Logical Methods in Computer Science
Vol. 4 (2:6) 2008, pp. 1–27
www.lmcs-online.org

Submitted Sep. 21, 2007
Published May 15, 2008

RELATIONAL PARAMETRICITY AND SEPARATION LOGIC

LARS BIRKEDAL [a] AND HONGSEOK YANG [b]

[a] IT University of Copenhagen, Denmark
e-mail address: birkedal@itu.dk

[b] Queen Mary, University of London, UK
e-mail address: hyang@dcs.qmul.ac.uk

Abstract. Separation logic is a recent extension of Hoare logic for reasoning about programs with references to shared mutable data structures. In this paper, we provide a new interpretation of the logic for a programming language with higher types. Our interpretation is based on Reynolds's relational parametricity, and it provides a formal connection between separation logic and data abstraction.

1. Introduction

Separation logic [18, 13, 7] is a Hoare-style program logic, and variants of it have been applied to prove correct interesting pointer algorithms such as copying a dag, disposing a graph, the Schorr-Waite graph algorithm, and Cheney's copying garbage collector. The main advantage of separation logic compared to ordinary Hoare logic is that it facilitates *local reasoning*, formalized via the so-called *frame rule* using a connective called *separating conjunction*. The development of separation logic initially focused on *low-level* languages with heaps and pointers, although in recent work [14, 8] it was shown how to extend separation logic first to languages with a simple kind of procedures [14] and then to languages also with higher-types [8]. Moreover, in [14] a second-order frame rule was proved sound and in [8] a whole range of higher-order frame rules were proved sound for a separation-logic type system.

In [14] and [8] it was explained how second and higher-order frame rules can be used to reason about static imperative modules. The idea is roughly as follows. Suppose that we prove a specification for a client c, depending on a module k,

$$\{P_1\}\, k\, \{Q_1\} \;\vdash\; \{P\}\, c(k)\, \{Q\}.$$

The proof of the client depends only on the "abstract specification" of the module, which describes the external behavior of k. Suppose further that an actual implementation m of the module satisfies

$$\{P_1 * I\}\, m\, \{Q_1 * I\}.$$

1998 ACM Subject Classification: F.3, D.3.
Key words and phrases: Program Verification, Separation Logic, Parametricity, Data Abstraction.

DOI:10.2168/LMCS-4 (2:6) 2008

$$
\begin{array}{lll}
\mathsf{init}_0(i) \equiv c.\mathsf{next} := i & \mathsf{inc}_0 \equiv \mathsf{let}\ i = c.\mathsf{next}\ \mathsf{in} & \mathsf{read}_0 \equiv \mathsf{let}\ i = c.\mathsf{next}\ \mathsf{in} \\
& \quad \mathsf{let}\ v = i.\mathsf{data}\ \mathsf{in} & \quad \mathsf{let}\ v = i.\mathsf{data}\ \mathsf{in} \\
& \quad i.\mathsf{data} := v{+}1 & \quad g.\mathsf{data} := v \\
\mathsf{init}_1(i) \equiv \mathsf{let}\ v = i.\mathsf{data}\ \mathsf{in} & \mathsf{inc}_1 \equiv \mathsf{let}\ i = c.\mathsf{next}\ \mathsf{in} & \mathsf{read}_1 \equiv \mathsf{let}\ i = c.\mathsf{next}\ \mathsf{in} \\
\quad i.\mathsf{data} := -v; & \quad \mathsf{let}\ v = i.\mathsf{data}\ \mathsf{in} & \quad \mathsf{let}\ v = i.\mathsf{data}\ \mathsf{in} \\
\quad c.\mathsf{next} := i & \quad i.\mathsf{data} := v{-}1 & \quad g.\mathsf{data} := -v
\end{array}
$$

Figure 1: Counter Modules

Here I is the internal resource invariant of the module m, describing the internal heap storage used by the module to implement the abstract specification. We can then employ a (higher-order) frame rule on the specification for the client to get

$$\{P_1 * I\}\, k\, \{Q_1 * I\} \ \vdash\ \{P * I\}\, c(k)\, \{Q * I\},$$

and combine it with the specification for m to obtain

$$\{P * I\}\, c(m)\, \{Q * I\}.$$

A key advantage of this approach to modularity is that it facilitates so-called "ownership transfer." For example, if the module is a queue, then the ownership of cells transfers from the client to module upon insertion into the queue. Moreover, the discipline allows clients to maintain pointers into cells that have changed ownership to the module. See [14] for examples and more explanations of these facts.

Note that the higher-order frame rules in essence provide implicit quantification over internal resource invariants. In [5] it is shown how one can employ a higher-order version of separation logic, with explicit quantification of assertion predicates to reason about dynamic modularity (where there can be several instances of the same abstract data type implemented by an imperative module), see also [15]. The idea is to existentially quantify over the internal resource invariants in a module, so that in the above example, c would depend on a specification for k of the form

$$\exists I.\ \{P_1 * I\}\, k\, \{Q_1 * I\}.$$

As emphasized in the papers mentioned above, note that, both in the case of implicit quantification over internal resource invariants (higher-order frame rules) and in the case of explicit quantification over internal resource invariants (existentials over assertion predicates), reasoning about a client does not depend on the internal resource invariant of possible module implementations. Thus the methodology allows us to formally reason about *mutable abstract data types*, aka. *imperative modules*.

However, the semantic models in the papers mentioned above do not allow us to make all the conclusions we would expect from reasoning about mutable abstract data types. In particular, we would expect that *clients should behave parametrically in the internal resource invariants*: When a client is applied to two different implementations of a mutable abstract data type, it should be the case that the client preserves relations between the internal resource invariants of the two implementations. This is analogous to Reynolds's style relational parametricity for abstract data types with quantification over type variables [17].

To understand this issue more clearly, consider the two implementations of a counter in Figure 1. A counter has three operations: $\mathsf{init}(i)$ for initializing the counter, and inc and read for increasing and reading the value of the counter. In the first implementation, $\mathsf{init}_0(i)$ takes a heap cell i containing an initial value for the counter, and stores its address i in the internal variable c, thereby setting the value of the counter to the contents of i. The intention is that when a client program calls this initialization routine with cell i, it should transfer the ownership of the cell to the counter – it should not dereference the cell after calling $\mathsf{init}_0(i)$. The operation inc_0 increases the value of the transferred cell i, and read_0 returns the value of cell i, by storing it in a pre-determined global variable g. The second implementation is almost identical to the first, except that the value of the counter is negated. Thus, when R is the relation that relates a heap containing cell i and variable c with the same heap with the value of cell i negated, all operations of these two implementations preserve this relation R.

Now suppose that we are given a client program of the form

$$\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\big(i.\mathsf{data}{:=}n;\ \mathsf{init}(i);\ b(\mathsf{inc},\mathsf{read})\big)$$

whose body b satisfies the following specification in separation logic:

$$\{\mathsf{emp}\}\,\mathsf{inc}\,\{\mathsf{emp}\},\quad \{g\mapsto \text{-}\}\,\mathsf{read}\,\{g\mapsto \text{-}\}\ \vdash\ \{P\}\,b(\mathsf{inc},\mathsf{read})\,\{Q\}$$

for some P, Q that do not mention cell i. We expect that the body b of the client preserves the relation R of the two implementations, and that the client cannot detect the difference between the two. Our expectation is based on the specification for b, which says that the triple $\{P\}\,b(\mathsf{inc},\mathsf{read})\,\{Q\}$ can be proved in separation logic, assuming only the "abstract specification" of the inc and read operations, where all the internal resources of the module, such as cell i, are hidden. This provability should prevent b from accessing the internal resources of a counter directly and thus detecting the difference between the two implementations. However, none of the existing models of separation logic can justify our expectation on the client program above.

