m) * ls(y,\mathbf{nil},m)\}$$

for every $m \geq 0$.

We make two separate proofs: one for the case $m = 0$ and another for $m > 0$.

- $m = 0$

$$\{ls(x,\mathbf{nil},0)^0\}$$

$$\{ls(x,\mathbf{nil},0) * y \doteq y\}$$

$$y := \mathbf{nil}$$

$$\{ls(x,\mathbf{nil},0) * y \doteq \mathbf{nil}\}$$

$$\{i \doteq i * ls(x,\mathbf{nil},0) * y \doteq \mathbf{nil}\}$$

$$i := x$$

$$\{i \doteq x * ls(x,\mathbf{nil},0) * y \doteq \mathbf{nil}\}$$

$$\{i \doteq x * ls(x,\mathbf{nil},0) * y \doteq \mathbf{nil} * j \doteq j\}$$

$$j := \mathbf{nil}$$

$$\{i \doteq x * ls(x,\mathbf{nil},0) * y \doteq \mathbf{nil}\} * j \doteq \mathbf{nil}\}$$

$$\mathbf{while}\ i \neq \mathbf{nil}\ \mathbf{do}\ \{y := \mathbf{new}(j); i := [i]; j := y\}$$

$$\{(ls(x,\mathbf{nil},0) * ls(y,\mathbf{nil},0)) \wedge i = \mathbf{nil}\}$$

$$\{ls(x,\mathbf{nil},0) * ls(y,\mathbf{nil},0)\}$$

To infer

$$\{i \doteq x * ls(x, \mathbf{nil}, 0) * y \doteq \mathbf{nil}\} * j \doteq \mathbf{nil}\}$$

$$\mathbf{while}\ i \neq \mathbf{nil}\ \mathbf{do}\ \{y := \mathbf{new}(j); i := [i]; j := y\}$$

$$\{(ls(x, \mathbf{nil}, 0) * ls(y, \mathbf{nil}, 0)) \wedge i = \mathbf{nil}\}$$

we use the following:

$$\{(i \doteq x * ls(x, \mathbf{nil}, 0) * y \doteq \mathbf{nil}\} * j \doteq \mathbf{nil}) \wedge i \neq \mathbf{nil}\}$$

$$\{(i \doteq x * x \doteq \mathbf{nil} * y \doteq \mathbf{nil}\} * j \doteq \mathbf{nil}) \wedge i \neq \mathbf{nil}\}$$

$$\{(i \doteq \mathbf{nil} * y \doteq \mathbf{nil}\} * j \doteq \mathbf{nil}) \wedge i \neq \mathbf{nil}\}$$

$$\{\mathbf{false}\}$$

$$y := \mathbf{new}(j); i := [i]; j := y$$

$$\{ls(x, \mathbf{nil}, 0) * ls(y, \mathbf{nil}, 0)\}$$

(we have $\{\mathbf{false}\}\ C\ \{A\}$ for any program C and assertion A.)

So we can say that

$$\{ls(x, \mathbf{nil}, 0)^0\}\ C\ \{ls(x, \mathbf{nil}, 0) * ls(y, \mathbf{nil}, 0)\}$$

– $m > 0$

$$\{ls(x, \mathbf{nil}, m)^m\}$$

$$\{ls(x, \mathbf{nil}, m) * \mathbf{emp}^m\}$$

$$\{ls(x, \mathbf{nil}, m) * y \doteq y * \mathbf{emp}^m\}$$

$$y := \mathbf{nil}$$

$$\{ls(x, \mathbf{nil}, m) * y \doteq \mathbf{nil} * \mathbf{emp}^m\}$$

$$\{i \doteq i * ls(x, \mathbf{nil}, m) * y \doteq \mathbf{nil} * \mathbf{emp}^m\}$$

$$i := x$$

$$\{i \doteq x * ls(i, \mathbf{nil}, m) * y \doteq \mathbf{nil} * \mathbf{emp}^m\}$$

$$\{i \doteq x * ls(i, \mathbf{nil}, m) * y \doteq \mathbf{nil} * j \doteq j * \mathbf{emp}^m\}$$

$$j := \mathbf{nil}$$

$$\{i \doteq x * ls(i, \mathbf{nil}, m) * y \doteq \mathbf{nil} * j \doteq \mathbf{nil} * \mathbf{emp}^m\}$$

$$\{i \doteq x * ls(i, \mathbf{nil}, m) * y \doteq j * j \doteq \mathbf{nil} * \mathbf{emp}^m\}$$

$$\{ls(x, i, 0) * ls(i, \mathbf{nil}, m) * y \doteq j * ls(j, \mathbf{nil}, 0) * \mathbf{emp}^m\}$$

$$\{\exists n.\ [ls(x, i, n) * ls(i, \mathbf{nil}, m - n) * y \doteq j * ls(j, \mathbf{nil}, n) * \mathbf{emp}^{m-n}]\}$$

$$\mathbf{while}\ i \neq \mathbf{nil}\ \mathbf{do}\ \{y := \mathbf{new}(j); i := [i]; j := y\}$$

$$\{\exists n.\ [ls(x, i, n) * ls(i, \mathbf{nil}, m - n) * y \doteq j * ls(j, \mathbf{nil}, n) * \mathbf{emp}^{m-n}] \wedge i = \mathbf{nil}\}$$

$$\{ls(x, i, m) * ls(i, \mathbf{nil}, 0) * ls(y, \mathbf{nil}, m) * \mathbf{emp}^0\}$$

$$\{ls(x, \mathbf{nil}, m) * ls(y, \mathbf{nil}, m)\}$$

To infer

$$\{\exists n.\ [ls(x,i,n) * ls(i,\mathbf{nil},m-n) * y \doteq j * ls(j,\mathbf{nil},n) * \mathbf{emp}^{m-n}]\}$$

$$\mathbf{while}\ i \neq \mathbf{nil}\ \mathbf{do}\ \{y := \mathbf{new}(j); i := [i]; j := y\}$$

$$\{\exists n.\ [ls(x,i,n) * ls(i,\mathbf{nil},m-n) * y \doteq j * ls(j,\mathbf{nil},n) * \mathbf{emp}^{m-n}] \wedge i = \mathbf{nil}\}$$

we use the following:

$$\{\exists n.\ [ls(x,i,n) * ls(i,\mathbf{nil},m-n) * y \doteq j * ls(j,\mathbf{nil},n) * \mathbf{emp}^{m-n}] \wedge i \neq \mathbf{nil}\}$$

$$\{(ls(x,i,k) * ls(i,\mathbf{nil},m-k) * y \doteq j * ls(j,\mathbf{nil},k) * \mathbf{emp}^{m-k}) \wedge i \neq \mathbf{nil}\}$$

$$\{(ls(x,i,k) * ls(i,\mathbf{nil},m-k) * y \doteq j * ls(j,\mathbf{nil},k) * \mathbf{emp}^{m-(k+1)} * \mathbf{emp}^{1}) \wedge i \neq \mathbf{nil}\}$$

$$\{(ls(x,i,k) * ls(i,\mathbf{nil},m-k) * \mathbf{emp}^{1} \wedge y = j * ls(j,\mathbf{nil},k) * \mathbf{emp}^{m-(k+1)}) \wedge i \neq \mathbf{nil}\}$$

$$y := \mathbf{new}(j)$$

$$\{(ls(x,i,k) * ls(i,\mathbf{nil},m-k) * y \mapsto j * ls(j,\mathbf{nil},k) * \mathbf{emp}^{m-(k+1)}) \wedge i \neq \mathbf{nil}\}$$

$$\{i = u \wedge (ls(x,u,k) * i \mapsto v * ls(v,\mathbf{nil},m-(k+1))) * y \mapsto j * ls(j,\mathbf{nil},k) * \mathbf{emp}^{m-(k+1)}\}$$

$$i := [i]$$

$$\{i = v \wedge (ls(x,u,k) * u \mapsto v * ls(v,\mathbf{nil},m-(k+1))) * y \mapsto j * ls(j,\mathbf{nil},k) * \mathbf{emp}^{m-(k+1)}\}$$

$$\{i = v \wedge (ls(x,v,k+1) * ls(v,\mathbf{nil},m-(k+1))) * y \mapsto j * ls(j,\mathbf{nil},k) * \mathbf{emp}^{m-(k+1)}\}$$

$$\{ls(x,i,k+1) * ls(i,\mathbf{nil},m-(k+1)) * ls(y,\mathbf{nil},k+1) * \mathbf{emp}^{m-(k+1)}\}$$

$$\{ls(x,i,k+1) * ls(i,\mathbf{nil},m-(k+1)) * j \doteq j * ls(y,\mathbf{nil},k+1) * \mathbf{emp}^{m-(k+1)}\}$$

$$j := y$$

$$\{ls(x,i,k+1) * ls(i,\mathbf{nil},m-(k+1)) * j \doteq y * ls(y,\mathbf{nil},k+1) * \mathbf{emp}^{m-(k+1)}\}$$

$$\{ls(x,i,k+1) * ls(i,\mathbf{nil},m-(k+1)) * y \doteq j * ls(y,\mathbf{nil},k+1) * \mathbf{emp}^{m-(k+1)}\}$$

$$\{\exists n.\ [ls(x,i,n) * ls(i,\mathbf{nil},m-n) * y \doteq j * ls(j,\mathbf{nil},n) * \mathbf{emp}^{m-n}]\}$$

And so, we have

$$\{ls(x,\mathbf{nil},m)^m\}\ C\ \{ls(x,\mathbf{nil},m) * ls(y,\mathbf{nil},m)\}$$

3.1.4 Results

First of all, we will prove Safety Monotonicity and the Frame Property so we can use them to prove soundness of the Frame Rule immediately after.

Lemma 3.4 (Frame Property)**.** *If $\langle C,(s,h_2,n_2)\rangle$ is safe and $\langle C,(s,h_2 \cdot h_1,n_2+n_1)\rangle \rightsquigarrow^* \langle(s',h',n')\rangle$ then there exist $h_2' \in Heaps$ and $n_2' \in \mathbb{N}$ such that $\langle C,(s,h_2,n_2)\rangle \rightsquigarrow^* \langle(s',h_2',n_2')\rangle$, $h' = h_2' \cdot h_1$ and $n' = n_2' + n_1$.*

Proof. We will reason by structural induction on C. The only cases substantially different from the standard heap model are the cases for allocation and deallocation of memory.

We will use the notation $[v \mapsto v']$ to represent singleton heaps.

– $x := E$

$\langle x := E, (s, h_2, n_2)\rangle$ is safe and $\langle x := E, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Then it must be $s' = s[x \mapsto [\![E]\!]s]$, $h' = h_2 \cdot h_1$ and $n' = n_2 + n_1$, and also the operational semantics gives us $\langle x := E, (s, h_2, n_2)\rangle \rightsquigarrow^* (s[x \mapsto [\![E]\!]s], h_2, n_1)$.

– $x := [E]$

$\langle x := [E], (s, h_2, n_2)\rangle$ is safe and $\langle x := [E], (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Then $[\![E]\!]s \in \mathit{dom}(h_2) \subseteq \mathit{dom}(h_2 \cdot h_1)$ and for some v it is $h_2([\![E]\!]s) = h_2 \cdot h_1([\![E]\!]s) = v$. Hence, we have $s' = s[x \mapsto v]$, $h' = h_2 \cdot h_1$ and $n' = n_2 + n_1$, and also the operational semantics gives us $\langle x := [E], (s, h_2, n_2)\rangle \rightsquigarrow^* (s[x \mapsto v], h_2, n_1)$.

– $x := \mathbf{new}(E)$

$\langle x := \mathbf{new}(E), (s, h_2, n_2)\rangle$ is safe and $\langle x := \mathbf{new}(E), (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Then $n_2 + n_1 \geq n_2 > 0$ and $n' = n_2 + n_1 - 1 = (n_2 - 1) + n_1$, and there must be some $m \in \mathit{Addresses} \setminus \mathit{dom}(h_2 \cdot h_1)$ with $s' = s[x \mapsto m]$, $h' = h_2 \cdot h_1 \cdot [m \mapsto [\![E]\!]s] = h_2 \cdot [m \mapsto [\![E]\!]s] \cdot h_1$. Also, the operational semantics gives us $\langle x := \mathbf{new}(E), (s, h_2, n_2)\rangle \rightsquigarrow^* (s[x \mapsto m], h_2[m \mapsto [\![E]\!]s], n_2 - 1)$.

– $[E] := E'$

$\langle [E] := E', (s, h_2, n_2)\rangle$ is safe and $\langle [E] := E', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Then $[\![E]\!]s \in \mathit{dom}(h_2) \subseteq \mathit{dom}(h_2 \cdot h_1)$, $n' = n_2 + n_1$ and $h' = h_2 \cdot h_1[[\![E]\!]s \mapsto [\![E']\!]s] = h_2[[\![E]\!]s \mapsto [\![E']\!]s] \cdot h_1$, $s' = s$, and the operational semantics gives us $\langle [E] := E', (s, h_2, n_2)\rangle \rightsquigarrow^* (s, h_2[[\![E]\!]s \mapsto [\![E']\!]s], n_2)$.

– $\mathbf{dispose}(E)$

$\langle \mathbf{dispose}(E'), (s, h_2, n_2)\rangle$ is safe and $\langle \mathbf{dispose}(E'), (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Then $[\![E]\!]s \in \mathit{dom}(h_2) \subseteq \mathit{dom}(h_2 \cdot h_1)$, $n' = n_2 + n_1 + 1 = (n_2 + 1) + n_1$, $h' = (h_2 \cdot h_1)\rceil_{\mathit{dom}(h_2 \cdot h_1) \setminus [\![E]\!]s} = h_2\rceil_{\mathit{dom}(h_2) \setminus [\![E]\!]s} \cdot h_1$ and $s' = s$, and the operational semantics gives $\langle \mathbf{dispose}(E), (s, h_2, n_2)\rangle \rightsquigarrow^* (s, h_2\rceil_{\mathit{dom}(h_2) \setminus [\![E]\!]s}, n_2 + 1)$.

– $C; C'$

$\langle C; C', (s, h_2, n_2)\rangle$ is safe and $\langle C; C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Then there is some (s'', h'', n'') such that $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s'', h'', n'')\rangle$ and $\langle C', (s'', h'', n'')\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$.

By induction hypothesis there are h_2'', n_2'' such that $\langle C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s'', h_2'', n_2'')\rangle$ with $h'' = h_2'' \cdot h_1$ and $n'' = n_2'' + n_1$. Because $\langle C; C', (s, h_2, n_2)\rangle$ is safe, $\langle C', (s'', h_2', n_2')\rangle$ must be safe too, and since $\langle C', (s'', h'', n'')\rangle = \langle C', (s'', h_2'' \cdot h_1, n_2'' + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$, again by induction hypothesis there must be some h_2', n_2' with $h' = h_2' \cdot h_1$ and $n' = n_2' + n_1$ such that $\langle C', (s'', h_2'', n_2'')\rangle \rightsquigarrow^* \langle (s', h_2', n_2')\rangle$ and so $\langle C; C', (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s', h_2', n_2')\rangle$.

– **while** B **do** C

$\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ is safe and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$.

We will prove, in this case, the following result:

Given $m \in \mathbb{N}$, if $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ is safe and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$ in m steps, then there exist $h_2' \in \mathit{Heaps}$ and $n_2' \in \mathbb{N}$ such that $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s', h_2', n_2')\rangle$, $h' = h_2' \cdot h_1$ and $n' = n_2' + n_1$..

Let us apply induction on m:

- If $m = 1$ then it must be $[\![B]\!]s = \mathbf{false}$ and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle (s, h_2 \cdot h_1, n_2 + n_1)\rangle$. But then also $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow \langle (s, h_2, n_2)\rangle$.
- If $m > 1$ then it must be $[\![B]\!]s = \mathbf{true}$ and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is the only possible first step in a computation from $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$.

  There must be some (s'', h'', n'') such that $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s'', h'', n'')\rangle$ and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h'', n'')\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$.

  By induction hypothesis there are h''_2, n''_2 with $\langle C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s'', h''_2, n''_2)\rangle$ and $h'' = h''_2 \cdot h_1, n'' = n''_2 + n_1$. It follows from this that $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h''_2, n''_2)\rangle$, so we have that $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h''_2, n''_2)\rangle$ must be safe because $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ is safe.

  On the other hand, similarly, we have $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h'', n'')\rangle = \langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h''_2 \cdot h_1, n''_2 + n_1)\rangle$ and also $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h''_2 \cdot h_1, n''_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$ in less than m steps. Then, by induction (on m) hypothesis, because $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h''_2, n''_2)\rangle$ is safe, we have that there are h'_2, n'_2 with $h' = h'_2 \cdot h_1, n' = n'_2 + n_1$ and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s'', h''_2, n''_2)\rangle \rightsquigarrow^* \langle (s', h'_2, n'_2)\rangle$, and consequently $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s', h'_2, n'_2)\rangle$.

- **if** B **then** C **else** C'

  $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2, n_2)\rangle$ is safe and $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Either $[\![B]\!]s = \mathbf{true}$ or $[\![B]\!]s = \mathbf{false}$.

  If $[\![B]\!]s = \mathbf{true}$ then necessarily we have $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ and $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2, n_2)\rangle \rightsquigarrow \langle C, (s, h_2, n_2)\rangle$. Since we had $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$, we must also have $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$, and so by induction hypothesis there are h'_2, n'_2 with $h' = h'_2 \cdot h_1, n' = n'_2 + n_1$ and $\langle C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s', h'_2, n'_2)\rangle$, and consequently with $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s', h'_2, n'_2)\rangle$.

  Similarly, if $[\![B]\!]s = \mathbf{true}$ then we have $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ and $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2, n_2)\rangle \rightsquigarrow \langle C', (s, h_2, n_2)\rangle$. Since we had $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$, we must also have $\langle C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$, and so by induction hypothesis there are h'_2, n'_2 with $h' = h'_2 \cdot h_1, n' = n'_2 + n_1$ and $\langle C', (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s', h'_2, n'_2)\rangle$, and consequently with $\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (s', h'_2, n'_2)\rangle$.

□

Lemma 3.5 (Safety Monotonicity). *If $\langle C, (s, h_2, n_2)\rangle$ is safe then for any $h_1 \in Heaps$, any $n_1 \in \mathbb{N}$, $(s, h_2 \cdot h_1, n_2 + n_1) \in States$ implies $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe too.*

Proof. We have $\langle C, (s, h_2, n_2)\rangle$ safe. Suppose $(s, h_2 \cdot h_1, n_2 + n_1) \in States$. We need to see that $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe too. Let us proceed by induction on the structure of C:

- $x := E$

  $\langle x := E, (s, h_2, n_2)\rangle$ is safe and $\langle x := E, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe too because, according to our operational semantics, for any state the command $x := E$ never leads to `fault`.

- $x := [E]$

  $\langle x := [E], (s, h_2, n_2)\rangle$ is safe so it must be $[\![E]\!]s \in dom(h_2) \subseteq dom(h_2 \cdot h_1)$ and also $\langle x := [E], (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe.

- $x := \mathbf{new}(E)$

  $\langle x := \mathbf{new}(E), (s, h_2, n_2)\rangle$ is safe so it must be $n_2 > 0$. But then $n_2 + n_1 \geq n_2 > 0$, so $\langle x := \mathbf{new}(E), (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe too.

- $[E] := E'$

  $\langle [E] := E, (s, h_2, n_2)\rangle$ is safe so necessarily $[\![E]\!]s \in dom(h_2) \subseteq dom(h_2 \cdot h_1)$ and therefore $\langle [E] := E, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe as well.

- $\mathbf{dispose}(E)$

  $\langle \mathbf{dispose}(E), (s, h_2, n_2)\rangle$ is safe so it must be $[\![E]\!]s \in dom(h_2) \subseteq dom(h_2 \cdot h_1)$ and $\langle \mathbf{dispose}(E), (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe too.

- $C; C'$

  $\langle C; C', (s, h_2, n_2)\rangle$ is safe so it must be the case that $\langle C, (s, h_2, n_2)\rangle$ is safe and, for any $(\hat{s}, \hat{h_2}, \hat{n_2})$ with $\langle C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (\hat{s}, \hat{h_2}, \hat{n_2})\rangle$, $\langle C', (\hat{s}, \hat{h_2}, \hat{n_2})\rangle$ is safe too.

  By structural induction, since $\langle C, (s, h_2, n_2)\rangle$ is safe $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ must be safe too. Now suppose $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle (\hat{s}, \hat{h}, \hat{n})\rangle$. By the Frame Property it must be $(\hat{s}, \hat{h}, \hat{n}) = (\hat{s}, \hat{h_2} \cdot h_1, \hat{n_2} + n_1)$ for some $\hat{h_2}, \hat{n_2}$ with $\langle C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (\hat{s}, \hat{h_2}, \hat{n_2})\rangle$. Since $\langle C', (\hat{s}, \hat{h_2}, \hat{n_2})\rangle$ must be safe, as we just saw, $\langle C', (\hat{s}, \hat{h}, \hat{n})\rangle = \langle C', (\hat{s}, \hat{h_2} \cdot h_1, \hat{n_2} + n_1)\rangle$ must be safe too, again by structural induction. Hence, $\langle C; C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe.