In this paper we provide a new parametric model of separation logic, which captures that clients behave parametrically in internal resource invariants of mutable abstract data types. For instance, our model shows that $b(\mathsf{inc},\mathsf{read})$ preserves the relation R, and thus it behaves in the same way no matter whether we use the first or second implementation of a counter. In the present paper, we will focus on the implicit approach to quantification over internal resource invariants via higher-order frame rules, since it is technically simpler than the explicit approach.[1]

Our new model of separation logic is based on two novel ideas. The first is to read specifications in separation logic as relations between two programs. For instance, in our model, the Hoare triple $\{P\}\,b(\mathsf{inc},\mathsf{read})\,\{Q\}$ describes a relationship between two instantiations $[\![b(\mathsf{inc},\mathsf{read})]\!]_{\eta_0}$ and $[\![b(\mathsf{inc},\mathsf{read})]\!]_{\eta_1}$ of the client's body b by environments η_0 and η_1. Intuitively, environment η_i defines an implementation of module operations inc and read, so $[\![b(\mathsf{inc},\mathsf{read})]\!]_{\eta_i}$ means b is linked with the implementation $\eta_i(\mathsf{inc})$ and $\eta_i(\mathsf{read})$. Note that when used with appropriate η_0, η_1 (i.e., η_i that maps inc and read to the meaning of inc_i and read_i), the triple expresses how $b(\mathsf{inc}_0,\mathsf{read}_0)$ is related to $b(\mathsf{inc}_1,\mathsf{read}_1)$.

[1]The reason is that the implicit quantification of separation logic uses quantification in a very disciplined way so that the usual reading of assertions as sets of heaps can be maintained; if we use quantification without any restrictions, as in [3], it appears that we cannot have the usual reading of assertions as sets of heaps because, then, the rule of consequence is not sound.

The second idea is to parameterize the interpretation by relations on heaps. Mathematically, this means that the interpretation uses a Kripke structure that consists of relations on heaps. The relation parameter describes how the internal resource invariants of two modules are related, and it lets us express the preservation of this relation by client programs. In our counter example, an appropriate parameter is the relation R above. When the triple $\{P\}\, b(\mathsf{inc}, \mathsf{read})\, \{Q\}$ is interpreted with R (and η_i corresponding to $\mathsf{inc}_i, \mathsf{read}_i$), it says, in particular, that $b(\mathsf{inc}_0, \mathsf{read}_0)$ and $b(\mathsf{inc}_1, \mathsf{read}_1)$ should preserve the relation R between the internal resources of the two implementations of a counter.

1.1. **Related Work.** Technically, it has proven to be a very non-trivial problem to define a parametric model for separation logic. One of the main technical challenges in developing a relationally parametric model of separation logic, even for a simple first-order language, is that the standard models of separation logic allow the identity of locations to be observed in the model. This means in particular that allocation of new heap cells is not parametric because the identity of the location of the allocated cell can be observed in the model. (We made this observation in earlier unpublished joint work with Noah Torp-Smith, see [20, Ch. 6].)

This problem of non-parametric memory allocation has also been noticed by recent work on data refinement for heap storage, which exploits semantic ideas from separation logic [10, 11]. However, the work on data refinement does not provide a satisfactory solution. Either it avoids the problem by assuming that clients do not allocate cells [10], or its solution has difficulties for handling higher-order procedures and formalizing (observational) equivalences, not refinements, between two implementations of a mutable abstract data type [11].

Our solution to this challenge is to define a more refined semantics of the programming language using FM domain theory, in the style of Benton and Leperchey [4], in which one can name locations but not observe the identity of locations because of the built-in use of permutation of locations. Part of the trick of *loc. cit.* is to define the semantics in a continuation-passing style so that one can ensure that new locations are suitably fresh with respect to the remainder of the computation. (See Section 4 for more details.) Benton and Leperchey used the FM domain-theoretic model to reason about contextual equivalence and here we extend the approach to give a semantics of separation logic in a continuation-passing style. We relate this new interpretation to the standard direct-style interpretation of separation logic via the so-called observation closure $(-)^{\perp\!\perp}$ of a relation, see Section 7.

The other main technical challenge in developing a relationally parametric model of separation logic for reasoning about mutable abstract data types is to devise a model which validates a wide range of higher-order frame rules. Our solution to this challenge is to define an intuitionistic interpretation of the specification logic over a Kripke structure, whose ordering relation intuitively captures the framing-in of resources. Technically, the intuitionistic interpretation, in particular the associated Kripke monotonicity, is used to validate a generalized frame rule. Further, to show that the semantics of the logic does indeed satisfy Kripke monotonicity for the base case of triples, we interpret triples using a universal quantifier, which intuitively quantifies over resources that can possibly be framed in. In the earlier non-parametric model of higher-order frame rules for separation-logic typing in [8] we also made use of a Kripke structure. The difference is that in the present work the elements of the Kripke structure are *relations* on heaps rather than predicates on heaps because we build a *relationally* parametric model.

In earlier work, Banerjee and Naumann [1] studied relational parametricity for dynamically allocated heap objects in a Java-like language. Banerjee and Naumann made use of a non-trivial semantic notion of confinement to describe internal resources of a module; here instead we use separation logic, in particular separating conjunction and frame rules, to describe which resources are internal to the module. Our model directly captures that whenever a client has been proved correct in separation logic with respect to an abstract view of a module, then it does not matter how the module has been implemented internally. And, this holds for a higher-order language with higher-order frame rules.

This paper is organized as follows. In Section 2 we describe the programming and assertion languages we consider and in Section 3 we define our version of separation logic. In Section 4 we define the semantics of our programming language in the category of FM-cpos, and describe our relational interpretation of separation logic in Section 5. In Section 6 we present a general abstract construction that provides models of specification logic with higher-order frame rules and show that the semantics of the previous section is in fact a special case of the general construction. Section 7 relates our relational interpretation to the standard interpretation of separation logic, and in Section 8 we present the abstraction theorem that our parametric model validates. We describe examples in Section 9, and finally we conclude and discuss future work in Section 10.

An extended abstract of this paper was presented at the FOSSACS 2007 conference [9]. This paper includes proofs that were missing in the conference version, and describes a general mathematical construction that lies behind our parametric model of separation logic. We also include a new example that illustrates the subtleties of the problems and results.

2. Programs and Assertions

In this paper, we consider a higher-order language with immutable stack variables. The types and terms of the language are defined as follows:

$$
\begin{array}{rrcl}
\text{Types} & \tau & ::= & \mathsf{com} \mid \mathsf{val} \to \tau \mid \tau \to \tau \\
\text{Expressions} & E & ::= & i \mid 0 \mid 1 \mid -1 \mid E + E \mid E - E \\
\text{Terms} & M & ::= & x \mid \lambda i.\, M \mid M\ E \mid \lambda x\colon \tau.\, M \mid M\, M \\
 & & \mid & \mathsf{fix}\, M \mid \mathsf{if}\ (E{=}E)\ \mathsf{then}\ M\ \mathsf{else}\ M \mid M; M \\
 & & \mid & \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \mid \mathsf{free}(E) \mid \mathsf{let}\ i{=}E.f\ \mathsf{in}\ M \mid E.f{:=}E \quad (f \in \{0,1\})
\end{array}
$$

The language separates expressions E from terms M. Expressions denote heap-independent values, which are either the address of a heap cell or an integer. Expressions are bound to *stack variables* i, j. On the other hand, terms denote possibly heap-dependent computations, and they are bound to *identifiers* x, y. The syntax of the language ensures that expressions always terminate, while terms can diverge. The types are used to classify terms only. com denotes commands, $\mathsf{val} \to \tau$ means functions that take an expression parameter, and $\tau \to \tau'$ denotes functions that takes a term parameter. Note that to support two different function types, the language includes two kinds of abstraction and application, one for expression parameters and the other for term parameters. We assume that term parameters are passed by name, and expression parameters are passed by value.

To simplify the presentation, we take a simple storage model where each heap cell has only two fields 0 and 1. Command $\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M$ allocates such a binary heap cell, binds the address of the cell to i, and runs M under this binding. The f'th field of this newly

$$\frac{}{\Delta, i \vdash i} \qquad \frac{}{\Delta \vdash 0} \qquad \frac{\Delta \vdash E_1 \quad \Delta \vdash E_2}{\Delta \vdash E_1 + E_2} \qquad \frac{\Delta \vdash E_1 \quad \Delta \vdash E_2}{\Delta \vdash E_1 - E_2}$$

$$\frac{}{\Delta \mid \Gamma, x : \tau \vdash x : \tau} \quad \frac{\Delta, i \mid \Gamma \vdash M : \tau}{\Delta \mid \Gamma \vdash \lambda i.\, M : \mathsf{val} \to \tau} \quad \frac{\Delta \mid \Gamma \vdash M : \mathsf{val} \to \tau \quad \Delta \vdash E}{\Delta \mid \Gamma \vdash M\ E : \tau}$$

$$\frac{\Delta \mid \Gamma, x : \tau \vdash M : \tau'}{\Delta \mid \Gamma \vdash \lambda x : \tau.\, M : \tau \to \tau'} \quad \frac{\Delta \mid \Gamma \vdash M : \tau' \to \tau \quad \Delta \mid \Gamma \vdash N : \tau'}{\Delta \mid \Gamma \vdash M\ N : \tau} \quad \frac{\Delta \mid \Gamma \vdash M : \tau \to \tau}{\Delta \mid \Gamma \vdash \mathsf{fix}\ M : \tau}$$

$$\frac{\Delta \vdash E \quad \Delta \vdash F \quad \Delta \mid \Gamma \vdash M : \mathsf{com} \quad \Delta \mid \Gamma \vdash N : \mathsf{com}}{\Delta \mid \Gamma \vdash \mathsf{if}\ (E{=}F)\ \mathsf{then}\ M\ \mathsf{else}\ N : \mathsf{com}}$$

$$\frac{\Delta \mid \Gamma \vdash M : \mathsf{com} \quad \Delta \mid \Gamma \vdash N : \mathsf{com}}{\Delta \mid \Gamma \vdash M; N : \mathsf{com}} \quad \frac{\Delta, i \mid \Gamma \vdash M : \mathsf{com}}{\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M : \mathsf{com}} \quad \frac{\Delta \vdash E}{\Delta \mid \Gamma \vdash \mathsf{free}(E) : \mathsf{com}}$$

$$\frac{\Delta, i \mid \Gamma \vdash M : \mathsf{com} \quad \Delta \vdash E}{\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}E.f\ \mathsf{in}\ M : \mathsf{com}}\ f \in \{0,1\} \quad \frac{\Delta \vdash E \quad \Delta \vdash F}{\Delta \mid \Gamma \vdash E.f := F : \mathsf{com}}\ f \in \{0,1\}$$

Figure 2: Typing Rules for Expressions and Terms

allocated cell at address i is read by $\mathsf{let}\ j = i.f\ \mathsf{in}\ N$ and updated by $i.f := E$. The cell i is deallocated by $\mathsf{free}(i)$.

The language uses typing judgments of the form $\Delta \vdash E(:\mathsf{val})$ and $\Delta \mid \Gamma \vdash M : \tau$, where Δ is a finite set of stack variables and Γ is a standard type environment for identifiers x. The typing rules for expressions and terms are shown in Figure 2.

We use the standard assertions from separation logic to describe properties of the heap:[2]

$$P \ ::= \ E = E \mid E \leq E \mid E \mapsto E, E \mid \mathsf{emp} \mid P * P \mid P \wedge P \mid \neg P \mid \exists i.\, P.$$

The points-to predicate $E \mapsto E_0, E_1$ means that the current heap has only one cell at address E and that the i-th field of the cell has the value E_i. The emp predicate denotes the empty heap, and the separating conjunction $P * Q$ means that the current heap can be split into two parts so that P holds for the one and Q holds for the other. The other connectives have the usual meaning from classical logic. All the missing connectives from classical logic are defined as usual.

In the paper, we will use the three abbreviations $(E \mapsto \text{-})$, $(E \mapsto \text{-}, E_1)$ and $(E \mapsto E_0, \text{-})$. The first $E \mapsto \text{-}$ is a syntactic sugar for $\exists i, j.\, E \mapsto i, j$, and denotes heaps with cell E only. $E \mapsto \text{-}, E_1$ is an abbreviation for $\exists i.\, E \mapsto i, E_1$, and means heaps that contain only cell E and store E' in the second field of this unique cell E. The last $E \mapsto E_0, \text{-}$ is defined similarly.

Assertions only depend on stack variables i, j, not identifiers x, y. Thus assertions are typed by a judgment $\Delta \vdash P : \mathsf{Assertion}$. The typing rules for this judgment are completely standard, and thus omitted from this paper.

[2]We omit separating implication $-\!\!*$ to simplify presentation.

3. Separation Logic

Our version of separation logic is the first-order *intuitionistic* logic extended with Hoare triples and invariant extension. The formulas in the logic are called *specifications*, and they are defined by the following grammar:

$$\begin{array}{rcl} \varphi & ::= & \{P\}M\{Q\} \mid \varphi \otimes P \mid E = E \mid M = M \\ & \mid & \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \forall x\colon \tau.\varphi \mid \exists x\colon \tau.\varphi \mid \forall i.\varphi \mid \exists i.\varphi \end{array}$$

The formula $\varphi \otimes P$ means the extension of φ by the invariant P. It can be viewed as a syntactic transformation of φ that inserts $P*-$ into the pre and post conditions of all triples in φ. For instance, $(\{P\}x\{Q\} \Rightarrow \{P'\}M(x)\{Q'\})\otimes P_0$ is equivalent to $\{P * P_0\}x\{Q * P_0\} \Rightarrow \{P' * P_0\}M(x)\{Q' * P_0\}$. We write Specs for the set of all specifications.

Specifications are typed by the judgment $\Delta \mid \Gamma \vdash \varphi : \mathsf{Specs}$, where we overloaded Specs to mean the type for specifications.

The logic includes all the usual proof rules from first-order intuitionistic logic with equality, and a rule for fixed-point induction. In addition, it contains proof rules from separation logic, and *higher-order frame rules*, expressed in terms of rules for invariant introduction and distribution. Figure 3 shows some of these additional rules and a rule for fixed-point induction. In the figure, we often omit contexts $\Delta \mid \Gamma$ for specifications and also conditions about typing.

The rules for Hoare triples are the standard proof rules of separation logic adapted to our language. Note that in the rule of consequence, we use the standard semantics of assertions P, P', Q, Q', in order to express semantic implications between those assertions (of course, standard logical derivability $\Delta \mid P \vdash P'$ and $\Delta \mid Q' \vdash Q$ are sufficient conditions). The rules for invariant extension formalize higher-order frame rules, extending the idea in [8]. The generalized higher-order frame rule $\varphi \Rightarrow \varphi \otimes P$ adds an invariant P to specification φ, and the other rules distribute this added invariant all the way down to the triples. We just show one use of those rules that lead to the second-order frame rule:

$$\dfrac{\dfrac{\dfrac{\Delta \mid \Gamma, x\colon \mathsf{com} \vdash \{P\}x\{Q\} \Rightarrow \{P'\}M(x)\{Q'\}}{\Delta \mid \Gamma, x\colon \mathsf{com} \vdash (\{P\}x\{Q\} \Rightarrow \{P'\}M(x)\{Q'\}) \otimes P_0}}{\Delta \mid \Gamma, x\colon \mathsf{com} \vdash \{P\}x\{Q\} \otimes P_0 \Rightarrow \{P'\}M(x)\{Q'\} \otimes P_0}}{\Delta \mid \Gamma, x\colon \mathsf{com} \vdash \{P * P_0\}x\{Q * P_0\} \Rightarrow \{P' * P_0\}M(x)\{Q' * P_0\}}$$

The last rule is for fixed-point induction, and it relies on the restriction that a specification is of the form $\gamma(\mathsf{fix}\ M)$. The grammar for γ guarantees that $\gamma(x)$ defines an admissible predicate for x, thus ensuring the soundness of fixed-point induction. Moreover, it also guarantees that $\gamma(x)$ holds when M means $\bot$, so allowing us to omit a usual base case, "$\gamma(\bot)$," from the rule.