- **while** B **do** C

  $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ is safe so it must be one of the following two cases:

  $[\![B]\!]s = \mathbf{false}$, which implies $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow \langle (s, h_2, n_2)\rangle$ and then we have $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle (s, h_2 \cdot h_1, n_2 + n_1)\rangle$, and therefore $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe.

  $[\![B]\!]s = \mathbf{true}$, and so $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$. Let us see that $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot, n_2 + n_1)\rangle$ does not lead to `fault`.

  Suppose $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \texttt{fault}$. That would imply that $\langle C; \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \texttt{fault}$. Then we would either have $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \texttt{fault}$, or $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (\tilde{s}, \tilde{h}, \tilde{n})\rangle$ together with $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (\tilde{s}, \tilde{h}, \tilde{n})\rangle \rightsquigarrow^* \texttt{fault}$ for some $(\tilde{s}, \tilde{h}, \tilde{n})$.

  If $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \texttt{fault}$ then there would be a contradiction with the fact that $[\![B]\!]s = \mathbf{true}$ and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ is safe because we have $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle \rightsquigarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ so necessarily $\langle C; \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ must be safe, and this implies that $\langle C, (s, h_2, n_2)\rangle$ is safe too, and so by induction hypothesis $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ must be safe as well.

  If, on the other hand, it is the case that $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \langle (\tilde{s}, \tilde{h}, \tilde{n})\rangle$ and $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (\tilde{s}, \tilde{h}, \tilde{n})\rangle \rightsquigarrow^* \texttt{fault}$, then by the Frame Property there must be some h_2' and n_2' such that $\tilde{h} = h_2' \cdot h_1$, $\tilde{n} = n_2' + n_1$ and $\langle C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (\tilde{s}, h_2', n_2')\rangle$. Since we know that $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2, n_2)\rangle$ is safe and $\langle C, (s, h_2, n_2)\rangle \rightsquigarrow^* \langle (\tilde{s}, h_2', n_2')\rangle$, $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (\tilde{s}, h_2', n_2')\rangle$ must be safe too, and by structural induction we know that $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (\tilde{s}, \tilde{h}, \tilde{n})\rangle = \langle \mathbf{while}\ B\ \mathbf{do}\ C, (\tilde{s}, h_2' \cdot h_1, n_2' + n_1)\rangle$ must be safe as well. We have reached a contradiction again, so it cannot possibly be the case that $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow^* \texttt{fault}$. $\langle \mathbf{while}\ B\ \mathbf{do}\ C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ must therefore be safe.

- **if** B **then** C **else** C'

  We know that $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2, n_2)\rangle$ is safe.

  If $[\![B]\!]s = \textbf{true}$ then $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is the only possible first step from $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$, and so is $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2, n_2)\rangle \rightsquigarrow \langle C, (s, h_2, n_2)\rangle$ from $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2, n_2)\rangle$. But then, since $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2, n_2)\rangle$ is safe $\langle C, (s, h_2, n_2)\rangle$ must be safe, and by induction hypothesis so must be $\langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$. Since the only possible first step from $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ was $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle C, (s, h_2 \cdot h_1, n_2 + n_1)\rangle$, this implies that $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe as well.

  Simililarly, if $[\![B]\!]s = \textbf{false}$ then necessarily $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ and $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2, n_2)\rangle \rightsquigarrow \langle C', (s, h_2, n_2)\rangle$. And so $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2, n_2)\rangle$ safe implies $\langle C', (s, h_2, n_2)\rangle$ must be safe, and by induction hypothesis so must be $\langle C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$. Since the only possible first step from $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ was $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle \rightsquigarrow \langle C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$, this implies that $\langle \textbf{if } B \textbf{ then } C \textbf{ else } C', (s, h_2 \cdot h_1, n_2 + n_1)\rangle$ is safe as well.

□

Proposition 3.1 (Soundness of the Frame Rule)**.** *If* $\{A\}\ C\ \{A'\}$ *holds and* $Modifies(C)\#D$*, then also* $\{A * D\}\ C\ \{A' * D\}$ *holds.*

Proof. Suppose that $\{A\}\ C\ \{A'\}$ holds. Then for any state (s, h, n) with $(s, h, n) \models A$ the configuration $\langle C, (s, h, n)\rangle$ is safe and whenever $\langle C, (s, h, n)\rangle \rightsquigarrow \langle (s', h', n')\rangle$ we have $(s', h', n') \models A'$.

Now suppose $(s, h, n) \models A * D$. Then $(s, h, n) = (s, h_1 \cdot h_2, n_1 + n_2)$ for some h_1, h_2, n_1, n_2 with $(s, h_1, n_1) \models A$ and $(s, h_2, n_2) \models D$. Since $\{A\}\ C\ \{A'\}$ holds, $\langle C, (s, h_1, n_1)\rangle$ is safe, and by Safety Monotonicity so is $\langle C, (s, h_1 \cdot h_2, n_1 + n_2)\rangle$. Let $\langle C, (s, h_1 \cdot h_2, n_1 + n_2)\rangle \rightsquigarrow \langle (s', h', n')\rangle$, by the Frame Property there must be some h'_1, n'_1 with $\langle C, (s, h_1, n_1)\rangle \rightsquigarrow \langle (s', h'_1, n'_1)\rangle$ — and so with $(s', h'_1, n'_1) \models A'$ — and $h' = h' \cdot h_2, n' = n'_1 + n_2$.

Now let us remember the definition of the relation $\#$: $X\#A$ iff for any $s, s' \in Stacks$ with $s\rceil_{Variables \setminus X} = s'\rceil_{Variables \setminus X}$ we have $(s, h) \models A \Rightarrow (s', h) \models A$.

Since our s' is obtained from s by application of C, by definition of $Modifies(C)$ as the set of variables affected by C, it must be the case that $s\rceil_{Variables \setminus Modifies(C)} = s'\rceil_{Variables \setminus Modifies(C)}$. Now since we have $Modifies(C)\#D$ by hypothesis, this means that because $(s, h_2, n_2) \models D$ then also $(s', h_2, n_2) \models D$. Finally, because $(s', h'_1, n'_1) \models A'$, we have $(s', h'_1 \cdot h_2, n'_1 + n_2) \models A' * D$ and consequently $\{A * D\}\ C\ \{A' * D\}$. □

To prove soundness of all the other rules in the logic we will need some auxiliary results:

Lemma 3.6. $[\![F]\!]s[x \mapsto [\![E]\!]s] = [\![F[E/x]]\!]s$.

Proof. Structural induction on F.

- x

  $[\![x]\!]s[x \mapsto [\![E]\!]s] = [\![E]\!]s = [\![x[E/x]]\!]s$.

- $y \neq x$

  $[\![y]\!]s[x \mapsto [\![E]\!]s] = [\![y]\!]s = [\![y[E/x]]\!]s$ (since $y \neq x$, it is $y = y[E/x]$).

– 0 or 1

$[\![0]\!]s[x \mapsto [\![E]\!]s] = [\![0]\!]s = [\![0[E/x]]\!]s$ and $[\![1]\!]s[x \mapsto [\![E]\!]s] = [\![1]\!]s = [\![1[E/x]]\!]s$.

– $F\ op_a\ F'$

$[\![F\ op_a\ F']\!]s[x \mapsto [\![E]\!]s] = [\![F]\!]s[x \mapsto [\![E]\!]s]\ \overline{op_a}\ [\![F']\!]s[x \mapsto [\![E]\!]s]$. By induction hypothesis, this is equal to $[\![F[E/x]]\!]s\ \overline{op_a}\ [\![F'[E/x]]\!]s = [\![F[E/x]\ op_a\ F'[E/x]]\!]s$.

□

Lemma 3.7. *If $[\![x]\!]s = [\![y]\!]s$ then for any arithmetic expression E, it is $[\![E]\!]s = [\![E[y/x]]\!]s$.*

Proof. Structural induction on E.

– x

$[\![x]\!]s = [\![y]\!]s = [\![x[y/x]]\!]s$ $(x[y/x] = y)$.

– $z \neq x$

$z[y/x] = z$ and so $[\![z]\!]s = [\![z[y/x]]\!]s$.

– 0 or 1

$0[y/x] = 0$ and $1[y/x] = 1$, so $[\![0]\!]s = [\![0[y/x]]\!]s$ and $[\![1]\!]s = [\![1[y/x]]\!]s$.

– $F\ op_a\ F'$

$[\![F\ op_a\ F']\!]s = [\![F]\!]s\ \overline{op_a}\ [\![F']\!]s$. By induction hypothesis

$[\![F]\!]s\ \overline{op_a}\ [\![F']\!]s = [\![F[y/x]]\!]s\ \overline{op_a}\ [\![F'[y/x]]\!]s = [\![F[y/x]\ op_a\ F'[y/x]]\!]s = [\![(F\ op_a\ F')[y/x]]\!]s$.

□

Lemma 3.8. *If $dom(s) = dom(s') = D$ and $s\rceil_{D\setminus\{x\}} = s'\rceil_{D\setminus\{x\}}$, then for any arithmetic expression E that contains no occurrences of x it is $[\![E]\!]s = [\![E]\!]s'$.*

Proof. Structural induction on E.

– $z \neq x$

$z \in D \setminus \{x\}$ so $s(z) = s'(z)$, and so $[\![z]\!]s = s(z) = s'(z) = [\![z]\!]s'$.

– 0 or 1

$[\![0]\!]s = 0 = [\![0]\!]s'$ and $[\![1]\!]s = 1 = [\![1]\!]s'$.

– $F\ op_a\ F'$

$[\![F\ op_a\ F']\!]s = [\![F]\!]s\ \overline{op_a}\ [\![F']\!]s$. Since $F\ op_a\ F'$ contains no x, none of F, F' contain any x either. Hence, by induction hypothesis, $[\![F]\!]s\ \overline{op_a}\ [\![F']\!]s = [\![F]\!]'s\ \overline{op_a}\ [\![F']\!]s' = [\![F\ op_a\ F']\!]s'$.

□

And, finally, let us prove soundness of the rules. We give them all once again, together, in Figure 7.

Proposition 3.2 (Soundness)**.** *If $\{A\}\ C\ \{A'\}$ is derivable using the proof rules in Figure 7, then $\{A\}\ C\ \{A'\}$ indeed holds in the sense of Definition 3.2.*

Proof.
We will just show that, for each proof, if its premises hold then so does its conclusion.

$$\frac{}{\{x \doteq y\}\ x := E\ \{x \doteq E[y/x]\}}$$

$$\frac{}{\{x = y \wedge (E \mapsto z)\}\ x := [E]\ \{x = z \wedge (E[y/x] \mapsto z)\}}\ x, y, z \text{ distinct}$$

$$\frac{}{\{\exists x.\ (E \mapsto x)\}\ [E] := F\ \{E \mapsto F\}}$$

$$\frac{}{\{x = y \wedge \mathbf{emp}^1\}\ x := \mathit{new}(E)\ \{x \mapsto E[y/x]\}}$$

$$\frac{}{\{\exists x.\ (E \mapsto x)\}\ \mathit{dispose}(E)\ \{\mathbf{emp}^1\}}$$

$$\frac{\{A\}\ C_1\ \{A''\} \quad \{A''\}\ C_2\ \{A'\}}{\{A\}\ C_1; C_2\ \{A'\}}$$

$$\frac{[\![A_1]\!] \subseteq [\![A_1']\!] \quad \{A_1'\}\ C\ \{A_2'\} \quad [\![A_2']\!] \subseteq [\![A_2]\!]}{\{A_1\}\ C\ \{A_2\}}$$

$$\frac{\{A \wedge B\}\ C_1\ \{A'\} \quad \{A \wedge \neg B\}\ C_2\ \{A'\}}{\{A\}\ \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \{A'\}}$$

$$\frac{\{A \wedge B\}\ C\ \{A\}}{\{A\}\ \mathbf{while}\ B\ \mathbf{do}\ C\ \{A \wedge \neg B\}}$$

Figure 7: Rules for the BoundedeHeap Model

– $\dfrac{}{\{x \doteq y\}\ x := E\ \{x \doteq E[y/x]\}}$

Let $(s, h, n) \models x \doteq y$. This means that $dom(h) = \emptyset$ and $[\![x]\!]s = [\![y]\!]s$. We can see in the operational semantics that $\langle x := E, (s, h, n)\rangle \not\rightsquigarrow^* \texttt{fault}$. Indeed, the only possible computation is $\langle x := E, (s, h, n)\rangle \rightsquigarrow \langle (s[x \mapsto [\![E]\!]s], h, n)\rangle$. To see that $(s[x \mapsto [\![E]\!]s], h, n) \models x \doteq E[y/x]$ we only need to check that $[\![x]\!]s[x \mapsto [\![E]\!]s] = [\![E[y/x]]\!]s[x \mapsto [\![E]\!]s]$.

$[\![x]\!]s[x \mapsto [\![E]\!]s] = [\![E]\!]s$ and by Lemma 3.8 $[\![E[y/x]]\!]s[x \mapsto [\![E]\!]s] = [\![E[y/x]]\!]s$ because there are no occurrences of x in $E[y/x]$. But by Lemma 3.7 we have $[\![E[y/x]]\!]s = [\![E]\!]s$, so we have $[\![x]\!]s[x \mapsto [\![E]\!]s] = [\![E]\!]s = [\![E[y/x]]\!]s = [\![E[y/x]]\!]s[x \mapsto [\![E]\!]s]$.

– $\dfrac{}{\{x = y \wedge (E \mapsto z)\}\ x := [E]\ \{x = z \wedge (E[y/x] \mapsto z)\}}\ x, y, z \text{ distinct}$

Let $(s, h, n) \models x = y \wedge (E \mapsto z)$. Then $[\![x]\!]s = [\![y]\!]s$, $[\![E]\!]s \in dom(h)$ and $h([\![E]\!]s) = [\![z]\!]s$.

The operational semantics then gives $\langle x := [E], (s, h, n)\rangle \rightsquigarrow \langle (s[x \mapsto [\![z]\!]s], h, n)\rangle$ as the only possible computation step.

We need to see that $(s[x \mapsto [\![z]\!]s], h, n) \models x = z \wedge (E[y/x] \mapsto z)$, so we need to check that $(s[x \mapsto [\![z]\!]s], h, n) \models x = z$ and $(s[x \mapsto [\![z]\!]s], h, n) \models E[y/x] \mapsto z$.

$(s[x \mapsto [\![z]\!]s], h, n) \models x = z$ since $[\![x]\!]s[x \mapsto [\![z]\!]s] = [\![z]\!]s = [\![z]\!]s[x \mapsto [\![z]\!]s]$ by Lemma 3.8 since z and x are distinct.

Also, $(s[x \mapsto [\![z]\!]s], h, n) \models E[y/x] \mapsto z$ because, applying Lemma 3.7 (because $[\![x]\!]s = [\![y]\!]s$) and Lemma 3.8 (because $E[y/x]$ contains no occurrences of x) we have $dom(h) = \{[\![E]\!]s\} = \{[\![E[y/x]]\!]s\} = \{[\![E[y/x]]\!]s[x \mapsto [\![z]\!]s]\}$ and applying Lemma 3.8 again like before (because z is distinct from x) we have $h([\![E]\!]s) = [\![z]\!]s = [\![z]\!]s[x \mapsto [\![z]\!]s]$.

- $$\frac{}{\{\exists x.\ (E \mapsto x)\}\ [E] := F\ \{E \mapsto F\}}$$

  Let $(s, h, m) \models \exists x.\ (E \mapsto x)$. This implies $dom(h) = \{[\![E]\!]s\}$.

  The operational semantics gives $\langle [E] := F, (s, h, m)\rangle \rightsquigarrow \langle (s, h[[\![E]\!]s \mapsto [\![F]\!]s], m)\rangle$ as the only possible computation step.

  We need to see that $(s, h[[\![E]\!]s \mapsto [\![F]\!]s], m) \models E \mapsto F$. Let $h' = h[[\![E]\!]s \mapsto [\![F]\!]s]$, we have $m = 0$, $dom(h') = dom(h) = \{[\![E]\!]s\}$ and $h'([\![E]\!]s) = [\![F]\!]s$, so $(s, h', m) \models E \mapsto F$, as we needed to show.

- $$\frac{}{\{x = y \wedge \mathbf{emp}^1\}\ x := new(E)\ \{x \mapsto E[y/x]\}}$$

  Let $(s, h, n) \models x = y \wedge \mathbf{emp}^1$. That means $[\![x]\!]s = [\![y]\!]s$, $dom(h) = \emptyset$ and $n \geq 1$.

  Since $n \geq 1 > 0$, the operational semantics gives $\langle x := new(E), (s, h, n)\rangle \not\rightsquigarrow^* \texttt{fault}$ and $\langle x := new(E), (s, h, n)\rangle \rightsquigarrow \langle (s[x \mapsto m], h[m \mapsto [\![E]\!]s], n-1)\rangle$ for any $m \in Addresses$, $m \notin dom(h)$.

  Now we just need to see that for any such m $(s[x \mapsto m], h[x \mapsto [\![E]\!]s], n-1) \models x \mapsto E[y/x]$. Let $s' = s[x \mapsto m]$, $h' = h[m \mapsto [\![E]\!]s] = [m \mapsto [\![E]\!]s]$. We have $dom(h') = \{m\} = \{[\![x]\!]s'\}$ and also $h'(m) = [\![E]\!]s = [\![E[y/x]]\!]s = [\![E[y/x]]\!]s'$ (using Lemmas 3.7 — $[\![x]\!]s = [\![y]\!]s$— and 3.8 —$E[y/x]$ contains no x). Hence, $(s', h', n-1) \models x \mapsto E[y/x]$.

- $$\frac{}{\{\exists x.\ (E \mapsto x)\}\ dispose(E)\ \{\mathbf{emp}^1\}}$$

  Let $(s, h, m) \models \exists x.\ (E \mapsto x)$. Then $dom(h) = \{[\![E]\!]s\}$.

  The operational semantics gives $\langle dispose(E), (s, h, m)\rangle \rightsquigarrow \langle (s, h\rceil_\emptyset, m+1)\rangle$ as the only possible computation step (since $dom(h) \setminus \{[\![E]\!]s\} = \emptyset$).

  Now $(s, h\rceil_\emptyset, m+1) \models \mathbf{emp}^1$ because $m + 1 \geq 1$ and $dom(h\rceil_\emptyset) = \emptyset$.

- $$\frac{\{A\}\ C_1\ \{A''\} \quad \{A''\}\ C_2\ \{A'\}}{\{A\}\ C_1; C_2\ \{A'\}}$$

  Let $(s, h, n) \models A$. Suppose $\langle C_1; C_2, (s, h, n)\rangle \rightsquigarrow^* \texttt{fault}$, then either $\langle C_1, (s, h, n)\rangle \rightsquigarrow^* \texttt{fault}$ — which is a contradiction because we have $\{A\}\ C_1\ \{A''\}$ — or there is some (s'', h'', n'') such that $\langle C_1, (s, h, n)\rangle \rightsquigarrow^* \langle (s'', h'', n'')\rangle$ and $\langle C_2, (s'', h'', n'')\rangle \rightsquigarrow^* \texttt{fault}$. This leads to a contradiction too, since by $\{A\}\ C_1\ \{A''\}$ any such (s'', h'', n'') verifies $(s'', h'', n'') \models A''$, and by $\{A''\}\ C_2\ \{A'\}$ we have that $\langle C_2, (s'', h'', n'')\rangle$ is safe, so it is impossible that $\langle C_2, (s'', h'', n'')\rangle \rightsquigarrow^* \texttt{fault}$. Hence $\langle C_1; C_2, (s, h, n)\rangle$ is safe.

  Now suppose we have $\langle C_1; C_2, (s'', h'', n'')\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Then there must be some state (s'', h'', n'') such that $\langle C_1, (s, h, n)\rangle \rightsquigarrow^* \langle (s'', h'', n'')\rangle$ — and so $(s'', h'', n'') \models A''$ as we have seen before — and $\langle C_2, (s'', h'', n'')\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$. Because $(s'', h'', n'') \models A''$ and $\{A''\}\ C_2\ \{A'\}$, we have $(s', h', n') \models A'$. Therefore, for any (s', h', n') verifying $\langle C_1; C_2, (s'', h'', n'')\rangle \rightsquigarrow^* \langle (s', h', n')\rangle$ we have $(s', h', n') \models A'$.