Note that the rules do *not* include the so-called conjunction rule:

$$(\{P\}M\{Q\} \wedge \{P'\}M\{Q'\}) \ \Rightarrow\ \{P \wedge P'\}M\{Q \wedge Q'\}$$

The omission of this rule is crucial, since our parametricity interpretation does not validate the rule. We discuss the conjunction rule further in Section 10.

Example 3.1. Recall the counter example from the introduction and consider the following simple client

$$\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ \big(i.0 := 5; \mathsf{init}(i); \mathsf{inc}; \mathsf{read}\big),$$

PROOF RULES FOR HOARE TRIPLES

$$
\begin{array}{rcl}
(\forall i.\{P\}M\{Q\}) & \Rightarrow & \{\exists i.\, P\}M\{\exists i.\, Q\} \quad (\text{where } i \notin \mathsf{fv}(M)) \\
(\{P\}M\{Q\} \wedge \{P'\}M\{Q'\}) & \Rightarrow & \{P \vee P'\}M\{Q \vee Q'\} \\
\{P \wedge E{=}F\}M\{Q\} \wedge \{P \wedge E{\neq}F\}N\{Q\} & \Rightarrow & \{P\}\mathsf{if}\ (E{=}F)\ \mathsf{then}\ M\ \mathsf{else}\ N\{Q\} \\
\{P\}M\{P_0\} \wedge \{P_0\}N\{Q\} & \Rightarrow & \{P\}M; N\{Q\} \\
(\forall i.\, \{P * i \mapsto 0, 0\}M\{Q\}) & \Rightarrow & \{P\}\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M\{Q\}\ (\text{where } i{\notin}\mathsf{fv}(P,Q)) \\
(\forall i.\, \{P * E \mapsto i, E_1\}M\{Q\}) & \Rightarrow & \{\exists i.\, P * E \mapsto i, E_1\}\mathsf{let}\ i{=}E.0\ \mathsf{in}\ M\{Q\} \\
 & & (\text{where } i{\notin}\mathsf{fv}(E,Q))
\end{array}
$$

$$\{E \mapsto \text{-}\}\mathsf{free}(E)\{\mathsf{emp}\}$$

$$\{E \mapsto \text{-}, E_1\}E.0 := F\{E \mapsto F, E_1\}$$

$$\frac{[\![P]\!]_\rho \subseteq [\![P']\!]_\rho \text{ and } [\![Q']\!]_\rho \subseteq [\![Q]\!]_\rho \text{ for all } \rho \in [\![\Delta]\!]}{\Delta \mid \Gamma \vdash \{P'\}M\{Q'\} \Rightarrow \{P\}M\{Q\}}$$

PROOF RULES FOR INVARIANT EXTENSION $- \otimes P$

$$
\begin{array}{rcl@{\qquad}rcl}
\varphi & \Rightarrow & \varphi \otimes P & \{P\}M\{P'\} \otimes Q & \Leftrightarrow & \{P * Q\}M\{P' * Q\} \\
(E = F) \otimes Q & \Leftrightarrow & E = F & (M = N) \otimes Q & \Leftrightarrow & (M = N) \\
(\varphi \otimes P) \otimes Q & \Leftrightarrow & \varphi \otimes (P * Q) & (\varphi \oplus \psi) \otimes P & \Leftrightarrow & (\varphi \otimes P) \oplus (\psi \otimes P) \\
 & & & & & (\text{where } \oplus \in \{\Rightarrow, \wedge, \vee\}) \\
(\kappa x\colon \tau.\, \varphi) \otimes P & \Leftrightarrow & \kappa x\colon \tau.\, \varphi \otimes P & (\kappa i.\, \varphi) \otimes P & \Leftrightarrow & \kappa i.\, \varphi \otimes P \\
 & & (\text{where } \kappa \in \{\forall, \exists\}) & & & (\text{where } \kappa \in \{\forall, \exists\} \text{ and } i \notin \mathsf{fv}(P))
\end{array}
$$

RULE FOR FIXED-POINT INDUCTION

$$C ::= [\,] \mid \lambda i.C \mid C\,E \mid \lambda x\colon \tau.C \mid C\,M \mid \mathsf{fix}\,C \mid C; M \qquad \gamma ::= \{P\}C\{Q\} \mid \gamma{\wedge}\gamma \mid \forall x\colon \tau.\gamma \mid \forall i.\gamma$$

$$(\forall x.\, \gamma(x) \Rightarrow \gamma(M\ x)) \ \Rightarrow\ \gamma(\mathsf{fix}\ M)$$

where $\gamma(N)$ is a capture-avoiding insertion of N into the hole $[-]$ in γ.

Figure 3: Sample Proof Rules

whose body consists of inc; read. The client initializes the value of the counter to 5, increases the counter, and reads the value of the counter.

In our logic, we can prove that the body of the client satisfies:

$$\Delta \mid \Gamma \ \vdash\ \varphi \ \Rightarrow\ \{g \mapsto \text{-}\}\mathsf{inc}; \mathsf{read}\{g \mapsto \text{-}\}$$

where Δ is a set of stack variables containing g and Γ, φ are defined by

$$\Gamma \stackrel{def}{=} \{\mathsf{inc} : \mathsf{com}, \mathsf{read} : \mathsf{com}\}, \quad \varphi \stackrel{def}{=} \{\mathsf{emp}\}\mathsf{inc}\{\mathsf{emp}\} \wedge \{g \mapsto \text{-}\}\mathsf{read}\{g \mapsto \text{-}\}.$$

Note that cell i, which is transferred to the counter by $\mathsf{init}(i)$, does not appear in any assertion of the specification for the client's body. This implies, correctly, that the client does not dereference the transferred cell i, after calling $\mathsf{init}(i)$.

The formal proof of the specification of the body uses the first-order frame rule, and it is given below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta \mid \Gamma \vdash \varphi \Rightarrow \{\mathsf{emp}\}\mathsf{inc}\{\mathsf{emp}\}}\,1}{\Delta \mid \Gamma \vdash \varphi \Rightarrow (\{\mathsf{emp}\}\mathsf{inc}\{\mathsf{emp}\} \otimes (g \mapsto \text{-}))}\,2}{\Delta \mid \Gamma \vdash \varphi \Rightarrow \{\mathsf{emp} * g \mapsto \text{-}\}\mathsf{inc}\{\mathsf{emp} * g \mapsto \text{-}\}}\,3}{\Delta \mid \Gamma \vdash \varphi \Rightarrow \{g \mapsto \text{-}\}\mathsf{inc}\{g \mapsto \text{-}\}}\,4 \qquad \dfrac{}{\Delta \mid \Gamma \vdash \varphi \Rightarrow \{g \mapsto \text{-}\}\mathsf{read}\{g \mapsto \text{-}\}}\,5}{\Delta \mid \Gamma \vdash \varphi \Rightarrow \{g \mapsto \text{-}\}\mathsf{inc};\mathsf{read}\{g \mapsto \text{-}\}}\,6$$

The interesting parts of the proof are steps $2, 3$, where we use rules for invariant extensions, in order to add the frame axiom $g \mapsto$ - into the pre and post conditions of a triple. Note that the addition of this frame axiom starts with a generalized frame rule $\varphi \Rightarrow \varphi \otimes P$, and continues with the rule that moves P inside φ. The remaining steps $1, 5, 4, 6$ are instances of usual rules for first-order intuitionistic logic or Hoare logic, such as the elimination rule for conjunction and the rule of Consequence. □

4. Semantics of Programming Language

Let *Loc* be a countably infinite set of locations. The programming language is interpreted in the category of FM-cpos on *Loc*.