- $$\frac{A_1 \models A_1' \quad \{A_1'\}\ C\ \{A_2'\} \quad A_2' \models A_2}{\{A_1\}\ C\ \{A_2\}}$$

Let $(s,h,n) \models A_1$. Since $A_1 \models A_1'$, $(s,h,n) \models A_1'$. Now because $\{A_1'\}\ C\ \{A_2'\}$, $\langle C,(s,h,n)\rangle$ is safe, and whenever $\langle C,(s,h,n)\rangle \leadsto^* \langle(s',h',n')\rangle$ we have $(s',h',n') \models A_2'$, so we have $(s',h',n') \models A_2$ because $A_2' \models A_2$.

- $\dfrac{\{A \wedge B\}\ C_1\ \{A'\} \quad \{A \wedge \neg B\}\ C_2\ \{A'\}}{\{A\}\ \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \{A'\}}$

  Let $(s,h,n) \models A$. Then either $(s,h,n) \models A \wedge B$ or $(s,h,n) \models A \wedge \neg B$.

  - $(s,h,n) \models A \wedge B$

    Then $\langle \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,(s,h,n)\rangle \leadsto^* \texttt{fault}$ implies $\langle C_1,(s,h,n)\rangle \leadsto^* \texttt{fault}$. That is not possible, since we have $\{A \wedge B\}\ C_1\ \{A'\}$ by hypothesis. Therefore, $\langle \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,(s,h,n)\rangle$ must be safe. Also, if the operational semantics gives $\langle \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,(s,h,n)\rangle \leadsto^* \langle(s',h',n')\rangle$ then it must be $\langle C_1,(s,h,n)\rangle \leadsto^* \langle(s',h',n')\rangle$, and so $(s',h',n') \models A'$.

  - $(s,h,n) \models A \wedge \neg B$

    Then $\langle \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,(s,h,n)\rangle \leadsto^* \texttt{fault}$ implies $\langle C_2,(s,h,n)\rangle \leadsto^* \texttt{fault}$. That cannot be so, because by hypothesis we have $\{A \wedge \neg B\}\ C_2\ \{A'\}$. Hence, $\langle \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,(s,h,n)\rangle$ is safe. Also if $\langle \mathbf{if}\ B\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,(s,h,n)\rangle \leadsto^* \langle(s',h',n')\rangle$ then it must be the case that $\langle C_2,(s,h,n)\rangle \leadsto^* \langle(s',h',n')\rangle$, and so $(s',h',n') \models A'$.

- $\dfrac{\{A \wedge B\}\ C\ \{A\}}{\{A\}\ \mathbf{while}\ B\ \mathbf{do}\ C\ \{A \wedge \neg B\}}$

  Suppose $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle \leadsto^* T$ in a finite number m of steps, with T some terminal configuration, either of the form $\langle(s',h',n')\rangle$ or $\texttt{fault}$. We will show that if $(s,h,n) \models A$ then $T = \langle(s',h',n')\rangle$ for some s',h',n' with $(s',h',n') \models A \wedge \neg B$. This will imply, in particular, that $T \neq \texttt{fault}$, and so that $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle$ is safe.

  Let us reason by induction on m.

  - $m = 1$

    In this case we have $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle \leadsto T$. Suppose $(s,h,n) \models B$. Then $[\![B]\!]s = \mathbf{true}$ and $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle \leadsto \langle C; \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle$. But $\langle C; \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle$ is not a terminal configuration, so it must be $(s,h,n) \not\models B$ (and, consequently, $[\![B]\!]s = \mathbf{false}$). This gives only one possible computation: $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle \leadsto \langle(s,h,n)\rangle$. So we have that in this case T must be $\langle(s,h,n)\rangle$. We know by hypothesis that $(s,h,n) \models A$, and also $(s,h,n) \not\models B$, so we have $(s,h,n) \models A \wedge \neg B$.

  - $m > 1$

    As we have seen, $(s,h,n) \not\models B$ implies that $m = 1$, so in this case it must be $(s,h,n) \models B$ (and so $(s,h,n) \models A \wedge B$). This gives $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle \leadsto \langle C; \mathbf{while}\ B\ \mathbf{do}\ C,(s,h,n)\rangle \leadsto^* T$.

    If $T = \texttt{fault}$ then either $\langle C,(s,h,n)\rangle \leadsto^* \texttt{fault}$, or $\langle C,(s,h,n)\rangle \leadsto^* \langle(s'',h'',n'')\rangle$ (for some s'',h'',n'') and $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s'',h'',n'')\rangle \leadsto^* \texttt{fault}$. $\langle C,(s,h,n)\rangle \leadsto^* \texttt{fault}$ leads to contradiction, since we have $(s,h,n) \models A \wedge B$ and $\{A \wedge B\}\ C\ \{A\}$, and this implies that $\langle C,(s,h,n)\rangle$ must be safe.

    On the other hand, if $T = \langle(s',h',n')\rangle$ for some s',h',n' then it must be the case that $\langle C,(s,h,n)\rangle \leadsto^* \langle(s'',h'',n'')\rangle$ (for some s'',h'',n'') and $\langle \mathbf{while}\ B\ \mathbf{do}\ C,(s'',h'',n'')\rangle \leadsto^* \langle(s',h',n')\rangle$.

In both cases we have $\langle C; \textbf{while } B \textbf{ do } C, (s, h, n)\rangle \rightsquigarrow^* \langle \textbf{while } B \textbf{ do } C, (s'', h'', n'')\rangle \rightsquigarrow^* T$. Since $(s, h, n) \models A \wedge B$ and $\{A \wedge B\}\ C\ \{A\}$, it must be $(s'', h'', n'') \models A$. This means that we have $\langle \textbf{while } B \textbf{ do } C, (s'', h'', n'')\rangle \rightsquigarrow^* T$ in less than m steps, and $(s'', h'', n'') \models A$, and so by induction hypothesis $T = \langle (s', h', n')\rangle$ for some (s', h', n') with $(s', h', n') \models A \wedge \neg B$.

□

3.1.5 Alternative Approach

One may wonder why we do not choose to include a properly specified set of addresses instead of just a number of random cells as the new third component in our states, as we have been doing throughout this section so far. Since allocation takes place in a nondeterministic mannner, there is no reason why we should care about the precise addresses of the cells we own. Not only is our first approach simpler, we will see that some problems arise from considering a specific set of addresses. Also, it reflects better the notion of arity of a state: memory allocation commands do not need a certain memory cell available for allocation, they need *some* such cell.

Stacks and *Heaps* are still defined as in section 2, and this is the new definition of states:

$$States \stackrel{def}{=} \{(s, h, r) \in Stacks \times Heaps \times \mathcal{P}(Addresses) \text{ such that } dom(h) \cap r = \emptyset\}$$

The combination operation will now be

$$(s, h, r) \cdot (s, h', r') = \begin{cases} (s, h \cdot h', r \cup r') & \begin{array}{l}\text{if } h \cdot h' \text{ is defined and} \\ r \cap r' = r \cap dom(h') = r' \cap dom(h) = \emptyset\end{array} \\ \text{undefined otherwise} \end{cases}$$

The condition for $(s, h, r) \cdot (s, h', r')$ being defined is equivalent to $(s, h \cdot h', r \cup r') \in States$ and $r \cap r' = \emptyset$.

Had this model worked nicely, the operational semantics would have had to be revised so that memory allocation only took cells that are in the specified set of reserved cells of the state.

An inconvenience of this approach is that the Frame Property does not hold if adapted straightforwardly from its original formulation:

"If $\langle C, (s, h_2, r_2)\rangle \not\rightsquigarrow^* \texttt{fault}$ and $\langle C, (s, h_2 \cdot h_1, r_2 \cup r_1)\rangle \rightsquigarrow^* \langle (s', h', r')\rangle$ then there are h_2' and r_2' such that $\langle C, (s, h_2, r_2)\rangle \rightsquigarrow^* \langle (s', h_2', r_2')\rangle$ with $h' = h_2' \cdot h_1$ and $r' = r_2' \cup r_1$."

Let us understand where the problem lies. Suppose we have a configuration $\langle C, (s, h_2, r_2)\rangle$ such that $\langle C, (s, h_2, r_2)\rangle \not\rightsquigarrow^* \texttt{fault}$, and let $\langle C, (s, h_2 \cdot h_1, r_2 \cup r_1)\rangle \rightsquigarrow^* \langle (s', h', r')\rangle$. Suppose command C allocates some cell of memory which it never deallocates again. The chosen cell to be allocated when executing C on $(s, h_2 \cdot h_1, r_2 \cup r_1)$ might be one in r_1, and then it would be impossible to track the computation on the big state back to the smaller state.

Let us see this better with a specific counterexample: take C as $x := \textbf{new}(\textbf{nil})$ and consider states $(s, h_1, r_1), (s, h_2, r_2)$ with $h_1 = h_2 = e$, $r_1 = \{5\}$ and $r_2 = \{7\}$. Their combination would result in state $(s, h_1 \cdot h_2, r_1 \cup r_2) = (s, e, \{5, 7\})$, and a computation on this big state could be $\langle x := \textbf{new}(\textbf{nil}), (s, e, \{5, 7\})\rangle \rightsquigarrow^* \langle (s[x \mapsto 7], [7 \mapsto \textbf{nil}], \{5\})\rangle$. We would need to find r_2' such that $\langle x := \textbf{new}(\textbf{nil}), (s, e, \{5\})\rangle \rightsquigarrow^* \langle (s[x \mapsto 7], [7 \mapsto \textbf{nil}], r_2')\rangle$ and $\{5\} = r_2' \cup \{7\}$, when this is clearly impossible.

We tried finding a modified version of the Frame Property that would hold in this setting, and that would still guarantee soundness of the Frame Rule. What we did is try to find a reasonable notion of *isomorphy* between heaps, in the form of an equivalence relation between states. We chose the following relation, $\backsim$:

Definition 3.3 (Heap Isomorphism)**.** Given $h, h' \in \mathit{Heaps}$, we shall say that a bijective mapping $f : \mathit{Addresses} \to \mathit{Addresses}$ is a heap isomorphism between h and h' iff for any $a \in \mathit{dom}(h)$ we have

- if $h(a) \in \mathit{Atoms} \cup \{\mathbf{nil}\}$ then $h'(f(a)) = h(a)$.
- if $h(a) \in \mathit{Addresses}$ then $h'(f(a)) = f(h(a))$.

Definition 3.4 (Isomorphic States)**.** Given two states $(s, h, r), (s', h', r') \in \mathit{States}$ we shall say that they are isomorphic, and will denote it $(s, h, r) \backsim (s', h', r')$, iff

- $|r'| = |r|$
- there is a heap isomorphism f between h and h' such that for any arithmetic expresssion E we have
  - if $[\![E]\!]s \in \mathit{Atoms} \cup \{\mathbf{nil}\}$ then $[\![E]\!]s' = [\![E]\!]s$
  - if $[\![E]\!]s \in \mathit{Addresses}$ then $[\![E]\!]s' = f([\![E]\!]s)$

At first sight, it seems a reasonable equivalence relation and, ideally, it would have made the following modified Frame Property hold:

If $\langle C, (s, h_2, r_2)\rangle$ *is safe and* $\langle C, (s, h_2 \cdot h_1, r_2 \cup r_1)\rangle \rightsquigarrow^* \langle (s', h', r')\rangle$ *then there are* $\overline{s'}, h'_2$ *and* r'_2 *such that* $\langle C, (s, h_2, r_2)\rangle \rightsquigarrow^* \langle (\overline{s'}, h'_2, r'_2)\rangle$ *and* $(s', h', r') \backsim (\overline{s'}, h'_2 \cdot h_1, r'_2 \cup h_1)$.

However, the possibility of performing address arithmetic ruins everything. Not even the following basic property, clearly necessary for the given modified Frame Property to guarantee soundness of the Frame Rule, holds:

If $(s, h, r) \backsim (s', h', r')$ *then* $(s, h, r) \models A$ *implies* $(s', h', r') \models A$ *too.*

Consider the assertion $x < y$. There could be two isomorphic states $(s, h, r), (s', h', r')$ with $[\![E]\!]s, [\![E]\!]s' \in \mathit{Addresses}$ but $[\![E]\!]s < [\![E']\!]s$ and $[\![E]\!]s' < [\![E']\!]s'$. We would need to modify the semantics of arithmetic and boolean assertions so as to ensure that addresses cannot be used at all in arithmetic or boolean expressions, which would be a very unnatural thing to do.

3.2 The Bounded Heap Model: allocating contiguous cells

Our first approach to the bounded heap model had a considerable flaw: it did not allow allocation of several contiguous cells. Address arithmetic, however, is quite useful to represent various data structures [1] such as lists — grouping two contiguous cells together so that one holds the contents and the other points to the next element in the list — or binary trees — similarly, grouping three contiguous cells so that one holds the contents and the other two point to the children.

Definition 3.5. Taking *Stacks*, *Heaps*, *Addresses* and `MAX` defined as in section 3.1, we can take a multiset of natural numbers as the new third component of our states, representing disjoint contiguous gaps in the memory over which we have permission:

$$States \subseteq Stacks \times Heaps \times \mathcal{MP}(\mathbb{N})$$

(where $\mathcal{MP}(\mathbb{N}) = \{A \mid A$ is a multiset and $a \in A$ implies $a \in \mathbb{N}\})$ with $(s, h, g) \in States$ (we use the letter g for *gaps*) iff $g = \{g_1, \ldots, g_n\}$ is finite and $\exists\{m_1, \ldots, m_n\} \in Addresses \setminus dom(h)$ such that, if we define

$$S_i = \{m_i, m_i + 1, \ldots, m_i + g_i - 1\}$$

then $\bigcup_{i=1}^{n} S_i \subseteq Addresses \setminus dom(h)$ and for every $j, i \in \{1, \ldots, n\}$ with $i \neq j$, we have $S_i \cap S_j = \emptyset$.

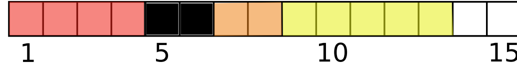
What this definition says is that the gaps described in the third component can be located, disjointly, in our finite memory heap with all cells in $dom(h)$ being already allocated.

The combination operation now will be:

$$(s, h_1, g_1) \cdot (s, h_2, g_2) = (s, h_1 \cdot h_2, g_1 \cup g_2)$$

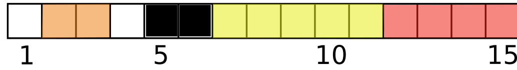
iff $(s, h_1 \cdot h_2, g_1 \cup g_2) \in States$, and undefined otherwise.

The reserved blocks can be anywhere in the memory heap, as long as they take no cells that are allocated according to the heap component of the state. For example, suppose we have `MAX` $= 15$ and a state (s, h, g) where $dom(h) = \{5, 6\}$ and $g = \{2, 4, 5\}$. One possible way of locating the gaps in the memory would be



(Allocated cells in black.)

But the gaps could also be distributed, for example, in the following way:



The commands remain the same as in section 3.1, except for the allocation and deallocation commands — not only can we allocate contiguous cells now, but also deallocate several contiguous cells at once:

$$C ::= \ldots \mid x := \mathbf{new}(E_1, \ldots, E_n) \mid \mathbf{dispose}(E, E')$$

We propose the following operational semantics for allocation/deallocation of memory:

$$\frac{[\![E]\!]s = n \quad [\![E']\!] = m \quad \{m, \ldots, m+n-1\} \subseteq dom(h)}{\langle \mathbf{dispose}(E, E'), (s, h, g)\rangle \rightsquigarrow^* \langle (s, h\rceil_{dom(h)\setminus\{m,\ldots,m+n-1\}}, g \cup \{n\})\rangle}$$

$$\frac{[\![E]\!]s = n \quad [\![E']\!] = m \quad \{m, \ldots, m+n-1\} \not\subseteq dom(h)}{\langle \mathbf{dispose}(E, E'), (s, h, g)\rangle \rightsquigarrow^* \texttt{fault}}$$

$$\frac{\exists g_i \in g \text{ such that } n \le g_i \quad \{a, \ldots, a+n-1\} \subseteq \mathit{Addresses} \setminus \mathit{dom}(h)}{\langle x := \mathbf{new}(E_1, \ldots, E_n), (s,h,g)\rangle \rightsquigarrow^* \langle (s[x \mapsto a], h[a \mapsto [\![E_1]\!]s, \ldots, a+n-1[\![E_n]\!]s], g(g_i, n))\rangle}$$

$$\frac{\nexists g_i \in g \text{ such that } n \le g_i}{\langle x := \mathbf{new}(E_1, \ldots, E_n), (s,h,g)\rangle \rightsquigarrow^* \texttt{fault}}$$

$$\frac{\text{for all } a \in \mathit{Addresses} \;\; \{a, \ldots, a+n-1\} \not\subseteq \mathit{Addresses} \setminus \mathit{dom}(h)}{\langle x := \mathbf{new}(E_1, \ldots, E_n), (s,h,g)\rangle \rightsquigarrow^* \texttt{fault}}$$

with

$$g(g_i, n) = \begin{cases} \emptyset & \text{if } |g| < 3 \text{ and } n = g_i \\ \{\lceil \frac{g_i - n}{2} \rceil\} & \text{if } |g| < 3 \text{ and } n < g_i \\ g \setminus \{g_i, g_{max1}, g_{max2}\} & \text{if } |g| \ge 3, n = g_i \text{ and } g_{max1}, g_{max2} \text{ are the two biggest elements in } g \setminus \{g_i\} \; (g_{max2} \le g_{max1}) \\ (g \setminus \{g_i, g_{max1}, g_{max2}\}) \cup \{\lceil \frac{g_i - n}{2} \rceil\} & \text{if } |g| \ge 3, n < g_i \text{ and } g_{max1}, g_{max2} \text{ are the two biggest elements in } g \setminus \{g_i\} \; (g_{max2} \le g_{max1}) \end{cases}$$

While the operational semantics for memory deallocation is quite straightforward, the operational semantics for the command $x := \mathbf{new}(E_1, \ldots, E_n)$ is not so. We have chosen to make it dependent only on the component describing the gaps — not dependent at all on the allocated memory component — in order to make it concise.

When we deallocate a block of n contiguous cells, we just have to add a new gap of the same size to our reserved gaps. However, when we allocate n contiguous cells we cannot be sure of where they are going to be placed in the memory just by the information in g, and it is not straightforward to calculate the size of the remaining gaps. This results, inevitably, in some reserved cells being lost.

However, the given operational semantics is sound in the following sense:

Proposition 3.3. *If according to the given operational semantics we have*

$$\langle x := \mathbf{new}(E_1, \ldots, E_n), (s,h,g)\rangle \rightsquigarrow^* \langle (s', h', g')\rangle$$

then $(s', h', g') \in \mathit{States}$.

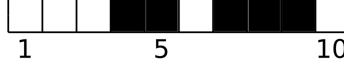
Before giving the proof for this result, let us see what our initial attempts to find a sensible operational semantics for memory allocation in this model were. For each one, we will see an example that shows why they do not work. All the approaches differ in their definition of $g(g_i, n)$.

In our first attempt we defined $g(g_i, n)$ as follows:

$$g(g_i, n) = \begin{cases} g \setminus \{g_i\} & \text{if } n = g_i \\ (g \setminus \{g_i\}) \cup \{\lceil \frac{g_i - n}{2} \rceil\} & \text{if } n < g_i \end{cases}$$

An example of why this is not a sensible operational semantics for $x := \mathbf{new}(E_1, \ldots, E_n)$ would be this: take (s, h, g) with $\mathit{Addresses} = \{1, \ldots, 10\}$, $\mathit{dom}(h) = \{4, 5\}$ and $g = \{3, 5\}$:

If we try to allocate 3 contiguous cells ($x := \mathbf{new}(E_1, E_2, E_3)$), this operational semantics could lead us to terminal configuration $\langle(s[x \mapsto 7], h[7 \mapsto [\![E_1]\!]s, 8 \mapsto [\![E_2]\!]s, 9 \mapsto [\![E_3]\!]s], \{5\})\rangle$ taking $g_i = 3$ and $\{7, 8, 9\} \subseteq \mathit{Addresses} \setminus \mathit{dom}(h)$. However, allocating cells $7, 8, 9$ results in a memory heap with shape



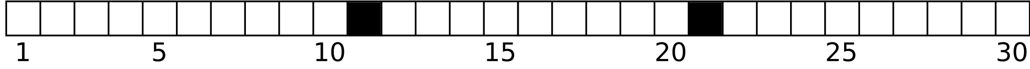
which contains no gap of size 5. The problem is that the gap actually used to allocate the n cells may be different from (and larger than) the one described by g_i. This means that naively removing g_i from the list of gaps is not good enough.