We remind the reader of the basics of FM domain theory. Call a bijection π on *Loc* a *permutation* when $\pi(l) \neq l$ only for finitely many l, and let perm be the set of all permutations. An FM-set is a pair of a set A and a function $\cdot$ of type $\mathsf{perm} \times A \to A$, such that (1) $\mathsf{id} \cdot a = a$ and $\pi \cdot (\pi' \cdot a) = (\pi \circ \pi') \cdot a$, and (2) every $a \in A$ is *supported* by some finite subset L of *Loc*, i.e.,

$$\forall \pi \in \mathsf{perm}.\ (\forall l \in L.\ \pi(l) = l) \implies \pi \cdot a = a.$$

It is known that every element a in an FM-set A has a smallest set L that supports a. This smallest set is denoted $\mathsf{supp}(a)$. An FM function f from an FM-set A to an FM-set B is a function from A to B such that $f(\pi \cdot a) = \pi \cdot (f(a))$ for all a, π.

An FM-poset is an FM-set A with a partial order $\sqsubseteq$ on A such that $a \sqsubseteq b \implies \pi \cdot a \sqsubseteq \pi \cdot b$ for all π, a, b. We say that a (ω-)chain $\{a_i\}_i$ in FM-poset A is *finitely supported* iff there is a finite subset L of *Loc* that supports all elements in the chain. Finally, an FM-cpo is an FM-poset $(A, \sqsubseteq)$ for which every finitely-supported chain $\{a_i\}_i$ has a least upper bound, and an FM continuous function f from an FM-cpo A to an FM-cpo B is an FM function from A to B that preserves the least upper bounds of all finitely supported chains.

Types are interpreted as pointed FM-cpos, using the categorical structure of the category of FM-cpos, see Figure 4. In the figure, we use the FM-cpo *Val* of references defined by:

$$\mathit{Val} \stackrel{\mathit{def}}{=} \mathit{Loc} + \mathit{Int} + \{\mathit{default}\}$$

where $\pi \cdot v \stackrel{\mathit{def}}{=} \mathsf{if}\ (v \notin \mathit{Loc})\ \mathsf{then}\ v\ \mathsf{else}\ \pi(v)$ and *default* denotes a default value used for type-incorrect expressions, such as the addition of two locations. The only nonstandard part is the semantics of the command type com, which we define in the continuation passing style

$$
\begin{array}{rcl@{\qquad}rcl}
\mathit{Val} & \stackrel{def}{=} & \mathit{Loc} + \mathit{Int} + \{\mathit{default}\} & O & \stackrel{def}{=} & \{\mathit{normal}, \mathit{err}\}_\perp \\
\mathit{Heap} & \stackrel{def}{=} & \mathit{Loc} \rightharpoonup_{\mathsf{fin}} \mathit{Val} \times \mathit{Val} & \mathit{Cont} & \stackrel{def}{=} & \mathit{Heap} \to O \\
[\![\mathsf{val} \to \tau]\!] & \stackrel{def}{=} & \mathit{Val} \to [\![\tau]\!] & [\![\tau \to \tau']\!] & \stackrel{def}{=} & [\![\tau]\!] \to [\![\tau']\!] \\
[\![\mathsf{com}]\!] & \stackrel{def}{=} & \mathit{Heap} \times \mathit{Cont} \to O & & & \\
[\![\Delta]\!] & \stackrel{def}{=} & \prod_{i \in \Delta} \mathit{Val} & [\![\Gamma]\!] & \stackrel{def}{=} & \prod_{x\colon \tau \in \Gamma} [\![\tau]\!].
\end{array}
$$

Figure 4: Interpretation of Types and Typing Contexts

following [19, 4]:

$$
\begin{array}{rcl@{\qquad}rcl}
O & \stackrel{def}{=} & \{\mathit{normal}, \mathit{err}\}_\perp \text{ (with } \pi \cdot o = o) & \mathit{Heap} & \stackrel{def}{=} & \mathit{Loc} \rightharpoonup_{\mathsf{fin}} \mathit{Val} \times \mathit{Val} \\
\mathit{Cont} & \stackrel{def}{=} & (\mathit{Heap} \to O) & [\![\mathsf{com}]\!] & \stackrel{def}{=} & (\mathit{Heap} \times \mathit{Cont} \to O).
\end{array}
$$

Here $A \times B$ and $A \to B$ are cartesian product and exponential in the category of FM-cpos. And $A \rightharpoonup_{\mathsf{fin}} B$ is the FM-cpo of the finite partial functions from A to B whose order and permutation action are defined below:

(1) $f \sqsubseteq g \stackrel{def}{\iff} \mathsf{dom}(f) = \mathsf{dom}(g)$ and $f(a) \sqsubseteq g(a)$ for all $a \in \mathsf{dom}(f)$,

(2) $(\pi \cdot f)(a) \stackrel{def}{=} \mathsf{if}\ (a \in \pi(\mathsf{dom}(f)))\ \mathsf{then}\ (\pi \cdot ((f \circ \pi^{-1})(a)))\ \mathsf{else}$ undefined.

The first FM-cpo O specifies all possible observations, which are normal termination *normal*, erroneous termination *err* or divergence $\perp$. The next FM-cpo *Heap* denotes the set of heaps. It formalizes that a heap contains only finitely many allocated cells and each cell in the heap has two fields. The third FM-cpo *Cont* represents the set of continuations that consume heaps. Finally, $[\![\mathsf{com}]\!]$ is the set of cps-style commands. Those commands take a current heap h and a continuation k, and compute an observation in O (often by computing a final heap h', and calling the given continuation k with h').

Note that *Heap* has the usual heap disjointness predicate $h \# h'$, which denotes the disjointness of $\mathsf{dom}(h)$ and $\mathsf{dom}(h')$, and the usual partial heap combining operator $\bullet$, which takes the union of (the graphs of) two disjoint heaps. The $\#$ predicate and $\bullet$ operator fit well with FM domain theory, because they preserve all permutations: $h \# h' \iff (\pi \cdot h) \# (\pi \cdot h')$ and $\pi \cdot (h \bullet h') = (\pi \cdot h) \bullet (\pi \cdot h')$.

The semantics of typing contexts Δ and Γ is given by cartesian products: $[\![\Delta]\!] \stackrel{def}{=} \prod_{i \in \Delta} \mathit{Val}$ and $[\![\Gamma]\!] \stackrel{def}{=} \prod_{x\colon \tau \in \Gamma} [\![\tau]\!]$. The products here are taken over finite families, so they give well-defined FM-cpos.[3] We will use symbols ρ and η to denote environments in $[\![\Delta]\!]$ and $[\![\Gamma]\!]$, respectively.