The is how $g(g_i, n)$ looked in our second attempt:

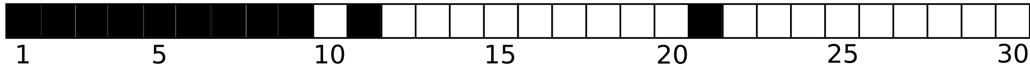
$$g(g_i, n) = \begin{cases} g \setminus \{g_{max}\} & \text{if } n = g_i \\ (g \setminus \{g_{max}\}) \cup \{\lceil \frac{g_i - n}{2} \rceil\} & \text{if } n < g_i \end{cases}$$

with g_{max} being the greatest element in g.

Again, here we have an example of why this is not a good operational semantics for memory allocation in this model: consider (s, h, g) with $\mathit{Addresses} = \{1, \ldots, 30\}$, $\mathit{dom}(h) = \{11, 21\}$ and $g = \{2, 3, 5, 9, 9\}$:



If we try to allocate 9 contiguous cells ($x := \mathbf{new}(E_1, \ldots, E_9)$), this operational semantics could lead to terminal configuration $\langle(s[x \mapsto 1], h[1 \mapsto [\![E_1]\!]s, \ldots, 9 \mapsto [\![E_9]\!]s], \{2, 3, 5, 9\})\rangle$ taking $g_i = 9$ and $\{1, \ldots, 9\} \subseteq \mathit{Addresses} \setminus \mathit{dom}(h)$. However, allocating cells $1, \ldots, 9$ produces a memory heap with shape



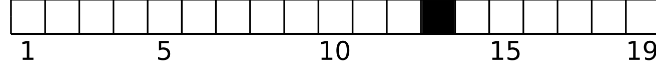
and it is not possible to fit there four disjoint gaps of sizes $2, 3, 5, 9$.

The last failed attempt was to define:

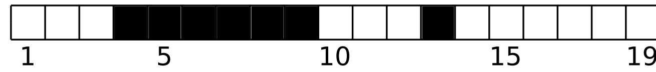
$$g(g_i, n) = \begin{cases} \emptyset & \text{if } |g| < 2 \text{ and } n = g_i \\ \{\lceil \frac{g_i - n}{2} \rceil\} & \text{if } |g| < 2 \text{ and } n < g_i \\ g \setminus \{g_i, g_{max}\} & \text{if } |g| \geq 2, n = g_i \\ (g \setminus \{g_i, g_{max}\}) \cup \{\lceil \frac{g_i - n}{2} \rceil\} & \text{if } |g| \geq 2, n < g_i \end{cases}$$

with g_{max} being the greatest element in $g \setminus \{g_i\}$.

Once more, here we have an example that shows why this is not a sensible operational semantics for $x := \textbf{new}(E_1, \ldots, E_n)$: take (s, h, g) with $Addresses = \{1, \ldots, 19\}$, $dom(h) = \{13\}$ and $g = \{4, 4, 4, 6\}$:



Allocating 6 contiguous cells ($x := \textbf{new}(E_1, \ldots, E_6)$) with this operational semantics can lead to terminal configuration $\langle (s[x \mapsto 4], h[4 \mapsto [\![E_1]\!]s, \ldots, 9 \mapsto [\![E_6]\!]s], \{4, 4\}) \rangle$ if g_i is taken to be 6 ($\{4, \ldots, 9\} \subseteq Addresses \setminus dom(h)$). However, allocating cells $4, \ldots, 9$ results in a memory heap with shape



which obviously cannot hold two gaps of size 4.

Finally, here is the proof to Proposition 3.3.

Proof. Let $(s, h, g) \in States$ and suppose $\langle x := \textbf{new}(E_1, \ldots, E_n), (s, h, g) \rangle \rightsquigarrow^* \langle (s', h', g(g_i, n)) \rangle$. Let us consider some fixed arbitrary positions for the gaps described by the elements in g inside the heap described by h. With the gaps fixed in the memory, we will explore all the possibilities for how the new n cells can be allocated, and in each of them we will find a way to locate the gaps described by $g(g_i, n)$ within $Addresses \setminus dom(h')$. That is, we will find a way to assign each element in $g(g_i, n)$ to describe a gap in $Addresses \setminus dom(h')$ so that the gaps described by any two elements of $g(g_i, n)$ are disjoint. This will prove that $(s', h', g(g_i, n)) \in States$.

Throughout the proof, we will refer to the gaps in the initial fixed (arbitrary) distribution as *the gap initially described by* g_j.

We can consider, without loss of generality, that no cells apart from the ones assigned to our gaps in the chosen arbitrary distribution are used in the allocation. If this was the case, then it would mean that fewer cells from the gaps we are considering have been taken, and so we would have fewer restrictions to locate the gaps described by $g(g_i, n)$.

We study the different cases considered in the definition of $g(g_i, n)$:

– If $|g| < 3$ and $n = g_i$ then $g(g_i, n) = \emptyset$, which is always an acceptable set of reserved gaps.

– If $|g| < 3$ and $n < g_i$ then $g(g_i, n) = \{\lceil \frac{g_i - n}{2} \rceil\}$.

  The n cells will get allocated somewhere inside the at most two gaps initially located in the heap.

  Whether there are two gaps or just the one described by g_i, if they are allocated using only the gap corresponding to g_i, then for sure there will still be a contiguous gap of size at least $\lceil \frac{g_i - n}{2} \rceil$ — after taking n contiguous cells from the gap, there will stil be $g_i - n$ cells, separated in at most two blocks, of which one must surely be bigger than $\lceil \frac{g_i - n}{2} \rceil$.

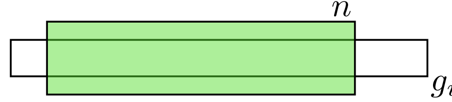
  If $|g| = 2$ and they get allocated in the gap corresponding to the other $g_j \in g \setminus \{g_i\}$, then there will still be a gap of size g_i, and so a gap of size $\lceil \frac{g_i - n}{2} \rceil$ contained in it.

Finally, if the two gaps are placed adjacently and the n cells get allocated using cells from both gaps, then there will still be a contiguous gap of size at least $\lceil \frac{g_i - n}{2} \rceil$ — we can follow the same reasoning as in the case where only the gap g_i was used, only starting with an even bigger gap.

– If $|g| \geq 3$ and $n < g_i$ and g_{max1}, g_{max2} are the two biggest elements in $g \setminus \{g_i\}$ then $g(g_i, n) = (g \setminus \{g_i, g_{max1}, g_{max2}\}) \cup \{\lceil \frac{g_i - n}{2} \rceil\}$.

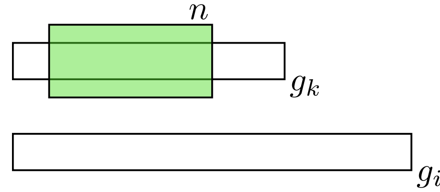
  Let us consider the elements in $g = \{g_1, \ldots, g_m\}$ enumerated in increasing order: $g_1 \leq g_2 \leq \cdots \leq g_i \leq \cdots \leq g_m$).

  – Suppose the n cells are allocated using only the gap initially described by g_i.



    Then obviously all the elements in $g(g_i, n) \setminus \{\lceil \frac{g_i - n}{2} \rceil\}$ can still be used to describe the same gaps as they initially described. Also, as explained in the previous case, after allocating n cells in the gap of size g_i, no matter where in the gap they have been allocated, there will still be a gap of size at least $\lceil \frac{g_i - n}{2} \rceil$, which we can now assign to be described by $\lceil \frac{g_i - n}{2} \rceil \in g(g_i, n)$.

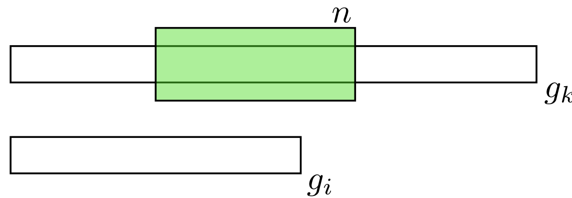
  – Suppose the n cells are allocated using only a gap smaller than g_i (g_k with $k < i$).



    If g_i, g_{max1}, g_{max2} are the three greatest elements in g then it might be the case that $g_k = g_{max1}$ or $g_k = g_{max2}$. If $g_k = g_{max1}$ or $g_k = g_{max2}$ then every element in $g(g_i, n) \setminus \{\lceil \frac{g_i - n}{2} \rceil\}$ can still describe the same gap as before, for they are intact, and $\{\lceil \frac{g_i - n}{2} \rceil\}$ can describe any subset of $\lceil \frac{g_i - n}{2} \rceil < g_i$ contiguous cells contained in the gap initially described by g_i. If $g_k \neq g_{max1}, g_{max2}$ ($g_k < g_i, g_{max1}, g_{max2}$), then every $g_j \in g(g_i, n) \setminus \{g_k, \lceil \frac{g_i - n}{2} \rceil\}$ can still describe the same gap as before, g_k can describe any subset of $g_k < g_i, g_{max1}, g_{max2}$ cells of the gap initially described by $min\{g_i, g_{max1}, g_{max2}\}$ and $\lceil \frac{g_i - n}{2} \rceil < max\{g_i, g_{max1}, g_{max2}\}$ (since $\lceil \frac{g_i - n}{2} \rceil < g_i$) can describe any subset of $\lceil \frac{g_i - n}{2} \rceil$ contiguous cells contained in the gap initially described by $max\{g_i, g_{max1}, g_{max2}\}$.

    If g_i, g_{max1}, g_{max2} are not the three greatest elements in g then we know it must be $g_k < g_i < g_{max1}, g_{max2}$. Then, again, every $g_j \in g(g_i, n) \setminus \{g_k, \lceil \frac{g_i - n}{2} \rceil\}$ can still describe the same gap as before, g_k can describe, for example, any subset of $g_k < g_{max1}$ cells of the gap initially described by g_{max1} and $\lceil \frac{g_i - n}{2} \rceil < g_i$ can describe any subset of $\lceil \frac{g_i - n}{2} \rceil$ contiguous cells contained in the gap initially described by g_i.

  – Suppose the n cells are allocated using only a gap bigger than g_i (g_k with $k > i$).

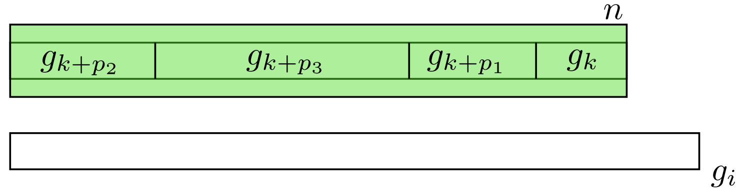


If $g_k = g_{max1}$ or $g_k = g_{max2}$ then every element in $g(g_i, n) \setminus \{\lceil \frac{g_i - n}{2} \rceil\}$ can still describe the same gap as before, for they are intact, and $\lceil \frac{g_i - n}{2} \rceil$ can describe any subset of $\lceil \frac{g_i - n}{2} \rceil < g_i$ contiguous cells contained in the gap initially described by g_i.

If $g_i < g_k < g_{max1}, g_{max2}$, then every $g_j \in g(g_i, n) \setminus \{g_k, \lceil \frac{g_i - n}{2} \rceil\}$ can still describe the same gap as before, g_k can describe, for example, any subset of $g_k < g_{max1}$ cells of the gap initially described by $g_{max1}\}$ and $\lceil \frac{g_i - n}{2} \rceil < g_i$ can describe any subset of $\lceil \frac{g_i - n}{2} \rceil$ contiguous cells contained in the gap initially described by g_i.

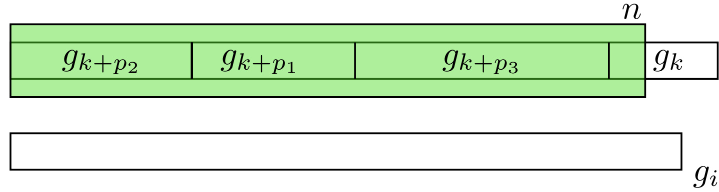
– Suppose the n cells are allocated using $r+1$ ($r \geq 1$) different gaps $g_k, g_{k+p_1}, \dots, g_{k+p_r}$ — these gaps must be placed adjacently.

If all these gaps are fully taken



then it must be $g_k + g_{k+p_1} + \cdots + g_{k+p_r} = n < g_i$ ($n < g_i$ since $\lceil \frac{g_i - n}{2} \rceil > 0$). In this case we can split the gap initially described by g_i into $r+2$ gaps of sizes $g_k, g_{k+p_1}, \dots, g_{k+p_r}$ and $\lceil \frac{g_i - n}{2} \rceil$. Thus, every element in $g(g_i, n)$ can describe either the gap it originally described, or the subgap assigned to it in the gap initially described by g_i — note that some of these $g_k, g_{k+p_1}, \dots, g_{k+p_r}$ might be g_{max1} or g_{max2} if $g_i > g_{max1}, g_{max2}$, in which case we would not even need to *use* the corresponding subgap, with the argument still holding.

If all these gaps are fully taken except for one

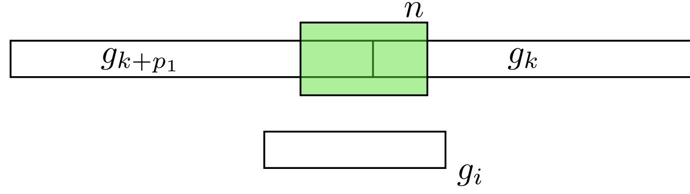


we can assume without loss of generality that it is the least of these gaps, g_k, that is not fully taken (since they are placed adjacently, the order in which they are distributed is irrelevant). It must then be $g_{k+p_1} + \cdots + g_{k+p_r} < n < g_i$, and so again we can split the gap initially described by g_i into $r+1$ subgaps of sizes $g_{k+p_1}, \dots, g_{k+p_r}$ and $\lceil \frac{g_i - n}{2} \rceil$. Thus, every element in $g(g_i, n) \setminus \{g_k\}$ can describe either the gap it originally described (if it is still a gap), or the subgap assigned to it in the gap initially described by g_i. As for g_k, since it is the least of $g_k, g_{k+p_1}, \dots, g_{k+p_r}$ with $r \geq 1$ and $g_{k+p_1} + \cdots + g_{k+p_r} < n < g_i$, it also must be $g_k < g_i$, so $g_k \leq max\{g_{max1}, g_{max2}\}$. Hence, it can describe any subgap of g_k contiguous cells contained in the gap initially described by $max\{g_{max1}, g_{max2}\}$.

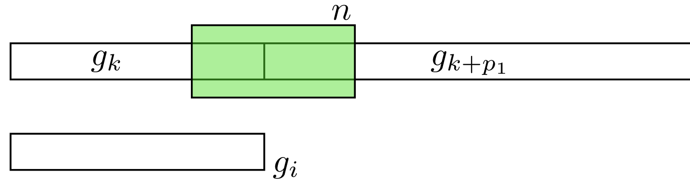
Finally, the only remaining possible case is that all these gaps are fully taken except for two. Again, we can suppose without loss of generality that it is the least two, g_k and g_{k+p_1}, that are not fully taken ($p_1 < \cdots < p_r$).

If $r = 1$ then either $g_k > g_i$ and $g_{k+p_1} > g_i$, or $g_k = g_i$ and $g_{k+p_1} > g_i$, or $g_k < g_i$ and $g_{k+p_1} > g_i$, or $g_k < g_i$ and $g_{k+p_1} = g_i$ or $g_k < g_i$ and $g_{k+p_1} < g_i$.

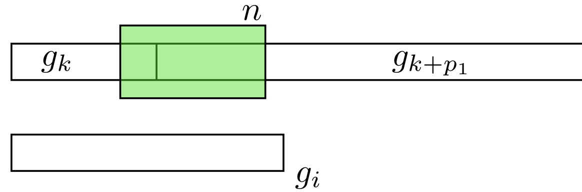
If $g_k > g_i$ and $g_{k+p_1} > g_i$



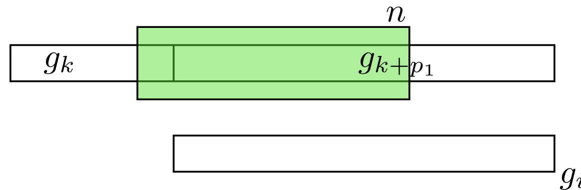
then also $g_{max1}, g_{max2} > g_i$. If this is the case and also $g_k, g_{k+p_1} < g_{max2} < g_{max1}$, then they can each describe some subset of g_{max1} and g_{max2}. If $g_{k+p_1} = g_{max2}$ then it will not be in $g(g_i, n)$ and g_k can describe some subgap of the gap initially described by g_{max2}. If $\{g_k, g_{k+p_1}\} = \{g_{max2}, g_{max1}\}$ then none of them will be in $g(g_i, n)$. Every other element in $g(g_i, n) \setminus \{g_k, g_{k+p_1}, \lceil \frac{g_i - n}{2} \rceil\}$ can describe the gap it originally described, and $\lceil \frac{g_i - n}{2} \rceil$ can describe some subgap of the gap initially described by g_i.
If $g_k = g_i$ and $g_{k+p_1} > g_i$



then at least $g_{k+p_1} \leq g_{max1}$ and we do not need to care about g_k. g_{k+p_1} can describe, if it is not g_{max1} or g_{max2} (and so not in $g(g_i, n)$), any subgap of size g_{k+p_1} of the gap initially described by g_{max1}. Every other element in $g(g_i, n) \setminus \{g_{k+p_1}, \lceil \frac{g_i - n}{2} \rceil\}$ can describe the gap it originally described, and $\lceil \frac{g_i - n}{2} \rceil$ can describe some subgap of the gap initially described by $g_i = g_k$ because, since less than n contiguous cells have been taken from it, there will still be a gap big enough.
If $g_k < g_i$ and $g_{k+p_1} > g_i$

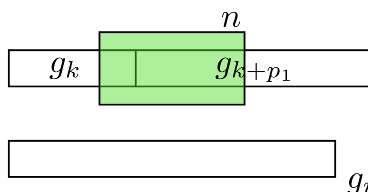


then at least $g_{k+p_1} \leq g_{max1}$ and $g_k \leq g_{max2}$. Again, g_{k+p_1} can describe, if necessary, any subgap of size g_{k+p_1} of the gap initially described by g_{max1}. g_{k+p_1} can describe any subgap of size g_k of the gap initially described by g_{max2}. Every other element in $g(g_i, n) \setminus \{g_k, g_{k+p_1}, \lceil \frac{g_i - n}{2} \rceil\}$ can describe the gap it originally described, and $\lceil \frac{g_i - n}{2} \rceil$ can describe some subgap of the gap initially described by g_i.
If $g_k < g_i$ and $g_{k+p_1} = g_i$

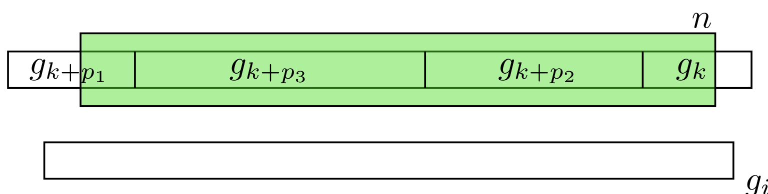


then surely $g_k \leq g_{max1}$ and we do not need to care about g_{k+p_1}. g_k can describe, if it is not g_{max1} or g_{max2} (and so not in $g(g_i, n)$), any subgap of size g_k of the gap initially described by g_{max1}. Every other element in $g(g_i, n) \setminus \{g_{k+p_1}, \lceil \frac{g_i - n}{2} \rceil\}$ can

describe the gap it originally described, and $\lceil \frac{g_i - n}{2} \rceil$ can describe some subgap of the gap initially described by $g_i = g_{k+p_1}$ because again less than n contiguous cells have been taken from it, so there will be a gap big enough.
If $g_k < g_i$ and $g_{k+p_1} < g_i$



then surely $g_k \leq g_{max2}$ and $g_{k+p_1} \leq g_{max1}$, so if g_k or g_{k+p_1} are in $g(g_i, n)$ they can be assigned to describe corresponding subgaps of g_{max2} and g_{max1}. Every other element in $g(g_i, n) \setminus \{g_k, g_{k+p_1}, \lceil \frac{g_i - n}{2} \rceil\}$ can describe the gap it originally described, and $\lceil \frac{g_i - n}{2} \rceil$ can describe some subgap of the gap initially described by g_i.
If $r > 1$



then we have $g_k \leq g_{k+p_1} \leq g_{k+p_2} + \cdots + g_{k+p_r} < n < g_i$ and so again we can split the gap initially described by g_i into r subgaps of sizes $g_{k+p_2}, \ldots, g_{k+p_r}$ and $\lceil \frac{g_i - n}{2} \rceil$. Thus, every element in $g(g_i, n) \setminus \{g_k, g_{k+p_1}\}$ can describe either the gap it originally described (if it is still a gap), or the subgap assigned to it in the gap initially described by g_i. As for g_k and g_{k+p_1}, $g_k \leq g_{k+p_1} \leq g_{k+p_2} + \cdots + g_{k+p_r} < n < g_i$ implies that, in particular, $g_k \leq g_{k+p_1} \leq g_{k+p_2} < g_i$, and so it must be $g_k \leq g_{max2}$ and $g_{k+p_1} \leq g_{max1}$. Hence, if g_k or g_{k+p_1} are in $g(g_i, n)$ they can be assigned to describe corresponding subgaps of g_{max2} and g_{max1}.

– If $|g| \geq 3$ and $n = g_i$ and g_{max1}, g_{max2} are the two biggest elements in $g \setminus \{g_i\}$ then $g(g_i, n) = g \setminus \{g_i, g_{max1}, g_{max2}\}$.

  The arguments used in the previous case work here as well, we just do not need to find a gap for $\lceil \frac{g_i - n}{2} \rceil$.

□

Due to time restrictions, the work on this adaptation of the memory model has been very limited — we have not proved soundness in this setting for the rest of rules, including the Frame Rule.

3.3 The Partial Hydrocarbon Model

This model provides a simplified representation of organic chemistry molecules (even not necessarily stable combinations of hydrogen and carbon atoms — combinations that could happen in the middle of a chemical reaction). The main feature of this resource model is that its combination operation is, unlike that of the heap model, nondeterministic.

Another particular feature of this model is that we will be interested in *liveness* of computations, rather than safety. While the focus in the memory model is on avoiding memory

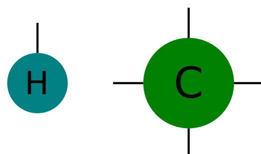
faults that may cause our programs to crash (we wanted every possible computation from a given state not to lead to `fault`), here we are interested in the *possibility* of building certain molecules by combining some others — we are happy if there is at least one computation that leads to the desired result —, which is a liveness property. We regard the nondeterminism in the commands in this model as *angelic* (refer to [7]): we trust that the right computation will somehow be chosen among all others. The kind of nondeterminism in the heap model would be *demonic* ([7]) since we had to consider that the worst could happen.

The previous work on this model ([17]) included the abstract definition of the resource and a programming language to manipulate it, together with an operational semantics which was proved to be sensible. We only conjectured Liveness Monotonicity (a property similar to Safety Monotonicity, adapted to the model) and the Frame Property. Here, we prove these properties and we see how they guarantee soundness of the Frame Rule. We had also sketched an assertion language to describe instances of the resource, and here we improve it and use it to give a whole set of command-specific axioms and rules to reason about programs that manipulate this resource, and we prove these sound.

Before looking into how arity is present in this model, let us explain how the model is defined.

We imagine molecules as undirected graphs, with nodes playing the part of atoms, and edges between nodes playing the part of bonds between atoms. Stable molecules have no free bonds, yet we can imagine instants in the middle of chemical reactions, when some bonds have been broken and others need to be formed. We represent these *partial* hydrocarbon molecules as undirected graphs with dangling edges.

Hydrocarbon molecules are built from hydrogen and carbon atoms, with hydrogen atoms having exactly one free bond and carbon atoms having exactly four.



Hence, in our graphs, each node representing a hydrogen atom will have one edge attached — either dangling or linking it to another node — and each one representing a carbon atom will have four — again, any combination of regular and dangling edges.

We can imagine now one possible intermediate state, with two very simple partial molecules, consisting each of a single carbon atom. It is quite natural to think that these two resources could be combined in various ways (supposing we do not necessarily require the result to be a stable molecule): we could just let them stay side by side, without sharing bonds, or let them share a single bond, or two, or three. This can be represented naturally as a nondeterministic binary operation.

We choose the following model to represent the graphs associated to partial hydrocarbon molecules: a partial hydrocarbon is a mapping from a finite set of nodes to a pair consisting of i) a label that records which kind of atom the node represents (`H` for hydrogen, `C` for carbon) and ii) an adjacency set that indicates whether the edges corresponding to that node are dangling or connecting it with another node.

In the model we have two different kinds of dangling edges: *free* edges and *reserved* edges. The motivation for this comes from the notion of arity, which we will explore after the following definition:

Let π_1 (respectively π_2) give the projection of the first (respectively second) component of a pair, and

$$\mathcal{MP}(X) := \{Y \mid Y \text{ is a multiset and } y \in Y \Rightarrow y \in X\}$$

be a variant of $\mathcal{P}(X)$ that considers multisets.

Definition 3.6 (Partial Hydrocarbon Set)**.** A *partial hydrocarbon set* is a mapping

$$h \in \mathbb{N} \rightharpoonup_{fin} \{\mathtt{H}, \mathtt{C}\} \times \mathcal{MP}(dom(h) \cup \{\square, \boxtimes\})$$

such that, for all $n, m \in dom(h)$ (where $dom(h) = \{n \in \mathbb{N} \mid h(n) \text{ is defined}\}$) we have

1. $\pi_1(h(n)) = \mathtt{H} \Rightarrow |\pi_2(h(n))| = 1$
2. $\pi_1(h(n)) = \mathtt{C} \Rightarrow |\pi_2(h(n))| = 4$
3. $|\{n, n, n, n\} \cap \pi_2(h(m))| = |\{m, m, m, m\} \cap \pi_2(h(n))|$

Free edges, represented by $\square$, are only available for the purposes of framing on new pieces of resource. Reserved edges (represented by $\boxtimes$), however, are reserved only for manipulation by commands. When we introduce the commands later we will see how they require a certain arity — certain capacity for extension, certain connection points — from the resource. Like the third component that we added to the states of the bounded memory model, distinguishing between reserved edges and free edges will allow the arity of a resource to be preserved under resource composition (i.e. under framing).

The adjacency set is given as a multiset because a carbon atom (with four edges) can have several free or reserved edges, or have a double or triple bond with another atom.

The combination operation in this model works in the following way: if the domains of two partial hydrocarbons h_1, h_2 are not disjoint, then $h_1 \circ h_2$ is not defined. If they are disjoint, then $h_1 \circ h_2$ will be the set of all the possible ways of taking the union of h_1 and h_2 and joining some number of their free edges together (reserved edges may not be altered).

Definition 3.7 (Hydrocarbon Model)**.** Our resource model is $\langle \mathcal{H}, \circ, e \rangle$, where
$\mathcal{H}$ is the set of all partial hydrocarbon sets.
e is the "*unit resource*". It is the partial hydrocarbon set with $dom(e) = \emptyset$.
$\circ : \mathcal{H} \times \mathcal{H} \to \mathcal{P}(\mathcal{H})$ is a commutative binary total operation, defined as follows: given $h_1, h_2 \in \mathcal{H}$, if $dom(h_1) \cap dom(h_2) \neq \emptyset$, then $h_1 \circ h_2 = \emptyset$. If $dom(h_1) \cap dom(h_2) = \emptyset$, then $h \in h_1 \circ h_2$ iff $h \in \mathcal{H}$ and

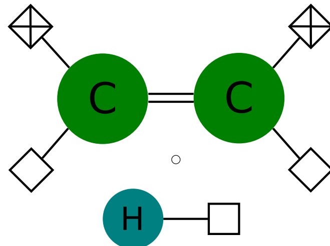
- $dom(h) = dom(h_1) \cup dom(h_2)$
- $\pi_1(h(n)) = \pi_1(h_i(n))$ for all $n \in dom(h_i),\ i = 1, 2$
- $\pi_2(h(n)) = \pi_2(h_i(n)) \setminus X_n \cup X'_n$ if $n \in dom(h_i), \{i, j\} = \{1, 2\}$
  for some X_n, X'_n with $X_n \subseteq \mathcal{MP}(\{\square\})$, $X'_n \subseteq dom(h_j)$ and $|X'_n| = |X_n|$.

We can observe in this definition that if both of the graphs being combined have a node with a free edge, then $\circ$ allows these nodes to be bonded together (losing the corresponding free edges). Nodes within the same original $dom(h_i)$ that were not bonded together before the combination will not be bonded together after either.

Example 3.1. *Let* h, h' *with*

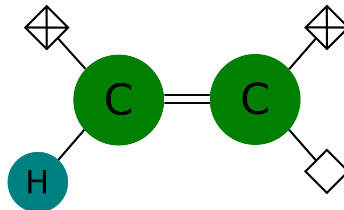
$$dom(h) = \{1, 2\}, \quad h(1) = (\mathtt{C}, \{2, 2, \boxtimes, \square\}), \quad h(2) = (\mathtt{C}, \{1, 1, \boxtimes, \square\})$$

$$dom(h') = \{3\}, \quad h'(3) = (\mathtt{H}, \{\square\})$$

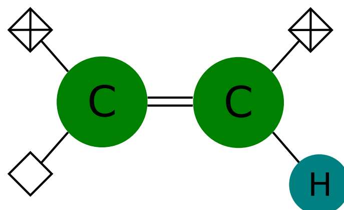


Then $h \circ h' = \{h_1, h_2, h_3\}$*, where* h_1, h_2, h_3 *are given as follows:*

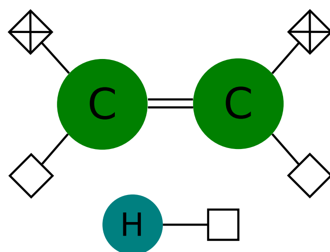
$$dom(h_1) = \{1, 2, 3\}$$
$$h_1(1) = (\mathtt{C}, \{2, 2, \boxtimes, 3\}), \quad h_1(2) = (\mathtt{C}, \{1, 1, \boxtimes, \square\}), \quad h_1(3) = (\mathtt{H}, \{1\})$$



$$dom(h_2) = \{1, 2, 3\}$$
$$h_2(1) = (\mathtt{C}, \{2, 2, \boxtimes, \square\}), \quad h_2(2) = (\mathtt{C}, \{1, 1, \boxtimes, 3\}), \quad h_2(3) = (\mathtt{H}, \{2\})$$



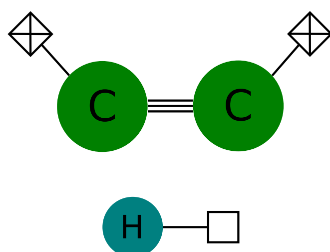
$$dom(h_3) = \{1, 2, 3\}$$
$$h_3(1) = (\mathtt{C}, \{2, 2, \boxtimes, \square\}), \quad h_3(2) = (\mathtt{C}, \{1, 1, \boxtimes, \square\}), \quad h_3(3) = (\mathtt{H}, \{\square\})$$



(h_3 represents the possibility of not bonding any free edges.)

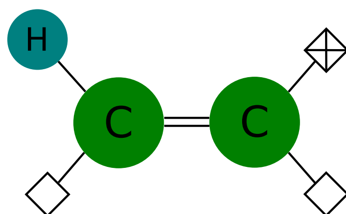
Since nodes in the same original partial molecule cannot be bonded as a result of the combination operation, $h_4 \notin h \circ h'$:

$$dom(h_4) = \{1, 2, 3\}$$
$$h_4(1) = (\mathtt{C}, \{2, 2, \boxtimes, 2\}), \quad h_4(2) = (\mathtt{C}, \{1, 1, \boxtimes, 1\}), \quad h_4(3) = (\mathtt{H}, \{\square\})$$

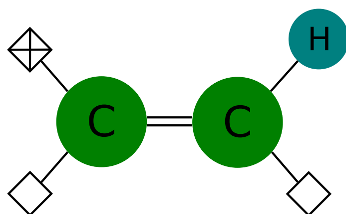


And since reserved edges cannot be bonded this way either, none of h_5, h_6 are in $h \circ h'$:

$$dom_5(h_5) = \{1, 2, 3\}$$
$$h_5(1) = (\mathtt{C}, \{2, 2, 3, \square\}), \quad h_5(2) = (\mathtt{C}, \{1, 1, \boxtimes, \square\}), \quad h_5(3) = (\mathtt{H}, \{1\})$$



$$dom(h_6) = \{1, 2, 3\}$$
$$h_6(1) = (\mathtt{C}, \{2, 2, \boxtimes, \square\}), \quad h_6(2) = (\mathtt{C}, \{1, 1, 3, \square\}), \quad h_6(3) = (\mathtt{H}, \{2\})$$



It is also worth pointing out that, since the first condition for h to be in $h_1 \circ h_2$ is to be in $\mathcal{H}$, it is necessary that $|X_n \cap \pi_2(h_i(n))| = |X'_n|$ $(n \in dom(h_i))$ — that is, for any node n the number of free edges that we remove from its adjacency set must be the same as the number of new bonds being created for that same node — and $n \in X'_m \Leftrightarrow m \in X'_n$ — when we bond two nodes together, each of them must enter the adjacency et of the other.

To talk about associativity, we must first extend our operation to act on $\mathcal{P}(\mathcal{H}) \times \mathcal{P}(\mathcal{H})$ instead of just $\mathcal{H} \times \mathcal{H}$, for obvious reasons. We will have $\circ : \mathcal{P}(\mathcal{H}) \times \mathcal{P}(\mathcal{H}) \to \mathcal{P}(\mathcal{H})$ as expected: given $A, B \in \mathcal{P}(\mathcal{H})$

$$A \circ B = \{h' \circ h \in \mathcal{H} \mid h' \in h_A \circ h_B \text{ for some } h_A \in A, h_B \in B$$

The following result was proved in [17]:

Lemma 3.9 (Associativity)**.** *The model given by $\langle \mathcal{H}, \circ, e \rangle$ is associative: for any $A, B, C \in \mathcal{P}(\mathcal{H})$, we have*

$$(A \circ B) \circ C = A \circ (B \circ C)$$

Proof.

– $(A \circ B) \circ C \subseteq A \circ (B \circ C)$

Let $h \in (A \circ B) \circ C$, then $\exists h_A \in A, h_B \in B, h_{AB} \in h_A \circ h_B, h_C \in C$ such that $h \in h_{AB} \circ h_C$

$dom(h) = dom(h_{AB}) \cup dom(h_C) = dom(h_A) \cup dom(h_B) \cup dom(h_C)$

$$\pi_1(h(n)) = \begin{cases} \pi_1(h_AB(n)) = \pi_1(h_A(n)) & \text{if } n \in \ dom(h_A) \\ \pi_1(h_AB(n)) = \pi_1(h_B(n)) & \text{if } n \in \ dom(h_B) \\ \pi_1(h_C(n)) & \text{if } n \in \ dom(h_C) \end{cases}$$

$$\pi_2(h(n)) = \begin{cases} \pi_2(h_{AB}(n)) \setminus X_{C,n} \cup X'_{C,n} & \\ \quad = (\pi_2(h_A(n)) \setminus X_{B,n} \cup X'_{B,n}) \setminus X_{C,n} \cup X'_{C,n} & \text{if } n \in \ dom(h_A) \\ \pi_2(h_{AB}(n)) \setminus X_{C,n} \cup X'_{C,n} & \\ \quad = (\pi_2(h_B(n)) \setminus X_{A,n} \cup X'_{A,n}) \setminus X_{C,n} \cup X'_{C,n} & \text{if } n \in \ dom(h_B) \\ \pi_2(h_C(n)) \setminus X_{AB,n} \cup X'_{AB,n} & \\ \quad = \pi_2(h_C(n)) \setminus (X_{A,n} \cup X_{B,n}) \cup (X'_{A,n} \cup X'_{B,n}) & \text{if } n \in \ dom(h_C) \end{cases}$$

(the additional subindices remind us that $X'_{A,n} \subseteq dom(h_A)$, $X'_{B,n} \subseteq dom(h_B)$, $X'_{C,n} \subseteq dom(h_C)$; the subindices for $X_{A,n}, X_{B,n}, X_{C,n}$ help matching them with their corresponding $X'_{-,n}$ so that $|X_{S,n}| = |X'_{S,n}|$ for $S = A, B, C$.)

Because $X_{C,n} \cap X'_{B,n} = \emptyset$ and $X_{C,n} \cap X'_{A,n} = \emptyset$ (since $X_{C,n} \subseteq \{\square, \square, \square, \square\}$, $X'_{B,n} \subseteq dom(h_B)$ and $X'_{A,n} \subseteq dom(h_A)$) we have

$$\pi_2(h(n)) = \begin{cases} \pi_2(h_A(n)) \setminus (X_{B,n} \cup X_{C,n}) \cup (X'_{B,n} \cup X'_{C,n}) & \text{if } n \in \ dom(h_A) \\ \pi_2(h_B(n)) \setminus (X_{A,n} \cup X_{C,n}) \cup (X'_{A,n} \cup X'_{C,n}) & \text{if } n \in \ dom(h_B) \\ \pi_2(h_C(n)) \setminus (X_{A,n} \cup X_{B,n}) \cup (X'_{A,n} \cup X'_{B,n}) & \text{if } n \in \ dom(h_C) \end{cases}$$

We are going to construct $h' \in A \circ (B \circ C)$ in such a way that $h' = h$.

Let $h_{BC} \in h_B \circ h_C$ such that

$$\pi_2(h_{BC}(n)) = \begin{cases} \pi_2(h_B(n)) \setminus X_{C,n} \cup X'_{C,n} & \text{if } n \in \ dom(h_B) \\ \pi_2(h_C(n)) \setminus X_{B,n} \cup X'_{B,n} & \text{if } n \in \ dom(h_C) \end{cases}$$

Now let $h' \in h_A \circ h_{BC}$ with

$$\pi_2(h'(n)) = \begin{cases} \pi_2(h_A(n)) \setminus X_{BC,n} \cup X'_{BC,n} \\ \quad = \pi_2(h_A(n)) \setminus (X_{B,n} \cup X_{C,n}) \cup (X'_{B,n} \cup X'_{C,n}) & \text{if } n \in \ dom(h_A) \\ \pi_2(h_{BC}(n)) \setminus X_{A,n} \cup X'_{A,n} \\ \quad = (\pi_2(h_B(n)) \setminus X_{C,n} \cup X'_{C,n}) \setminus X_{A,n} \cup X'_{A,n} & \text{if } n \in \ dom(h_B) \\ \pi_2(h_{BC}(n)) \setminus X_{A,n} \cup X'_{A,n} \\ \quad = (\pi_2(h_C(n)) \setminus X_{B,n} \cup X'_{B,n}) \setminus X_{A,n} \cup X'_{A,n} & \text{if } n \in \ dom(h_C) \end{cases}$$

(note that the $X_{S,n}, X'_{S,n}$ — $S = A, B, C$ — sets are the same as they for h. The fact that h was well defined implies that h' and h_{BC} are well defined too.)

Again, because $X'_{C,n} \cap X_{A,n} = \emptyset$ and $X'_{B,n} \cap X_{A,n} = \emptyset$ (like before, $X_{A,n} \subseteq \{\square, \square, \square, \square\}$, $X'_{B,n} \subseteq dom(h_B)$ and $X'_{C,n} \subseteq dom(h_C)$), we have

$$\pi_2(h'(n)) = \begin{cases} \pi_2(h_A(n)) \setminus (X_{B,n} \cup X_{C,n}) \cup (X'_{B,n} \cup X'_{C,n}) & \text{if } n \in \ dom(h_A) \\ \pi_2(h_B(n)) \setminus (X_{A,n} \cup X_{C,n}) \cup (X'_{A,n} \cup X'_{C,n}) & \text{if } n \in \ dom(h_B) \\ \pi_2(h_C(n)) \setminus (X_{A,n} \cup X_{B,n}) \cup (X'_{A,n} \cup X'_{B,n}) & \text{if } n \in \ dom(h_C) \end{cases}$$

So $h = h' \in h_A \circ h_{BC} \subseteq A \circ (B \circ C)$.

– $A \circ (B \circ C) \subseteq (A \circ B) \circ C$

It follows from $(A \circ B) \circ C \subseteq A \circ (B \circ C)$ and $\circ$ being commutative:

$$A \circ (B \circ C) = (C \circ B) \circ A \subseteq C \circ (B \circ A) = (A \circ B) \circ C$$

□

3.3.1 Commands and operational semantics

We consider the following language of commands:

$$C ::= break(N, M) \mid bond(N, M) \mid C; C \mid C + C \qquad (N, M \in \{\mathtt{H}, \mathtt{C}\})$$

with the operational semantics in Figure 8. We do not consider $bond(\mathtt{H}, \mathtt{C})$ or $break(\mathtt{H}, \mathtt{C})$, only $bond(\mathtt{C}, \mathtt{H})$ or $break(\mathtt{C}, \mathtt{H})$, because they would be redundant.

The substitution notation is the same as the one used in section 2.

The command $break(N, M)$ breaks a single bond between a node of kind N and a node of kind M (that are currently bonded), and adds a reserved edge to each one instead. $bond(N, M)$ creates a bond between a node of kind N and a node of kind M, provided each of them has a reserved edge. $C; C$ is sequential composition and $C + C$ is nondeterministic choice.