The semantics of expressions and terms is shown in Figures 5 and 6. It is standard, except for the case of allocation, where we make use of the underlying FM domain theory: The interpretation picks a location that is fresh with respect to currently known values (i.e., $\mathsf{supp}(h, \eta, \rho)$) as well as those that will be used by the continuation (i.e., $\mathsf{supp}(k)$). The cps-style interpretation gives us an explicit handle on which locations are used by the continuation, and the FM domain theory ensures that $\mathsf{supp}(h, \eta, \rho, k)$ is finite (so a new location l can be chosen) and that the choice of l does not matter, as long as l is not in

[3] An infinite product of FM-cpos is not necessarily an FM-cpo.

$$[\![\Delta \vdash E]\!] \;:\; [\![\Delta]\!] \to \mathit{Val}$$

$$[\![\Delta, i \vdash i]\!]_\rho \stackrel{def}{=} \rho(i) \qquad [\![\Delta \vdash 0]\!]_\rho \stackrel{def}{=} 0$$

$$[\![\Delta \vdash E_1 + E_2]\!]_\rho \stackrel{def}{=} \mathsf{if}\ ([\![E_1]\!]_\rho, [\![E_2]\!]_\rho \in \mathit{Int})\ \mathsf{then}\ ([\![E_1]\!]_\rho + [\![E_2]\!]_\rho)\ \mathsf{else}\ \mathit{default}$$

$$[\![\Delta \vdash E_1 - E_2]\!]_\rho \stackrel{def}{=} \mathsf{if}\ ([\![E_1]\!]_\rho, [\![E_2]\!]_\rho \in \mathit{Int})\ \mathsf{then}\ ([\![E_1]\!]_\rho - [\![E_2]\!]_\rho)\ \mathsf{else}\ \mathit{default}$$

Figure 5: Interpretation of Expressions

$$[\![\Delta \mid \Gamma \vdash M \colon \tau]\!] \;:\; [\![\Delta]\!] \times [\![\Gamma]\!] \to [\![\tau]\!]$$

$$[\![\Delta \mid \Gamma, x \colon \tau \vdash x \colon \tau]\!]_{\rho,\eta} \stackrel{def}{=} \eta(x)$$

$$[\![\Delta \mid \Gamma \vdash \lambda i.\, M \colon \mathsf{val} \to \tau]\!]_{\rho,\eta} \stackrel{def}{=} \lambda v \colon \mathit{Val}.\ [\![\Delta, i \mid \Gamma \vdash M \colon \tau]\!]_{\rho[i \to v],\eta}$$

$$[\![\Delta \mid \Gamma \vdash M\ E \colon \tau]\!]_{\rho,\eta} \stackrel{def}{=} ([\![\Delta \mid \Gamma \vdash M \colon \mathsf{val} \to \tau]\!]_{\rho,\eta})\ [\![E]\!]_\rho$$

$$[\![\Delta \mid \Gamma \vdash \lambda x \colon \tau'.\, M \colon \tau' \to \tau]\!]_{\rho,\eta} \stackrel{def}{=} \lambda m \colon [\![\tau']\!].\ [\![\Delta \mid \Gamma, x \colon \tau' \vdash M \colon \tau]\!]_{\rho,\eta[x \to m]}$$

$$[\![\Delta \mid \Gamma \vdash M\ N \colon \tau]\!]_{\rho,\eta} \stackrel{def}{=} ([\![\Delta \mid \Gamma \vdash M \colon \tau' \to \tau]\!]_{\rho,\eta})\ [\![\Delta \mid \Gamma \vdash N \colon \tau']\!]_{\rho,\eta}$$

$$[\![\Delta \mid \Gamma \vdash \mathsf{fix}\ M \colon \tau]\!]_{\rho,\eta} \stackrel{def}{=} \mathit{leastfix}\ [\![\Delta \mid \Gamma \vdash M \colon \tau \to \tau]\!]_{\rho,\eta}$$

$$[\![\Delta \mid \Gamma \vdash \mathsf{if}\ (E{=}F)\ \mathsf{then}\ M\ \mathsf{else}\ N \colon \mathsf{com}]\!]_{\rho,\eta} \stackrel{def}{=} \mathsf{if}\ [\![E]\!]_\rho{=}[\![F]\!]_\rho\ \mathsf{then}\ [\![\Delta \mid \Gamma \vdash M \colon \mathsf{com}]\!]_{\rho,\eta}\ \mathsf{else}\ [\![\Delta \mid \Gamma \vdash N \colon \mathsf{com}]\!]_{\rho,\eta}$$

$$[\![\Delta \mid \Gamma \vdash M; N \colon \mathsf{com}]\!]_{\rho,\eta}(h,k) \stackrel{def}{=} \mathsf{let}\ k'\ \mathsf{be}\ \lambda h'.\ [\![\Delta \mid \Gamma \vdash N \colon \mathsf{com}]\!]_{\rho,\eta}(h',k)\ \mathsf{in}\ [\![\Delta \mid \Gamma \vdash M \colon \mathsf{com}]\!]_{\rho,\eta}(h,k')$$

$$[\![\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \colon \mathsf{com}]\!]_{\rho,\eta}(h,k) \stackrel{def}{=} [\![\Delta, i \mid \Gamma \vdash M \colon \mathsf{com}]\!]_{\rho[i \to l],\eta}(h \bullet [l{\to}0, 0], k) \quad (\text{where } l \in (\mathit{Loc} - \mathsf{supp}(h,\rho,\eta,k)))$$

$$[\![\Delta \mid \Gamma \vdash \mathsf{free}(E) \colon \mathsf{com}]\!]_{\rho,\eta}(h,k) \stackrel{def}{=} \mathsf{if}\ [\![E]\!]_\rho \notin \mathsf{dom}(h)\ \mathsf{then}\ \mathit{err}\ \mathsf{else}\ (k(h')\ \text{for } h' \text{ s.t. } h' \bullet [[\![E]\!]_\rho{\to}h([\![E]\!]_\rho)] = h)$$

$$[\![\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}E.0\ \mathsf{in}\ M \colon \mathsf{com}]\!]_{\rho,\eta}(h,k) \stackrel{def}{=} \mathsf{if}\ [\![E]\!]_\rho \notin \mathsf{dom}(h)\ \mathsf{then}\ \mathit{err}\ \mathsf{else}\ \mathsf{let}\ (v,v') = h([\![E]\!]_\rho)\ \mathsf{in}\ [\![\Delta, i \mid \Gamma \vdash M \colon \mathsf{com}]\!]_{\rho[i \to v],\eta}(h,k)$$

$$[\![\Delta \mid \Gamma \vdash E.0 := F \colon \mathsf{com}]\!]_{\rho,\eta}(h,k) \stackrel{def}{=} \mathsf{if}\ [\![E]\!]_\rho \notin \mathsf{dom}(h)\ \mathsf{then}\ \mathit{err}\ \mathsf{else}\ (\mathsf{let}\ (v,v') = h([\![E]\!]_\rho)\ \mathsf{in}\ k(h[[\![E]\!]_\rho{\to}([\![F]\!]_\rho, v')]))$$

Figure 6: Interpretation of Terms

$\mathsf{supp}(h,\eta,\rho,k)$. (Formally, one shows by induction that the semantics is well-defined.) We borrowed this interpretation from Benton and Leperchey [4].

5. Relational Interpretation of Separation Logic

We now present the main result of this paper, a relational interpretation of separation logic. In this interpretation, a specification means a relation on terms, rather than a set of terms "satisfying" the specification. This relational reading formalizes the intuitive claim that proof rules in separation logic ensure parametricity with respect to the heap.

Our interpretation has two important components that ensure parametricity. The first is a Kripke structure $\mathcal{R}$. The possible worlds of $\mathcal{R}$ are finitely supported binary relations r on heaps,[4] and the accessibility relation is the preorder defined by the separating conjunction for relations:

$$\begin{array}{rcl} h_0[r * s]h_1 & \stackrel{def}{\Leftrightarrow} & \text{there exist splittings } n_0 \bullet m_0 = h_0 \text{ and } n_1 \bullet m_1 = h_1 \text{ such that} \\ & & \quad n_0[r]n_1 \text{ and } m_0[s]m_1, \\ r \sqsubseteq r' & \stackrel{def}{\Leftrightarrow} & \text{there exists } s \text{ such that } r * s = r'. \end{array}$$

Intuitively, $r \sqsubseteq r'$ means that r' is a $*$-extension of r by some s. The Kripke structure $\mathcal{R}$ parameterizes our interpretation, and it guarantees that all the logical connectives behave parametrically wrt. relations between internal resource invariants.