As in the memory model, we consider configurations of the form $\langle C, h\rangle$ and $\langle h\rangle$, and the special configuration `fault`. As we mentioned in the introduction to this subsection, we are interested in the existence of ways to build certain molecules from some other molecules, rather fault avoidance (safety). Therefore, the fact that our operational semantics allows every command to fault is not a problem.

As can be observed, when a bond between two nodes is broken, it is substituted by two corresponding reserved edges. Note that these two edges will be reserved rather than free

$$\frac{n,m \in dom(h) \quad h(n) = (N, E_n) \quad h(m) = (M, E_m) \quad \boxtimes \in E_n \cap E_m}{\langle bond(N,M), h\rangle \rightsquigarrow \langle h[n \mapsto (N, (E_n \setminus \{\boxtimes\}) \cup \{m\}), m \mapsto (M, (E_m \setminus \{\boxtimes\}) \cup \{n\})]\rangle}$$

$$\frac{n,m \in dom(h) \quad h(n) = (N, E_n) \quad h(m) = (M, E_m) \quad m \in E_n, n \in E_m}{\langle break(N,M), h\rangle \rightsquigarrow \langle h[n \mapsto (N, (E_n \setminus \{m\}) \cup \{\boxtimes\}), m \mapsto (M, (E_m \setminus \{n\}) \cup \{\boxtimes\})]\rangle}$$

$$\frac{\langle C_1, h\rangle \rightsquigarrow \langle h''\rangle \quad \langle C_2, h''\rangle \rightsquigarrow \langle h'\rangle}{\langle C_1; C_2, h\rangle \rightsquigarrow \langle h'\rangle}$$

$$\frac{\langle C_1, h\rangle \rightsquigarrow \langle h'\rangle}{\langle C_1 + C_2, h\rangle \rightsquigarrow \langle h'\rangle} \qquad \frac{\langle C_2, h\rangle \rightsquigarrow \langle h'\rangle}{\langle C_1 + C_2, h\rangle \rightsquigarrow \langle h'\rangle}$$

$$\frac{}{\langle C, h\rangle \rightsquigarrow \texttt{fault}}$$

Figure 8: Operational Semantics of Hydrocarbon Bonding Commands

edges. When we break a bond we want to maintain control over the dangling edges, just like when we deallocate memory in the bounded memory model we retain ownership of the cell.

It was also proved in [17] that this operational semantics is sensible, in the sense that it verifies the following Lemma.

Lemma 3.10. *If $h \in \mathcal{H}$ and $\langle C, h\rangle \rightsquigarrow \langle h'\rangle$, then $h' \in \mathcal{H}$.*

Proof. Structural induction on C:

$C = bond(N, M)$

In this case

$$h' = h[n \mapsto (N, (E_n \setminus \{\boxtimes\}) \cup \{m\}), m \mapsto (M, (E_m \setminus \{\boxtimes\}) \cup \{n\})]$$

so we have $h' \in \mathbb{N} \rightharpoonup_{fin} \{H, C\} \times \mathcal{MP}(dom(h') \cup \{\square, \boxtimes\})$, since

- $dom(h) = dom(h')$. Let us abbreviate $A = dom(h) = dom(h')$.
- $h'(n) = (N, (E_n \setminus \{\boxtimes\}) \cup \{m\}) \in \{H, C\} \times \mathcal{MP}(A \cup \{\square, \boxtimes\})$, because $h \in \mathcal{H}$ implies $N \in \{H, C\}$ and $E_n \in \mathcal{MP}(A \cup \{\square, \boxtimes\})$, and so $(E_n \setminus \{\boxtimes\}) \cup \{m\} \in \mathcal{MP}(A \cup \{\square, \boxtimes\})$ (because $m \in A$).
- $h'(m) = (M, (E_m \setminus \{\boxtimes\}) \cup \{n\}) \in \{H, C\} \times \mathcal{MP}(A \cup \{\square, \boxtimes\})$ — analogue to the previous case.
- if $k \in A \setminus \{n, m\}$ then $h'(k) = h(k) \in \{H, C\} \times \mathcal{MP}(A \cup \{\square, \boxtimes\})$, since $h \in \mathcal{H}$.

and, for all $k, k' \in dom(h)$ $(k \neq k')$ we have

- $\pi_1(h'(k)) = \pi_1(h(k)) = H \Rightarrow |\pi_2(h'(k))| = |\pi_2(h(k))| = 1$
- $\pi_1(h'(k)) = \pi_1(h(k)) = C \Rightarrow |\pi_2(h'(k))| = |\pi_2(h(k))| = 4$
- $|\{k, k, k, k\} \cap \pi_2(h'(k'))\}| = |\{k', k', k', k'\} \cap \pi_2(h'(k))|$
  - $\{k, k'\} \neq \{n, m\}$
    $|\{k, k, k, k\} \cap \pi_2(h'(k'))\}| = |\{k, k, k, k\} \cap \pi_2(h(k'))\}|$
    $= |\{k', k', k', k'\} \cap \pi_2(h(k))| = |\{k', k', k', k'\} \cap \pi_2(h'(k))|$

- $\{k,k'\} = \{n,m\}$
  $|\{n,n,n,n\} \cap \pi_2(h'(m))\}| = |\{n,n,n,n\} \cap \pi_2(h(m))\}| + 1$
  $= |\{m,m,m,m\} \cap \pi_2(h(n))| + 1 = |\{m,m,m,m\} \cap \pi_2(h'(n))|$
- $k = n, k' \neq m$
  $|\{n,n,n,n\} \cap \pi_2(h'(k'))\}| = |\{k,k,k,k\} \cap \pi_2(h(k'))\}|$
  $= |\{k',k',k',k'\} \cap \pi_2(h(n))| = |\{k',k',k',k'\} \cap \pi_2(h'(n))|$
- $k = m, k' \neq n$ analogue to the previous case.

$C = \mathit{break}(N,M)$

In this case

$$h' = h[n \mapsto (N, (E_n \setminus \{m\}) \cup \{\boxtimes\}), m \mapsto (M, (E_m \setminus \{n\}) \cup \{\boxtimes\})]$$

so we have $h' \in \mathbb{N} \rightharpoonup_{fin} \{H,C\} \times \mathcal{MP}(\mathit{dom}(h') \cup \{\Box, \boxtimes\})$, since

- $\mathit{dom}(h) = \mathit{dom}(h')$. Let us abbreviate $A = \mathit{dom}(h) = \mathit{dom}(h')$.
- $h'(n) = (N, (E_n \setminus \{\boxtimes\}) \cup \{m\}) \in \{H,C\} \times \mathcal{MP}(A \cup \{\Box, \boxtimes\})$, because $h \in \mathcal{H}$ implies $N \in \{H,C\}$ and $E_n \in \mathcal{MP}(A \cup \{\Box, \boxtimes\})$, and so $(E_n \setminus \{m\}) \cup \{\boxtimes\} \in \mathcal{MP}(A \cup \{\Box, \boxtimes\})$.
- $h'(m) = (M, (E_m \setminus \{n\}) \cup \{\boxtimes\}) \in \{H,C\} \times \mathcal{MP}(A \cup \{\Box, \boxtimes\})$ — analogue to the previous case.
- if $k \in A \setminus \{n,m\}$ then $h'(k) = h(k) \in \{H,C\} \times \mathcal{MP}(A \cup \{\Box, \boxtimes\})$, since $h \in \mathcal{H}$.

and, for all $k, k' \in \mathit{dom}(h)$ $(k \neq k')$ we have

- $\pi_1(h'(k)) = \pi_1(h(k)) = H \Rightarrow |\pi_2(h'(k))| = |\pi_2(h(k))| = 1$
- $\pi_1(h'(k)) = \pi_1(h(k)) = C \Rightarrow |\pi_2(h'(k))| = |\pi_2(h(k))| = 4$
- $|\{k,k,k,k\} \cap \pi_2(h'(k'))\}| = |\{k',k',k',k'\} \cap \pi_2(h'(k))|$
  - $\{k,k'\} \neq \{n,m\}$
    $|\{k,k,k,k\} \cap \pi_2(h'(k'))\}| = |\{k,k,k,k\} \cap \pi_2(h(k'))\}|$
    $= |\{k',k',k',k'\} \cap \pi_2(h(k))| = |\{k',k',k',k'\} \cap \pi_2(h'(k))|$
  - $\{k,k'\} = \{n,m\}$
    $|\{n,n,n,n\} \cap \pi_2(h'(m))\}| = |\{n,n,n,n\} \cap \pi_2(h(m))\}| - 1$
    $= |\{m,m,m,m\} \cap \pi_2(h(n))| - 1 = |\{m,m,m,m\} \cap \pi_2(h'(n))|$
  - $k = n, k' \neq m$
    $|\{n,n,n,n\} \cap \pi_2(h'(k'))\}| = |\{k,k,k,k\} \cap \pi_2(h(k'))\}|$
    $= |\{k',k',k',k'\} \cap \pi_2(h(n))| = |\{k',k',k',k'\} \cap \pi_2(h'(n))|$
  - $k = m, k' \neq n$ analogue to the previous case.

$C = C_1; C_2$

If we have $\langle C_1; C_2, h\rangle \rightsquigarrow \langle h'\rangle$ this means that there is some h'' with $\langle C_1, h\rangle \rightsquigarrow \langle h''\rangle$ and $\langle C_2, h''\rangle \rightsquigarrow \langle h'\rangle$. Since $\in \mathcal{H}$ and $\langle C_1, h\rangle \rightsquigarrow \langle h''\rangle$, by induction hypothesis we have $h'' \in \mathcal{H}$. But then, since $\langle C_2, h''\rangle \rightsquigarrow \langle h'\rangle$, again by induction hypothesis we have $h' \in \mathcal{H}$.

$C = C_1 + C_2$

If we have $\langle C_1 + C_2, h\rangle \rightsquigarrow \langle h'\rangle$ this means that either $\langle C_1, h\rangle \rightsquigarrow \langle h'\rangle$ or $\langle C_2, h\rangle \rightsquigarrow \langle h'\rangle$. In either case, by induction hypothesis we have $h' \in \mathcal{H}$.

$\Box$

3.3.2 Assertion Language

Like with the memory model, we want to perform reasoning on "programs" that manipulate partial hydrocarbons. For this, we need a language that describes the objects in our model accurately enough, yet as simply as possible.

Definition 3.8 (Hydrocarbon formulas)**.** We consider the following syntax for the assertions or formulas that we will use to describe partial hydrocarbons. A different syntax was suggested in [17], but it was not expressive enough.

$$\begin{array}{rcl} A & ::= & \top \mid \bot \mid H_i\{x\} \mid C_i\{x_1, x_2, x_3, x_4\} \mid \neg A \mid A \wedge A' \mid A \vee A' \mid A \rightarrow A' \\ & & \mid A * A' \mid A \mathrel{-\!\!*} A' \end{array}$$

where $i \in \mathbb{N}, \quad x, x_1, x_2, x_3, x_4 \in \mathbb{N} \cup \{\boxtimes, \square\}$.

We will write $h \models A$ to denote satisfaction of the assertion A by the partial hydrocarbon set h. Let the semantics of A be

$$[\![A]\!] \stackrel{def}{=} \{h \mid h \models A\}$$

with $h \models A$ defined in Figure 9. $\mathtt{H}_i\{x\}$ and $\mathtt{C}_i\{x_1, x_2, x_3, x_4\}$ are used to denote individual hydrogen and carbon atoms, in a way similar to how $E \mapsto E'$ is used within the memory model to denote individual memory cells.

$$\begin{array}{rcl} h \models \top & \text{always} & \\ h \models \bot & \text{never} & \\ h \models \mathtt{H}_i\{x\} & \Leftrightarrow & dom(h) = \{i\}, h(i) = (\mathtt{H}, \{x\}) \\ h \models \mathtt{C}_i\{x_1, x_2, x_3, x_4\} & \Leftrightarrow & dom(h) = \{i\}, h(i) = (\mathtt{C}, \{x_1, x_2, x_3, x_4\}) \\ h \models \neg A & \Leftrightarrow & h \not\models A \\ h \models A \wedge A' & \Leftrightarrow & h \models A \text{ and } h \models A' \\ h \models A \vee A' & \Leftrightarrow & h \models A \text{ or } h \models A' \\ h \models A \rightarrow A' & \Leftrightarrow & h \models A \text{ implies } h \models A' \\ h \models A * A' & \Leftrightarrow & \exists h', h''.\ h \in h' \circ h'',\ h' \models A \text{ and } h'' \models A' \\ h \models A \mathrel{-\!\!*} A' & \Leftrightarrow & \forall h', h''.\ \text{if } h' \models A \text{ and } h'' \in h \circ h' \text{ then } h'' \models A' \end{array}$$

Figure 9: Semantics of assertions for the Partial Hydrocarbon model

3.3.3 Locality Conditions. Soundness of the Frame Rule.

We need to adapt the locality conditions to make them appropriate for our model. First, we will specify the notion of *correctness* that we are going to consider.

Definition 3.9 (Correctness)**.** $\{A\}\ C\ \{A'\}$ holds iff for all $h \in \mathcal{H}$ with $h \models A$ there is some h' such that $\langle C, h\rangle \rightsquigarrow \langle h'\rangle$ and $h' \models A'$.

It talks about *total* correctness, since in our model we do not have non-terminating programs. Note that we do not require every possible result to satisfy the postcondition, but just one. This notion of correctness is in accordance with our interest in liveness properties.

In section 2 we talked about a certain notion of locality of the commands that manipulate the memory: a command would only affect a specific part of the resource, no matter how much bigger we make it. Even though we do not have locality in such a strict sense — the command *bond*(H, H), for example, can nondeterministically bond any two hydrogen atoms in a partial hydrocarbon, as long as they both have a reserved edge —, we still have some kind of locality

with our commands. They act only on a certain part of the resource which can be chosen, in general, in many ways, so the precise part of the resource that is modified is not always determined. However, the magnitude of the effect the command produces is always the same.

We now formally define the notion of liveness we have been referring to since the beginning of the subsection.

Definition 3.10 (Liveness)**.** $\langle C, h\rangle$ is *live* iff there is some h' such that $\langle C, h\rangle \rightsquigarrow \langle h'\rangle$.

As we said, our commands act locally in the sense that they only affect a small portion of the resource. This portion, however, is not predetermined, and can be chosen nondeterministically. We will not be interested in all the computations from a particular configuration, but just in some of them (the operational semantics even allows any random configuration to fault). Thus, we consider that the nondeterminism in our commands is, as we announced in the introduction to this section, angelic.

Since we look at liveness of configurations instead of safety, it makes sense that we consider a *Liveness Monotonicity* property instead of *Safety Monotonicity.*

Nondeterministic Liveness Monotonicity. If $\langle C, h_2\rangle$ is live and $h \in h_1 \circ h_2$, then $\langle C, h\rangle$ is live too.

Nondeterministic Frame Property. If $\langle C, h_2\rangle$ is live and $h \in h_1 \circ h_2$ then

$$\langle C, h_2\rangle \rightsquigarrow \langle h_2'\rangle \Rightarrow \exists h' \in h_1 \circ h_2'.\ \langle C, h\rangle \rightsquigarrow \langle h'\rangle$$

Before explaining this new formulation of the Frame Property, let us remember its original formulation in [2] for the concrete heap model: *If* $\langle C, (s, h_2)\rangle$ *does not fault and* $\langle C, (s, h_1 \cdot h_2)\rangle \rightsquigarrow^* \langle (s', h')\rangle$*, then there exists* $h_2' \in Heaps$ *such that* $\langle C, (s, h_2)\rangle \rightsquigarrow \langle (s', h_2')\rangle$ *and* $h' = h_1 \cdot h_2'$. It says that, given a state $((s, h_2))$ with the necessary resource to execute the program, every execution of the program in a larger state $((s, h_1 \cdot h_2))$ can be traced back to an execution in the original small state.

In this model, we cannot track a computation on a big state back to a smaller state because, due to the behaviour of our commands, it might affect a bond between atoms that are not in the small state. What we can do is mimic a computation on a small state on a bigger state. The difference is that, while memory allocation (the only nondeterministic command in the memory model) can nondeterministically allocate a cell that will not available for allocation in the bigger state (because it might be already allocated, as we saw in an example in section 2), in this model the commands can nondeterministically affect only bonds between atoms that are part of the partial hydrocarbon, and any bond in a small state will be available in the big state, but not the other way around.

Therefore, what our new version says is that, given a state h_2 with the necessary resource to execute a program C, and one of the possible outcomes h_2' of running C on h_2, if we choose one specific way, h, of combining h_2 with another (compatible) state h_1, then at least one of the outcomes of running C on h will be in the set $h_2' \circ h_1$.

It turns out that, with this new formulation of the properties, Nondeterministic Liveness Monotonicity is a direct consequence of the Nondeterministic Frame Property.

Lemma 3.11. *If the Nondeterministic Frame Property (as given above) holds, then so does Nondeterministic Liveness Monotonicity.*

Proof. Suppose $\langle C, h_2\rangle$ is live and $h \in h_1 \circ h_2$. Since $\langle C, h_2\rangle$ is live, there is some h_2' such that $\langle C, h_2\rangle \rightsquigarrow \langle h_2'\rangle$. But then, by the Nondeterministic Frame Property, $\exists h' \in h_1 \circ h_2'$ with $\langle C, h\rangle \rightsquigarrow \langle h'\rangle$, so $\langle C, h\rangle$ is live too. □

So in this case, instead of using both properties, we need to prove that soundness of the Frame Rule

$$\frac{\{P\}\ C\ \{Q\}}{\{F * P\}\ C\ \{F * Q\}}$$

follows from the Frame Property alone. We also need to prove that our model does indeed verify our new Frame Property.

Lemma 3.12 (Frame Property). *If $\langle C, h_2\rangle$ is live and $h \in h_1 \circ h_2$ then*

$$\langle C, h_2\rangle \rightsquigarrow \langle h_2'\rangle \Rightarrow \exists h' \in h_1 \circ h_2'.\ \langle C, h\rangle \rightsquigarrow \langle h'\rangle$$

Proof. Reasoning by structural induction:

- $C = bond(N, M)$

  In the operational semantics we can observe that, if $\langle bond(N, M), h_2\rangle$ is live and, in particular, $\langle bond(N, M), h_2\rangle \rightsquigarrow \langle h_2'\rangle$, then there must be some $n, m \in dom(h_2)$ such that

  $$h_2(n) = (N, E_n),\ h(m) = (M, E_m),\ \boxtimes \in E_n \cap E_m$$

  and

  $$h_2' = h_2[n \mapsto (N, (E_n - \{\boxtimes\}) \cup \{m\}), m \mapsto (N, (E_m - \{\boxtimes\}) \cup \{n\})]$$

  From the definition of $\circ$ follows that

  $$dom(h) = dom(h_1) \cup dom(h_2)$$

  $$\pi_1(h(n)) = \pi_1(h_2(n)),\ \pi_1(h(m)) = \pi_1(h_2(m))$$

  $$\boxtimes \in \pi_2(h(n)) \Leftrightarrow \boxtimes \in \pi_2(h_2(n)),\ \boxtimes \in \pi_2(h(m)) \Leftrightarrow \boxtimes \in \pi_2(h_2(m))$$

  Hence, it follows that $n, m \in dom(h)$ and

  $$h(n) = (N, E_n'),\ h(m) = (M, E_m'),\ \boxtimes \in E_n' \cap E_m'$$

  so $\langle bond(N, M), h\rangle \rightsquigarrow \langle h'\rangle$ with

  $$h' = h[n \mapsto (N, (E_n' - \{\boxtimes\}) \cup \{m\}), m \mapsto (N, (E_m' - \{\boxtimes\}) \cup \{n\})]$$

  Let us observe that, since $h \in h_1 \circ h_2$, it must be the case that for all $k \in dom(h_i)\ \ i = 1, 2$

  $$\pi_1(h(k)) = \pi_1(h_i(k))\ \ \forall k \in dom(h_i)$$

  $$\exists X_k, X_k'.\ \pi_2(h(k)) = (\pi_2(h_i(k)) - X_k) \cup X_k'$$

  All we need to show is that $h' \in h_1 \circ h_2'$. For this we need to check that

  - $h' \in \mathcal{H}$ which follows from the fact that $h \in \mathcal{H}$ and $\langle bond(N, M), h\rangle \rightsquigarrow \langle h'\rangle$, by Lemma 3.10.
  - $dom(h') = dom(h_1) \cup dom(h_2')$:

    $$dom(h') = dom(h) = dom(h_1) \cup dom(h_2) = dom(h_1) \cup dom(h_2')$$

  - $\pi_1(h'(k)) = \pi_1(h_1(k))$ if $k \in dom(h_1)$ and $\pi_1(h'(k)) = \pi_1(h_2'(k))$ if $k \in dom(h_2')$:

    $$\pi_1(h'(k)) = \pi_1(h(k)) = \begin{cases} \pi_1(h_1(k)) & \forall k \in dom(h_1) \\ \pi_1(h_2(k)) = \pi_1(h_2'(k)) & \forall k \in dom(h_2) = dom(h_2') \end{cases}$$

- $\pi_2(h'(k)) = \pi_2(h_1(k)) - X_k \cup X'_k$ for some X_k, X'_k with
  - $X_k \subseteq \mathcal{MP}(\{\square\})$
  - $X'_k \subseteq dom(h'_2)$

  if $k \in dom(h_1)$, and $\pi_2(h'(k)) = \pi_2(h'_2(k)) - X_k \cup X'_k$ for some X_k, X'_k with
  - $X_k \subseteq \mathcal{MP}(\{\square\})$
  - $X'_k \subseteq dom(h_1)$

  if $k \in dom(h'_2)$.

  This is also true, because:

$$\pi_2(h'(k)) = \pi_2(h(k)) = \begin{cases} (\pi_2(h_1(k)) - X_k) \cup X'_k & \forall k \in dom(h_1) \\ (\pi_2(h_2(k)) - X_k) \cup X'_k = (\pi_2(h'_2(k)) - X_k) \cup X'_k & \\ & \forall k \in dom(h'_2) - \{n, m\} \end{cases}$$

$$\begin{aligned} \pi_2(h'(n)) &= (\pi_2(h(n)) - \{\boxtimes\}) \cup \{m\} \\ &= (((\pi_2(h_2(n)) - X_n) \cup X'_n) - \{\boxtimes\}) \cup \{m\} \\ &\overset{*}{=} (((\pi_2(h_2(n)) - \{\boxtimes\}) \cup \{m\}) - X_n) \cup X'_n \\ &= (\pi_2(h'_2(n)) - X_n) \cup X'_n \end{aligned}$$

$$\begin{aligned} \pi_2(h'(m)) &= (\pi_2(h(m)) - \{\boxtimes\}) \cup \{n\} \\ &= (((\pi_2(h_2(m)) - X_m) \cup X'_m) - \{\boxtimes\}) \cup \{n\} \\ &\overset{**}{=} (((\pi_2(h_2(m)) - \{\boxtimes\}) \cup \{n\}) - X_m) \cup X'_m \\ &= (\pi_2(h'_2(m)) - X_m) \cup X'_m \end{aligned}$$

(*), (**): these two equations hold since $X_n,\ X'_n,\ \{m\},\ \{\boxtimes\}$ are all disjoint with each other, and so are $X_m,\ X'_m,\ \{n\},\ \{\boxtimes\}$.

And by definition of $\circ$, we can say that $h' \in h_1 \circ h'_2$.

- $C = \mathit{break}(N, M)$

In the operational semantics we can observe that if $\langle \mathit{break}(N, M), h_1\rangle$ is live and, in particular, $\langle \mathit{break}(N, M), h_2\rangle \rightsquigarrow \langle h'_2\rangle$, then there must be some $n, m \in dom(h_2)$ such that

$$h(n) = (N, E_n),\ h(m) = (M, E_m),\ m \in E_n, n \in E_m$$

and

$$h'_2 = h_2[n \mapsto (N, (E_n - \{m\}) \cup \{\boxtimes\}), m \mapsto (N, (E_m - \{n\}) \cup \{\boxtimes\})]$$

We have, again by the definition of $\circ$, that

$$dom(h) = dom(h_1) \cup dom(h_2)$$

$$\pi_1(h(n)) = \pi_1(h_1(n)),\ \pi_1(h(m)) = \pi_1(h_1(m))$$

$$m \in \pi_2(h(n)) \Leftrightarrow m \in \pi_2(h_1(n)),\ n \in \pi_2(h(m)) \Leftrightarrow n \in \pi_2(h_1(m))$$

Hence, it follows that $n, m \in dom(h)$ and

$$h(n) = (N, E'_n),\ h(m) = (M, E'_m),\ m \in E'_n, n \in E'_m$$

so $\langle break(N,M), h\rangle \rightsquigarrow \langle h'\rangle$ with

$$\langle h'\rangle = \langle h[n \mapsto (N, (E'_n - \{m\}) \cup \{\boxtimes\}), m \mapsto (N, (E'_m - \{n\}) \cup \{\boxtimes\})]\rangle.$$

Again, it is enough that we show $h' \in h_1 \circ h'_2$.

Remember that, since $h \in h_1 \circ h_2$, for all $k \in dom(h_i)$ $\;i = 1, 2$

$$\pi_1(h(k)) = \pi_1(h_i(k)) \;\; \forall k \in dom(h_i)$$

$$\exists X_k, X'_k.\; \pi_2(h(k)) = (\pi_2(h_i(k)) - X_k) \cup X'_k$$

We have:

- $h' \in \mathcal{H}$ because $h \in \mathcal{H}$ and $\langle break(N,M), h\rangle \rightsquigarrow \langle h'\rangle$, by Lemma 3.10.
- $dom(h') = dom(h_1) \cup dom(h'_2)$:

$$dom(h') = dom(h) = dom(h_1) \cup dom(h_2) = dom(h_1) \cup dom(h'_2)$$

- $\pi_1(h'(k)) = \pi_1(h_1(k))$ if $k \in dom(h_1)$ and $\pi_1(h'(k)) = \pi_1(h'_2(k))$ if $k \in dom(h'_2)$:

$$\pi_1(h'(k)) = \pi_1(h(k)) = \begin{cases} \pi_1(h_1(k)) & \forall k \in dom(h_1) \\ \pi_1(h_2(k)) = \pi_1(h'_2(k)) & \forall k \in dom(h_2) = dom(h'_2) \end{cases}$$

- $\pi_2(h'(k)) = \pi_2(h_1(k)) - X_k \cup X'_k$ for some X_k, X'_k with
  - $X_k \subseteq \mathcal{MP}(\{\square\})$
  - $X'_k \subseteq dom(h'_2)$

  if $k \in dom(h_1)$, and $\pi_2(h'(k)) = \pi_2(h'_2(k)) - X_k \cup X'_k$ for some X_k, X'_k with
  - $X_k \subseteq \mathcal{MP}(\{\square\})$
  - $X'_k \subseteq dom(h_1)$

  if $k \in dom(h'_2)$.

  This is true again, because:

$$\pi_2(h'(k)) \;=\; \pi_2(h(k)) = \begin{cases} (\pi_2(h_1(k)) - X_k) \cup X'_k \quad \forall k \in dom(h_1) \\ (\pi_2(h_2(k)) - X_k) \cup X'_k = (\pi_2(h'_2(k)) - X_k) \cup X'_k \\ \qquad\qquad\qquad\qquad \forall k \in dom(h'_2) - \{n, m\} \end{cases}$$

$$\begin{aligned} \pi_2(h'(n)) &= (\pi_2(h(n)) - \{m\}) \cup \{\boxtimes\} \\ &= (((\pi_2(h_2(n)) - X_n) \cup X'_n) - \{m\}) \cup \{\boxtimes\} \\ &\overset{*}{=} (((\pi_2(h_2(n)) - \{m\}) \cup \{\boxtimes\}) - X_n) \cup X'_n \\ &= (\pi_2(h'_2(n)) - X_n) \cup X'_n \end{aligned}$$

$$\begin{aligned} \pi_2(h'(m)) &= (\pi_2(h(m)) - \{n\}) \cup \{\boxtimes\} \\ &= (((\pi_2(h_2(m)) - X_m) \cup X'_m) - \{n\}) \cup \{\boxtimes\} \\ &\overset{**}{=} (((\pi_2(h_2(m)) - \{n\}) \cup \{\boxtimes\}) - X_m) \cup X'_m \\ &= (\pi_2(h'_2(m)) - X_m) \cup X'_m \end{aligned}$$

(*), (**): like in the previous case, $X_n,\; X'_n,\; \{m\},\; \{\boxtimes\}$ are all disjoint with each other, and so are $X_m,\; X'_m,\; \{n\},\; \{\boxtimes\}$.

So we can say that $h' \in h_1 \circ h'_2$.

– $C = C_1; C_2$

Because $\langle C_1; C_2, h_2\rangle$ is live, there is h'_2 with $\langle C_1; C_2, h_2\rangle \rightsquigarrow \langle h'_2\rangle$, which means that there is some h''_2 with $\langle C_1, h_2\rangle \rightsquigarrow \langle h''_2\rangle$ and $\langle C_2, h''_2\rangle \rightsquigarrow \langle h'_2\rangle$. This implies, in particular, that $\langle C_1, h_2\rangle$ is live, and so is $\langle C_2, h''_2\rangle$.

By induction hypothesis, since $\langle C_1, h_2\rangle$ is live, $h \in h_1 \circ h_2$ and we have h''_2 such that $\langle C_1, h_2\rangle \rightsquigarrow \langle h''_2\rangle$, we know that there is $h'' \in h_1 \circ h''_2$ such that $\langle C_1, h\rangle \rightsquigarrow \langle h''\rangle$.

On the other hand, since $\langle C_2, h''_2\rangle$ is live, $h'' \in h_1 \circ h''_2$ and $\langle C_2, h''_2\rangle \rightsquigarrow \langle h'_2\rangle$, by induction hypothesis, again, we know that there is $h' \in h_1 \circ h'_2$ such that $\langle C_2, h''\rangle \rightsquigarrow \langle h'\rangle$.

Having $\langle C_1, h\rangle \rightsquigarrow \langle h''\rangle$ and $\langle C_2, h''\rangle \rightsquigarrow \langle h'\rangle$ gives us $\langle C_1; C_2, h\rangle \rightsquigarrow \langle h'\rangle$ and, as we have seen, $h' \in h_1 \circ h'_2$.

– $C = C_1 + C_2$

If $\langle C_1 + C_2, h_2\rangle$ is live then there is some h'_2 such that $\langle C_1 + C_2, h_2\rangle \rightsquigarrow h'_2$, and this means that either $\langle C_1, h_2\rangle \rightsquigarrow h'_2$ or $\langle C_2, h_2\rangle \rightsquigarrow h'_2$. Without loss of generality, let us suppose that $\langle C_1, h_2\rangle \rightsquigarrow h'_2$. Then $\langle C_1, h_2\rangle$ is live and by induction hypothesis we have $h' \in h_1 \circ h'_2$ with $\langle C_1, h\rangle \rightsquigarrow \langle h'\rangle$. And this means that we also have $\langle C_1 + C_2, h\rangle \rightsquigarrow \langle h'\rangle$ ($h' \in h_1 \circ h'_2$).

□

Lemma 3.13 (Soundness of the Frame Rule)**.** *The Frame Rule*

$$\frac{\{A\}\ C\ \{A'\}}{\{A * D\}\ C\ \{A' * D\}}$$

is sound, i.e., if the premise holds then so does the conclusion.

Proof. To see that the conclusion holds, we assume that $h \models A * D$ for some $h \in \mathcal{H}$, i.e. that $h \in h_1 \circ h_2$ with $h_1 \models D$ and $h_2 \models A$. We note that since $h_2 \models A$ and $\{A\}\ C\ \{A'\}$ holds by assumption, then there exists some h'_2 with $\langle h_2, C\rangle \rightsquigarrow \langle h'_2\rangle$ and $h'_2 \models A'$.

We must prove that there is some h' with $\langle h, C\rangle \rightsquigarrow \langle h'\rangle$ and $h' \models A' * D$. Since we have $\langle h_2, C\rangle \rightsquigarrow \langle h'_2\rangle$, and $h \in h_1 \circ h_2$, we can apply the Nondeterministic Frame Property to obtain $h' \in h_1 \circ h'_2$ such that $\langle C, h\rangle \rightsquigarrow \langle h'\rangle$, and thus we have found h' with $\langle C, h\rangle \rightsquigarrow \langle h'\rangle$ and $h' \models A' * D$, as required. □

3.3.4 Tight Specifications.

With an appropriate syntax for assertions describing our resource, we can now give (tight) specifications for our basic commands, which will be the axioms in our set of rules. These axioms are given in Figure 10 together with the rules for sequential composition and nondeterministic choice.

Example 3.2. *Let us give a proof for a "program" that turns two methane molecules into one ethane molecule and one hydrogen molecule. The program would be the following:*

$$\mathit{break}(\mathtt{C},\mathtt{H}); \mathit{break}(\mathtt{C},\mathtt{H}); \mathit{bond}(\mathtt{C},\mathtt{C}); \mathit{bond}(\mathtt{H},\mathtt{H})$$

We start with two methane molecules, h_1 and h_2:

$$dom(h_1) = \{1, 2, 3, 4, 5\}$$
$$h_1(1) = (\mathtt{C}, \{2, 3, 4, 5\}) \quad h_1(2) = h_1(3) = h_1(4) = h_1(5) = (\mathtt{H}, \{1\})$$

$$\overline{\{\mathtt{H}_i\{j\} * \mathtt{H}_j\{i\}\} \;\; \mathit{break}(\mathtt{H},\mathtt{H}) \;\; \{\mathtt{H}_i\{\boxtimes\} * \mathtt{H}_j\{\boxtimes\}\}}$$

$$\overline{\{\mathtt{H}_i\{j\} * \mathtt{C}_j\{i, x_2, x_3, x_4\}\} \;\; \mathit{break}(\mathtt{C},\mathtt{H}) \;\; \{\mathtt{H}_i\{\boxtimes\} * \mathtt{C}_j\{\boxtimes, x_2, x_3, x_4\}\}}$$

$$\overline{\{\mathtt{C}_i\{j, x_2, x_3, x_4\} * \mathtt{C}_j\{i, x'_2, x'_3, x'_4\}\} \;\; \mathit{break}(\mathtt{C},\mathtt{C}) \;\; \{\mathtt{C}_i\{\boxtimes, x_2, x_3, x_4\} * \mathtt{C}_j\{\boxtimes, x'_2, x'_3, x'_4\}\}}$$

$$\overline{\{\mathtt{H}_i\{\boxtimes\} * \mathtt{H}_j\{\boxtimes\}\} \;\; \mathit{bond}(\mathtt{H},\mathtt{H}) \;\; \{\mathtt{H}_i\{j\} * \mathtt{H}_j\{i\}\}}$$

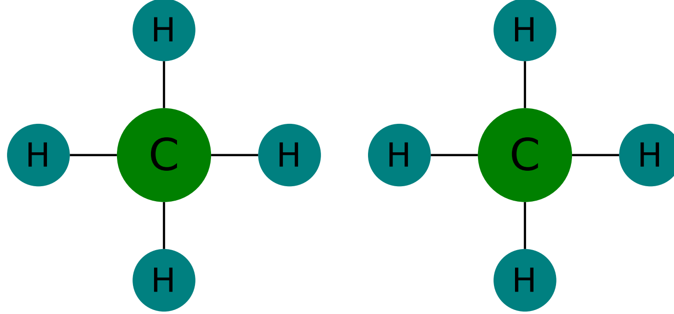
$$\overline{\{\mathtt{H}_i\{\boxtimes\} * \mathtt{C}_j\{\boxtimes, x_2, x_3, x_4\}\} \;\; \mathit{bond}(\mathtt{C},\mathtt{H}) \;\; \{\mathtt{H}_i\{j\} * \mathtt{C}_j\{i, x_2, x_3, x_4\}\}}$$

$$\overline{\{\mathtt{C}_i\{\boxtimes, x_2, x_3, x_4\} * \mathtt{C}_j\{\boxtimes, x'_2, x'_3, x'_4\}\} \;\; \mathit{bond}(\mathtt{C},\mathtt{C}) \;\; \{\mathtt{C}_i\{j, x_2, x_3, x_4\} * \mathtt{C}_j\{i, x'_2, x'_3, x'_4\}\}}$$

$$\frac{\{A\}\ C\ \{A''\} \quad \{A''\}\ C\ \{A'\}}{\{A\}\ C; C'\ \{A'\}} \qquad \frac{\{A\}\ C\ \{A'\} \quad \{A\}\ C'\ \{A'\}}{\{A\}\ C + C'\ \{A'\}}$$

Figure 10: Axioms and rules for the Partial Hydrocarbon model

$$dom(h_2) = \{6, 7, 8, 9, 10\}$$
$$h_2(6) = (\mathtt{C}, \{7, 8, 9, 10\}) \quad h_2(7) = h_2(8) = h_2(9) = h_2(10) = (\mathtt{H}, \{6\})$$



The picture represent as well the (unique) partial hydrocarbon in $h_1 \circ h_2$, which satisfies

$$\mathtt{C}_1\{2,3,4,5\} * \mathtt{H}_2\{1\} * \mathtt{H}_3\{1\} * \mathtt{H}_4\{1\} * \mathtt{H}_5\{1\}$$
$$* \mathtt{C}_6\{7,8,9,10\} * \mathtt{H}_7\{6\} * \mathtt{H}_8\{6\} * \mathtt{H}_9\{6\} * \mathtt{H}_{10}\{6\}$$

It is, in fact, the only partial hydrocarbon that satisfies this assertion.

Figure 11 will help see how this transformation takes place.

The proof that our program transforms (due to the nondeterminism in the commands it might be more accurate to say "is able to transform") two methane molecules into one ethane and one hydrogen molecule is in Figure 12.

We can observe that, indeed, only one partial hydrocarbon satisfies the last assertion

$$\{\mathtt{C}_1\{6,3,4,5\} * \mathtt{H}_2\{7\} * \mathtt{H}_3\{1\} * \mathtt{H}_4\{1\} * \mathtt{H}_5\{1\} * \mathtt{C}_6\{1,8,9,10\} * \mathtt{H}_7\{2\} * \mathtt{H}_8\{6\} * \mathtt{H}_9\{6\} * \mathtt{H}_{10}\{6\}\}$$