The second is *semantic quadruples*, which describe the relationship between two commands. We use the semantic quadruples to interpret Hoare triples relationally. Consider $c_0, c_1 \in [\![\mathsf{com}]\!]$ and $r, s \in \mathcal{R}$. For each subset D_0 of an FM-cpo D, define $\mathsf{eq}(D_0)$ to be the partial identity relation on D that equates only the elements in D_0. A *semantic quadruple* $[r](c_0, c_1)[s]$ holds iff

$$\begin{array}{l} \forall r' \in \mathcal{R}.\, \forall h_0, h_1 \in \mathit{Heap}.\, \forall k_0, k_1 \in \mathit{Cont}. \\ \quad (h_0[r * r']h_1 \wedge k_0[s * r' \to \mathsf{eq}(G)]k_1) \implies (c_0(h_0, k_0)[\mathsf{eq}(G)]c_1(h_1, k_1)), \end{array}$$

where G is the set $O - \{\mathit{err}\} = \{\mathit{normal}, \bot\}$ of good observations, and where $k_0[s * r' \to \mathsf{eq}(G)]k_1$ means that k_0, k_1 map heaps related in $s * r'$ into the diagonal of G. The above condition indirectly expresses that if the input heaps h_0, h_1 are $r * r'$-related, then the output heaps are related by $s * r'$. Note that the definition quantifies over relations r' for new heaps, thus implementing relational parametricity. In Section 7, we show how semantic quadruples are related to a more direct way of relating two commands and we also show that the parametricity in the definition of semantic quadruples implies the locality condition in separation logic [18].

The semantics of the logic is defined by the satisfaction relation $\models_{\Delta|\Gamma}$ between $[\![\Delta]\!] \times [\![\Gamma]\!]^2 \times \mathcal{R}$ and Specs, such that $\models_{\Delta|\Gamma}$ satisfies Kripke monotonicity:

$$(\rho, \eta_0, \eta_1, r \models_{\Delta|\Gamma} \varphi) \wedge (r \sqsubseteq r') \implies (\rho, \eta_0, \eta_1, r' \models_{\Delta|\Gamma} \varphi).$$

One way to understand the satisfaction relation is to assume two machines that execute the same set of terms. Each of these machines contains a chip that implements a module with a fixed set of operations. Intuitively, the $(\rho, \eta_0, \eta_1, r)$ parameter of $\models$ specifies the configurations of those machines: one machine uses (ρ, η_0) to bind free stack variables and identifiers of terms, and the other machine uses (ρ, η_1) for the same purpose; and the internal resources of the built-in modules in those machines are related by r. The judgment $(\rho, \eta_0, \eta_1, r) \models_{\Delta|\Gamma} \varphi$ means that if two machines are configured by $(\rho, \eta_0, \eta_1, r)$, then the meanings of the terms in two machines are φ-related. Note that we allow different environments for the Γ context only, not for the Δ context. This is because we are mainly concerned with parametricity with respect to the heap and only Γ entities, not Δ entities, depend on the heap.

Figure 7 shows the detailed interpretation of specifications. In the figure, we make use of the standard semantics of assertions [18]. We now explain three cases in the definition of $\models$.

[4] A relation r is finitely supported iff there is $L \subseteq_{\mathsf{fin}} \mathit{Loc}$ s.t. for every permutation π, if $\pi(l) = l$ for all $l \in L$, then $\forall h_0, h_1.\, h_0[r]h_1 \iff (\pi \cdot h_0)[r](\pi \cdot h_1)$.

For all environments $\rho \in [\![\Delta]\!]$ and $\eta_0, \eta_1 \in [\![\Gamma]\!]$ and all worlds $r \in \mathcal{R}$,

$$
\begin{array}{ll}
(\rho,\eta_0,\eta_1,r) \models \{P\}M\{Q\} & \stackrel{def}{\iff} [\mathsf{eq}([\![P]\!]_\rho) * r]([\![M]\!]_{\rho,\eta_0}, [\![M]\!]_{\rho,\eta_1})[\mathsf{eq}([\![Q]\!]_\rho) * r] \\
(\rho,\eta_0,\eta_1,r) \models \varphi \otimes P & \stackrel{def}{\iff} (\rho,\eta_0,\eta_1, r * \mathsf{eq}([\![P]\!]_\rho)) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models E = F & \stackrel{def}{\iff} [\![E]\!]_\rho = [\![F]\!]_\rho \\
(\rho,\eta_0,\eta_1,r) \models M = N & \stackrel{def}{\iff} [\![M]\!]_{\rho,\eta_0} = [\![N]\!]_{\rho,\eta_0} \text{ and } [\![M]\!]_{\rho,\eta_1} = [\![N]\!]_{\rho,\eta_1} \\
(\rho,\eta_0,\eta_1,r) \models \varphi \Rightarrow \psi & \stackrel{def}{\iff} \text{for all } s \in \mathcal{R}, \text{ if } (\rho,\eta_0,\eta_1, r * s) \models \varphi, \\
 & \qquad \text{then } (\rho,\eta_0,\eta_1, r * s) \models \psi \\
(\rho,\eta_0,\eta_1,r) \models \forall i.\, \varphi & \stackrel{def}{\iff} \text{for all } v \in \mathit{Val},\ (\rho[i{\to}v],\eta_0,\eta_1,r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \exists i.\, \varphi & \stackrel{def}{\iff} \text{there exists } v \in \mathit{Val} \text{ s.t. } (\rho[i{\to}v],\eta_0,\eta_1,r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \forall x\colon \tau.\, \varphi & \stackrel{def}{\iff} \text{for all } m,n \in [\![\tau]\!],\ (\rho,\eta_0[x{\to}m],\eta_1[x{\to}n],r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \exists x\colon \tau.\, \varphi & \stackrel{def}{\iff} \text{there exist } m,n \in [\![\tau]\!] \text{ s.t. } (\rho,\eta_0[x{\to}m],\eta_1[x{\to}n],r) \models \varphi \\
(\rho,\eta_0,\eta_1,r) \models \varphi \wedge \psi & \stackrel{def}{\iff} (\rho,\eta_0,\eta_1,r) \models \varphi \text{ and } (\rho,\eta_0,\eta_1,r) \models \psi \\
(\rho,\eta_0,\eta_1,r) \models \varphi \vee \psi & \stackrel{def}{\iff} (\rho,\eta_0,\eta_1,r) \models \varphi \text{ or } (\rho,\eta_0,\eta_1,r) \models \psi
\end{array}
$$

Figure 7: Relational Interpretation of Separation Logic

The first case is implication. Our interpretation of implication exploits the specific notion of accessibility in $\mathcal{R}$. It is equivalent to the standard Kripke semantics of implication:

$$\text{for all } r' \in \mathcal{R}, \text{ if } r \sqsubseteq r' \text{ and } (\rho,\eta_0,\eta_1,r') \models \varphi, \text{ then } (\rho,\eta_0,\eta_1,r') \models \psi,$$

because $r \sqsubseteq r'$ iff $r' = r * s$ for some s.

The second case is quantification. If a stack variable i is quantified, we consider one semantic value, but if an identifier x is quantified, we consider two semantic values. This is again to reflect that in our relational interpretation, we are mainly concerned with heap-dependent entities. Thus, we only read quantifiers for heap-dependent entities x relationally.

The last case is invariant extension $\varphi \otimes P$. Mathematically, it says that if we extend the r parameter by the partial equality for predicate P, specification φ holds. Intuitively, this means that some heap cells not appearing in a specification φ satisfy the invariant P.

A specification $\Delta \mid \Gamma \vdash \varphi$ is *valid* iff $(\rho,\eta_0,\eta_1,r) \models \varphi$ holds for all (ρ,η_0,η_1,r). A proof rule is *sound* when it is a valid axiom or an inference rule that concludes a valid specification from valid premises.

Lemma 5.1. *The axioms for $\otimes$ are sound.*

Proof. All the axioms for $\otimes$ have the form $\varphi \Rightarrow \psi$ or $\varphi \Leftrightarrow \psi$. When proving those axioms, we use the fact that $\varphi \Rightarrow \psi$ is valid if and only if $(\rho,\eta_0,\eta_1,r) \models \varphi$ implies $(\rho,\eta_0,\eta_1,r) \models \psi$ for all (ρ,η_0,η_1,r).