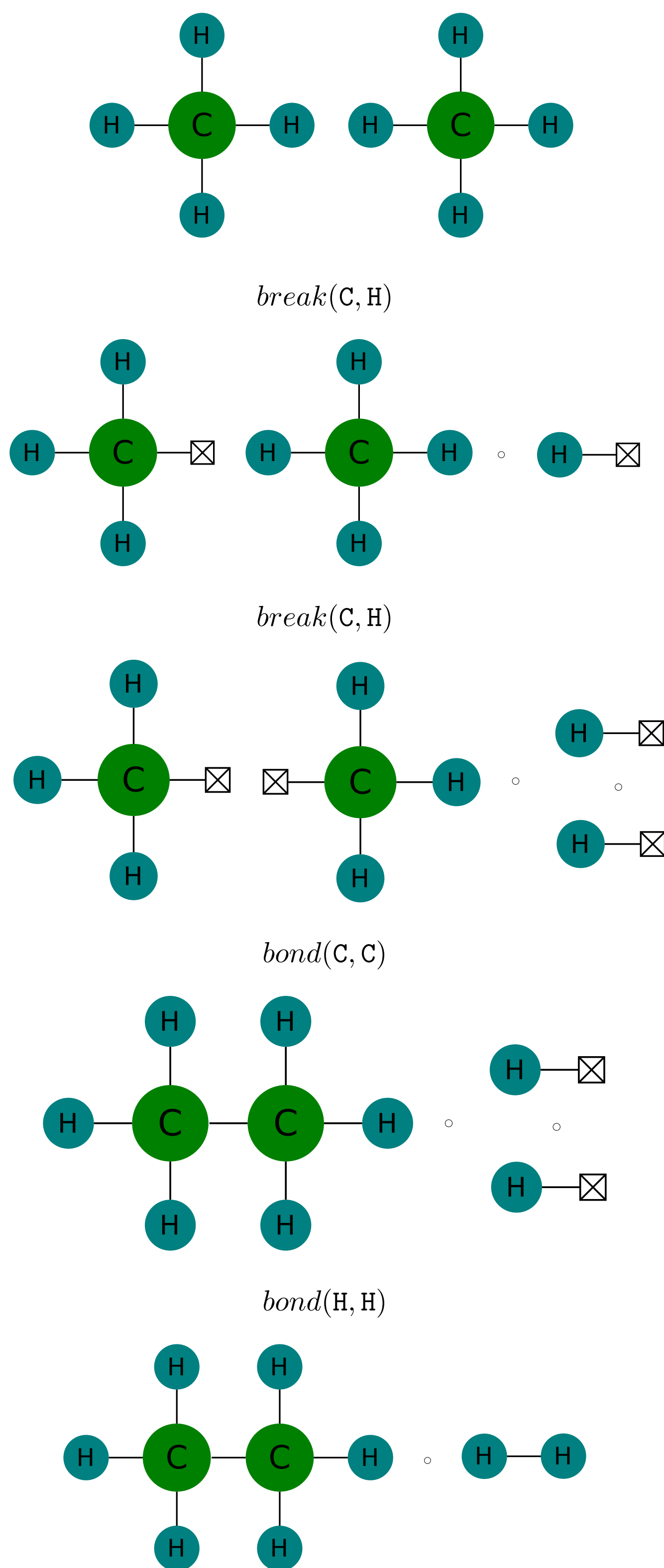


Figure 11: Two methane molecules turning into an ethane and a hydrogen molecule

$$\{\mathtt{C}_1\{2,3,4,5\} * \mathtt{H}_2\{1\} * \mathtt{H}_3\{1\} * \mathtt{H}_4\{1\} * \mathtt{H}_5\{1\}$$
$$*\mathtt{C}_6\{7,8,9,10\} * \mathtt{H}_7\{6\} * \mathtt{H}_8\{6\} * \mathtt{H}_9\{6\} * \mathtt{H}_{10}\{6\}\}$$

$$\mathit{break}(\mathtt{C},\mathtt{H})$$

$$\{\mathtt{C}_1\{\boxtimes,3,4,5\} * \mathtt{H}_2\{\boxtimes\} * \mathtt{H}_3\{1\} * \mathtt{H}_4\{1\} * \mathtt{H}_5\{1\}$$
$$*\mathtt{C}_6\{7,8,9,10\} * \mathtt{H}_7\{6\} * \mathtt{H}_8\{6\} * \mathtt{H}_9\{6\} * \mathtt{H}_{10}\{6\}\}$$

$$\mathit{break}(\mathtt{C},\mathtt{H})$$

$$\{\mathtt{C}_1\{\boxtimes,3,4,5\} * \mathtt{H}_2\{\boxtimes\} * \mathtt{H}_3\{1\} * \mathtt{H}_4\{1\} * \mathtt{H}_5\{1\}$$
$$*\mathtt{C}_6\{\boxtimes,8,9,10\} * \mathtt{H}_7\{\boxtimes\} * \mathtt{H}_8\{6\} * \mathtt{H}_9\{6\} * \mathtt{H}_{10}\{6\}\}$$

$$\mathit{bond}(\mathtt{C},\mathtt{C})$$

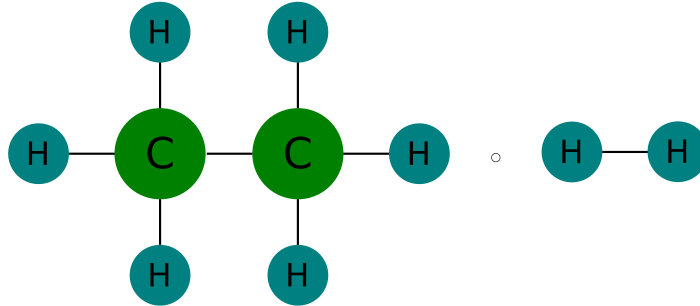
$$\{\mathtt{C}_1\{6,3,4,5\} * \mathtt{H}_2\{\boxtimes\} * \mathtt{H}_3\{1\} * \mathtt{H}_4\{1\} * \mathtt{H}_5\{1\}$$
$$*\mathtt{C}_6\{1,8,9,10\} * \mathtt{H}_7\{\boxtimes\} * \mathtt{H}_8\{6\} * \mathtt{H}_9\{6\} * \mathtt{H}_{10}\{6\}\}$$

$$\mathit{bond}(\mathtt{H},\mathtt{H})$$

$$\{\mathtt{C}_1\{6,3,4,5\} * \mathtt{H}_2\{7\} * \mathtt{H}_3\{1\} * \mathtt{H}_4\{1\} * \mathtt{H}_5\{1\}$$
$$*\mathtt{C}_6\{1,8,9,10\} * \mathtt{H}_7\{2\} * \mathtt{H}_8\{6\} * \mathtt{H}_9\{6\} * \mathtt{H}_{10}\{6\}\}$$

Figure 12: Proof of correctness of a simple program manipulating partial hydrocarbon

in Figure 12. It is the partial hydrocarbon that consists of one ethane and one hydrogen molecule:



Of course, to justify the use of the rules that have been given, it remains to prove their soundness.

Lemma 3.14. *The proof rules in Figure 10 are sound.*

Proof.

– $$\overline{\{\mathtt{H}_i\{j\} * \mathtt{H}_j\{i\}\}\ \mathit{break}(\mathtt{H},\mathtt{H})\ \{\mathtt{H}_i\{\boxtimes\} * \mathtt{H}_j\{\boxtimes\}\}}$$
Given h so that $h \models \mathtt{H}_i\{j\} * \mathtt{H}_j\{i\}$, we must have $\mathit{dom}(h) = \{i,j\}$ with $h(i) = (\mathtt{H},\{j\})$ and $h(j) = (\mathtt{H},\{i\})$. The operational semantics gives $\langle \mathit{break}(\mathtt{H},\mathtt{H}), h\rangle \rightsquigarrow \langle h'\rangle$ where

$$h' = h[i \mapsto (\mathtt{H},\{\boxtimes\}), j \mapsto (\mathtt{H},\{\boxtimes\})] = [i \mapsto (\mathtt{H},\{\boxtimes\}), j \mapsto (\mathtt{H},\{\boxtimes\})]$$

hence $h' \models \mathtt{H}_i\{\boxtimes\} * \mathtt{H}_j\{\boxtimes\}$.

– $\overline{\{\mathtt{H}_i\{j\} * \mathtt{C}_j\{i, x_2, x_3, x_4\}\}\ \mathit{break}(\mathtt{C}, \mathtt{H})\ \{\mathtt{H}_i\{\boxtimes\} * \mathtt{C}_j\{\boxtimes, x_2, x_3, x_4\}\}}$

Given h so that $h \models \mathtt{H}_i\{j\} * \mathtt{C}_j\{i, x_2, x_3, x_4\}$, we must have $\mathit{dom}(h) = \{i, j\}$ with $h(i) = (\mathtt{H}, \{j\})$ and $h(j) = (\mathtt{C}, \{i, x_2, x_3, x_4\})$. The operational semantics gives $\langle \mathit{break}(\mathtt{C}, \mathtt{H}), h\rangle \rightsquigarrow \langle h' \rangle$ where

$$h' = h[i \mapsto (\mathtt{H}, \{\boxtimes\}), j \mapsto (\mathtt{C}, \{\boxtimes, x_2, x_3, x_4\})] = [i \mapsto (\mathtt{H}, \{\boxtimes\}), j \mapsto (\mathtt{C}, \{\boxtimes, x_2, x_3, x_4\})]$$

hence $h' \models \mathtt{H}_i\{\boxtimes\} * \mathtt{C}_j\{\boxtimes, x_2, x_3, x_4\}$.

– $\overline{\{\mathtt{C}_i\{j, x_2, x_3, x_4\} * \mathtt{C}_j\{i, x'_2, x'_3, x'_4\}\}\ \mathit{break}(\mathtt{C}, \mathtt{C})\ \{\mathtt{C}_i\{\boxtimes, x_2, x_3, x_4\} * \mathtt{C}_j\{\boxtimes, x'_2, x'_3, x'_4\}\}}$

Given h so that $h \models \mathtt{C}_i\{j, x_2, x_3, x_4\} * \mathtt{C}_j\{i, x'_2, x'_3, x'_4\}$, we must have $\mathit{dom}(h) = \{i, j\}$ with $h(i) = (\mathtt{C}, \{j, x_2, x_3, x_4\})$ and $h(j) = (\mathtt{C}, \{i, x'_2, x'_3, x'_4\})$. The operational semantics gives $\langle \mathit{break}(\mathtt{C}, \mathtt{C}), h\rangle \rightsquigarrow \langle h' \rangle$ where

$$h' = h[i \mapsto (\mathtt{C}, \{\boxtimes, x_2, x_3, x_4\}), j \mapsto (\mathtt{C}, \{\boxtimes, x'_2, x'_3, x'_4\})] =$$
$$= [i \mapsto (\mathtt{H}, \{\boxtimes, x_2, x_3, x_4\}), j \mapsto (\mathtt{C}, \{\boxtimes, x'_2, x'_3, x'_4\})]$$

hence $h' \models \mathtt{C}_i\{\boxtimes, x_2, x_3, x_4\} * \mathtt{C}_j\{\boxtimes, x'_2, x'_3, x'_4\}$.

– $\overline{\{\mathtt{H}_i\{\boxtimes\} * \mathtt{H}_j\{\boxtimes\}\}\ \mathit{bond}(\mathtt{H}, \mathtt{H})\ \{\mathtt{H}_i\{j\} * \mathtt{H}_j\{i\}\}}$

Given h so that $h \models \mathtt{H}_i\{\boxtimes\} * \mathtt{H}_j\{\boxtimes\}$, we must have $\mathit{dom}(h) = \{i, j\}$ with $h(i) = (\mathtt{H}, \{\boxtimes\})$ and $h(j) = (\mathtt{H}, \{\boxtimes\})$. The operational semantics gives $\langle \mathit{bond}(\mathtt{H}, \mathtt{H}), h\rangle \rightsquigarrow \langle h' \rangle$ where

$$h' = h[i \mapsto (\mathtt{H}, \{j\}), i \mapsto (\mathtt{H}, \{i\})] = [i \mapsto (\mathtt{H}, \{j\}), j \mapsto (\mathtt{H}, \{i\})]$$

hence $h' \models \mathtt{H}_i\{j\} * \mathtt{H}_j\{i\}$.

– $\overline{\{\mathtt{H}_i\{\boxtimes\} * \mathtt{C}_j\{\boxtimes, x_2, x_3, x_4\}\}\ \mathit{bond}(\mathtt{C}, \mathtt{H})\ \{\mathtt{H}_i\{j\} * \mathtt{C}_j\{i, x_2, x_3, x_4\}\}}$

Given h so that $h \models \mathtt{H}_i\{\boxtimes\} * \mathtt{C}_j\{\boxtimes, x_2, x_3, x_4\}$, we must have $\mathit{dom}(h) = \{i, j\}$ with $h(i) = (\mathtt{H}, \{\boxtimes\})$ and $h(j) = (\mathtt{C}, \{\boxtimes, x_2, x_3, x_4\})$. The operational semantics gives $\langle \mathit{bond}(\mathtt{C}, \mathtt{H}), h\rangle \rightsquigarrow \langle h' \rangle$ where

$$h' = h[i \mapsto (\mathtt{H}, \{j\}), j \mapsto (\mathtt{C}, \{i, x_2, x_3, x_4\})] = [i \mapsto (\mathtt{H}, \{j\}), j \mapsto (\mathtt{C}, \{i, x_2, x_3, x_4\})]$$

hence $h' \models \mathtt{H}_i\{j\} * \mathtt{C}_j\{i, x_2, x_3, x_4\}$.

– $\overline{\{\mathtt{C}_i\{\boxtimes, x_2, x_3, x_4\} * \mathtt{C}_j\{\boxtimes, x'_2, x'_3, x'_4\}\}\ \mathit{bond}(\mathtt{C}, \mathtt{C})\ \{\mathtt{C}_i\{j, x_2, x_3, x_4\} * \mathtt{C}_j\{i, x'_2, x'_3, x'_4\}\}}$

Given h so that $h \models \mathtt{C}_i\{\boxtimes, x_2, x_3, x_4\} * \mathtt{C}_j\{\boxtimes, x'_2, x'_3, x'_4\}$, we must have $\mathit{dom}(h) = \{i, j\}$ with $h(i) = (\mathtt{C}, \{\boxtimes, x_2, x_3, x_4\})$ and $h(j) = (\mathtt{C}, \{\boxtimes, x'_2, x'_3, x'_4\})$. The operational semantics gives $\langle \mathit{bond}(\mathtt{C}, \mathtt{C}), h\rangle \rightsquigarrow \langle h' \rangle$ where

$$h' = h[i \mapsto (\mathtt{C}, \{j, x_2, x_3, x_4\}), j \mapsto (\mathtt{C}, \{i, x'_2, x'_3, x'_4\})] =$$
$$= [i \mapsto (\mathtt{C}, \{j, x_2, x_3, x_4\}), j \mapsto (\mathtt{C}, \{i, x'_2, x'_3, x'_4\})]$$

hence $h' \models \mathtt{C}_i\{j, x_2, x_3, x_4\} * \mathtt{C}_j\{i, x'_2, x'_3, x'_4\}$.

- $$\frac{\{A\}\ C\ \{A''\} \quad \{A''\}\ C\ \{A'\}}{\{A\}\ C;C'\ \{A'\}}$$

  Let $h \models A$ and suppose $\langle C;C',h\rangle \rightsquigarrow^* \langle h'\rangle$. Then according to the operational semantics there must be some h'' such that $\langle C,h\rangle \rightsquigarrow^* \langle h''\rangle$ and $\langle C',h''\rangle \rightsquigarrow^* \langle h'\rangle$. Because we have $\{A\}\ C\ \{A''\}$ it must be $h'' \models A''$, and because we have $\{A''\}\ C\ \{A'\}$ we must also have $h' \models A'$.

- $$\frac{\{A\}\ C\ \{A'\} \quad \{A\}\ C'\ \{A'\}}{\{A\}\ C+C'\ \{A'\}}$$

  Let $h \models A$ and suppose $\langle C;C',h\rangle \rightsquigarrow^* \langle h'\rangle$. Then according to the operational semantics either $\langle C,h\rangle \rightsquigarrow^* \langle h'\rangle$ or $\langle C',h\rangle \rightsquigarrow^* \langle h'\rangle$. Because we have $\{A\}\ C\ \{A'\}$ and $\{A\}\ C\ \{A'\}$, in either case we must also have $h' \models A'$.

$\square$

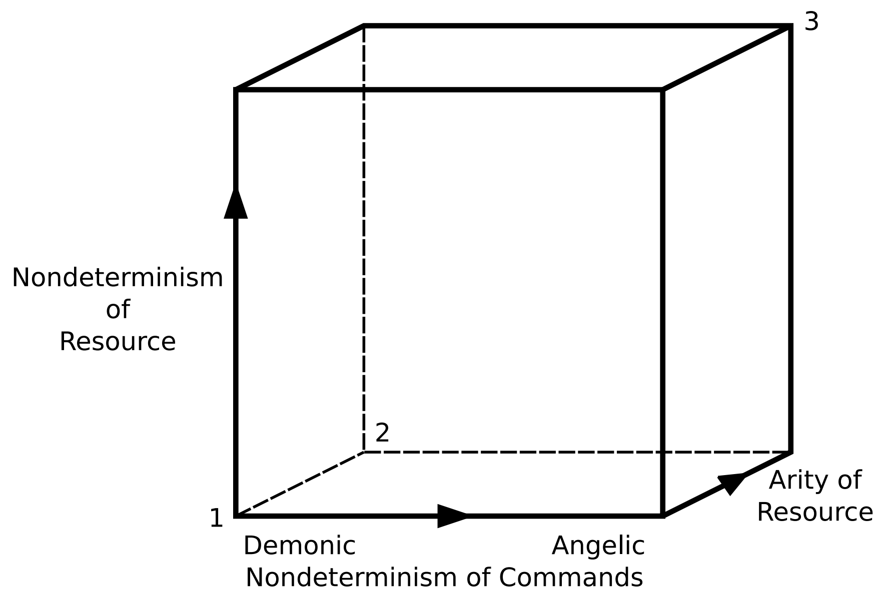
4 Conclusions and Future Work

The main contribution of this project has been to introduce the novel notion of arity of a resource instance, both with the bounded memory model and with the partial hydrocarbon model. We have made suitable modifications in the standard memory model (by introducing an additional component that refers to reserved inactive cells) and the corresponding set of axioms to get a model for finite memory allocation. We have proved that the rules — the modified axioms and the original rules — are sound in this setting, including the Frame Rule. This enables us to reason about programs that manipulate a finite shared memory heap, with memory allocation restricted to single cells. We have also suggested a way to improve the modifications in the model (turning the new third component into a set of natural numbers that describe reserved disjoint gaps in the memory) in such a way that we can handle allocation of several contiguous memory cells, and we have given a sensible operational semantics for memory allocation and deallocation in this setting. We have been quite cautious when giving this operational semantics in order to make it reasonable. To achieve this, we have sacrified a considerable amount of reserved gaps. This operational semantics could probably be improved, at least in the cases where the number of gaps reserved was less that 3, so that fewer reserved memory cells were sacrificed. It remains to prove soundness of the rest of the rules in this setting, in particular of the Frame Rule. Also, it would be interesting to see how the two successful aproaches can be extended to concurrent programs [10, 11].

In addition to extending the notion of arity to a resource different from a memory heap. with the partial hydrocarbon model we have looked into separation logic reasoning about a model where the commands have an angelic (vs. demonic) nondeterministic behaviour and the resource has a nondeterministic combination operation.

Our approach to the finite memory model is different to the one taken by Raza in [6]. Arguing that allocation and deallocation commands use, not only the information given explicitly in the states, but also some implicit information about the global allocation state of the whole memory, he chooses to make this information explicit by introducing a new basic construct for description of states. This new construct represents the whole set of unallocated addresses as a "unique, atomic piece of resource". In his approach, allocation/deallocation commands require ownership of the set of unallocated addresses — which must be non-empty in order not to lead to failure. He suggests a very useful analogy with the permissions model ([8]): in the permissions model, the right permission on a cell is needed to read or mutate its contents; in Raza's approach, permission over the whole set of inactive cells is needed to either allocate or dealllocate a cell. Our general approach to the finite memory model differs from this one in that we do not treat the set of non-allocated addresses as an atomic piece of resource, but split it into pieces at will, just like we do with the allocated parts of the memory. The permissions analogy, however, works in our approach too: we reserve (i.e. we have permission over) a certain number of cells (or some disjoint gaps in the memory, in another approach) for manipulation with our commands.

While exploring the models presented in previous sections, we have come accross three particular features that resource models can have: arity, deterministic vs. nondeterministic combination operation and deterministic/demonic nondeterministic vs. angelic nondeterministic commands. We find that it might be very interesting to look at various resources that present different combinations of these features, and even maybe present them in different degrees — if we found that there was a way of somehow measuring them. We can see this as exploring the following cube:



The standard memory model could be located in corner 1 of the cube, since it presents demonic nondeterminism in its commands, a (partial) deterministic combination operation and no notion of arity. The bounded heap model could be in corner 2, since it presents demonic nondeterminism in its commands and a deterministic combination operation, and we can talk about arity of its states. Finally, the partial hydrocarbon model could be located in corner 3, for it presents angelic nondeterminism in its commands, a nondeteministic combination operation, and we can talk about arity of its states as well.

References

[1] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *17th LICS*, pp 55-74, 2002.

[2] H. Yang and P. O'Hearn. A Semantic Basis for Local Reasoning. In *5th FOSSACS*, LNCS 2303, 2002.

[3] C. Calcagno, P. O'Hearn and H. Yang. Local Action and Abstract Separation Logic. *LICS'07*, 2007.

[4] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576-580 and 583, October 1969.

[5] C. A. R. Hoare. Proof of a program: FIND. *Communications of the ACM*, 14(1):39-45, January 1971.

[6] M. Raza. Resource Reasoning and Labelled Separation Logic, pages 51-55. *PhD thesis.* Department of Computing, Imperial College London, August 2010.

[7] H.Søndergaard and P. Sestof. Non-determinism in Functional Languages. *The Computer Journal*, 35(5):514-523, 1992.

[8] R. Bornat, C. Calcagno, P. OHearn and M. Parkinson. Permission Accounting in Separation Logic. In *32nd POPL*, pages 5970, 2005.

[9] D. Pym, P. OHearn and H. Yang. Possible worlds and Resources: The semantics of BI. *Theoretical Computer Science*, 315(1):257305, 2004.

[10] T. Dinsdale-Young, M. Dodds, M. Parkinson, P. Gardner, V. Vafeiadis, Concurrent abstract predicates. In *ECOOP, LNCS 6183*, pp. 504-528, 2010.

[11] P. O'Hearn. Resources, Concurrency, and Local Reasoning. *Theoretical Computer Science*, 375(1-3):271-307, 2007.

[12] S. Isthiaq and P. OHearn: BI as an Assertion Language for Mutable Data Structures. In *28th POPL*, 2001.

[13] J. Brotherston, A Unified Display Proof Theory for Bunched Logic. In Proceedings of MFPS-26, 2010.

[14] P. O'Hearn, J. Reynolds and H. Yang: Local Reasoning About Programs that Alter Data Structures. In *L. Fribourg, ed.: Computer Science Logic (CSL'01), Springer- Verlag*, 1-19 LNCS 2142, 2001.

[15] C.C. Morgan: The specification statement. In *ACM Transactions on Programming Languages and Systems*, 1988.

[16] W. Reisig, Distributed Algorithms: Modeling and Analising with Petri Nets, *Springer*, 1998.

[17] A. Armas, Extending Resource Reasoning with Nondeterminism, *Individual Study Option report*, Department of Computing, Imperial College London, April 2011.