First, consider the generalized frame rule $\varphi \Rightarrow \varphi \otimes P$. Suppose that $(\rho,\eta_0,\eta_1,r) \models \varphi$. Then, by Kripke monotonicity, $(\rho,\eta_0,\eta_1, r * \mathsf{eq}([\![P]\!]_\rho)) \models \varphi$. Thus, $(\rho,\eta_0,\eta_1,r) \models \varphi \otimes P$.

Second, consider the distribution rule for triples. We prove the validity of this rule as follows:

$$\begin{array}{ll} & (\rho,\eta_0,\eta_1,r) \models \{P\}M\{Q\} \otimes P_0 \\ \iff & (\rho,\eta_0,\eta_1,r * \mathsf{eq}([\![P_0]\!]_\rho)) \models \{P\}M\{Q\} \qquad \text{(by the semantics of } \otimes P\text{)}. \\ \iff & [\mathsf{eq}([\![P]\!]_\rho) * \mathsf{eq}([\![P_0]\!]_\rho) * r]([\![M]\!]_{\rho,\eta_0}, [\![M]\!]_{\rho,\eta_1})[\mathsf{eq}([\![Q]\!]_\rho) * \mathsf{eq}([\![P_0]\!]_\rho) * r] \\ \iff & [\mathsf{eq}([\![P * P_0]\!]_\rho) * r]([\![M]\!]_{\rho,\eta_0}, [\![M]\!]_{\rho,\eta_1})[\mathsf{eq}([\![Q * P_0]\!]_\rho) * r] \\ \iff & (\rho,\eta_0,\eta_1,r) \models \{P * P_0\}M\{Q * P_0\} \qquad \text{(by the semantics of triples)}. \end{array}$$

The second equivalence is by the semantics of triples, and the third equivalence holds because eq maps $*$ for predicates to $*$ for relations.

Third, we prove the soundness of the distribution rules for equality. Note that the semantics of $E = F$ and $M = N$ is independent of the heap relation r in (ρ,η_0,η_1,r). Thus, once we fix the ρ,η_0,η_1 components, either $E = F$ and $M = N$ hold for all r, or $E = F$ and $M = N$ hold for no r. Let φ be $E = F$ or $M = N$. From the property of φ that we have just pointed out, it follows that

$$(\rho,\eta_0,\eta_1,r) \models \varphi \iff (\rho,\eta_0,\eta_1,r * \mathsf{eq}([\![P]\!]_\rho)) \models \varphi \iff (\rho,\eta_0,\eta_1,r) \models \varphi \otimes P.$$

Finally, consider all the remaining rules, which are distribution rules for logical connectives. All cases can be proved mostly by unrolling and rolling the definition of $\models$. Here we explain two cases. The first case is the distribution rule for existential quantification of i. We prove that this rule is sound below:

$$\begin{array}{ll} & (\rho,\eta_0,\eta_1,r) \models \exists i.\, \varphi \otimes P \\ \iff & \text{there exists } v \in \mathsf{val} \text{ s.t. } (\rho[i{\to}v],\eta_0,\eta_1,r) \models \varphi \otimes P \\ \iff & \text{there exists } v \in \mathsf{val} \text{ s.t. } (\rho[i{\to}v],\eta_0,\eta_1,r * \mathsf{eq}([\![P]\!]_{\rho[i\to v]})) \models \varphi \\ \iff & \text{there exists } v \in \mathsf{val} \text{ s.t. } (\rho[i{\to}v],\eta_0,\eta_1,r * \mathsf{eq}([\![P]\!]_\rho)) \models \varphi \quad (\text{since } i \notin \mathsf{fv}(P)) \\ \iff & (\rho,\eta_0,\eta_1,r * \mathsf{eq}([\![P]\!]_\rho)) \models \exists i\colon \delta.\, \varphi \\ \iff & (\rho,\eta_0,\eta_1,r) \models (\exists i\colon \delta.\, \varphi) \otimes P. \end{array}$$

All the equivalences except the third follow by rolling/unrolling the definition of $\models$. The next case is the rule for implication, which we prove sound as follows:

$$\begin{array}{ll} & (\rho,\eta_0,\eta_1,r) \models (\varphi \Rightarrow \psi) \otimes P \\ \iff & (\rho,\eta_0,\eta_1,r * \mathsf{eq}([\![P]\!]_\rho)) \models \varphi \Rightarrow \psi \\ \iff & \forall s.\, \big((\rho,\eta_0,\eta_1,r * \mathsf{eq}([\![P]\!]_\rho) * s) \models \varphi\big) \implies \big((\rho,\eta_0,\eta_1,r * \mathsf{eq}([\![P]\!]_\rho) * s) \models \psi\big) \\ \iff & \forall s.\, \big((\rho,\eta_0,\eta_1,r * s) \models \varphi \otimes P\big) \implies \big((\rho,\eta_0,\eta_1,r * s) \models \psi \otimes P\big) \\ \iff & (\rho,\eta_0,\eta_1,r) \models (\varphi \otimes P) \Rightarrow (\psi \otimes P). \end{array}$$

Again, all the equivalences are obtained by rolling/unrolling the definition of $\models$. □

Theorem 5.2. *All the proof rules are sound.*

Proof. The interpretation of all the logical connectives is standard, so that the semantics validates all the usual rules from first-order intuitionistic logic with equality. Moreover, by Lemma 5.1, all the rules about $\otimes$ are sound as well. Thus, it remains to show that the rules about Hoare triples and fixed point induction are sound.

Note that most of the rules about triples and fixed point induction have the form $\varphi \Rightarrow \psi$. When proving the soundness of those rules, we use the fact that $\varphi \Rightarrow \psi$ is valid if and only if $(\rho,\eta_0,\eta_1,r) \models \varphi$ implies $(\rho,\eta_0,\eta_1,r) \models \psi$ for all (ρ,η_0,η_1,r).

The first case is the rule for memory allocation:

$$(\forall i.\{P * i \mapsto 0,0\}M\{Q\}) \;\Rightarrow\; \{P\}\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M\{Q\}.$$

Consider $(\rho, \eta_0, \eta_1, r)$ satisfying the assumption of the above axiom. We need to prove that $(\rho, \eta_0, \eta_1, r)$ also satisfies the conclusion, i.e.,

$$[\mathsf{eq}(\llbracket P \rrbracket_\rho) * r](\llbracket \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \rrbracket_{\rho,\eta_0}, \llbracket \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \rrbracket_{\rho,\eta_1})[\mathsf{eq}(\llbracket Q \rrbracket_\rho) * r].$$

Choose arbitrary $h_0, h_1 \in \mathit{Heap}$, $k_0, k_1 \in \mathit{Cont}$, and $s \in \mathcal{R}$ such that

$$h_0[\mathsf{eq}(\llbracket P \rrbracket_\rho) * r * s]h_1 \ \text{ and } \ k_0[(\mathsf{eq}(\llbracket Q \rrbracket_\rho) * r * s) \to \mathsf{eq}(G)]k_1.$$

Pick $l \in \mathit{Loc} - \mathsf{supp}(h_0, h_1, \rho, \eta_0, \eta_1, k_0, k_1)$. Then, the FM domain theory ensures that for $j = 0, 1$,

$$\llbracket \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \rrbracket_{\rho,\eta_j}(h_j, k_j) = \llbracket M \rrbracket_{\rho[i\to l],\eta_j}(h_j \bullet [l{\to}0, 0], k_j). \tag{5.1}$$

Let ρ' be $\rho[i{\to}l]$, and let h'_j be $h_j{\bullet}[l{\to}0, 0]$. We prove the required relationship for $\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M$ as follows:

$$\begin{array}{ll}
 & h_0\big[\mathsf{eq}(\llbracket P \rrbracket_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[(\mathsf{eq}(\llbracket Q \rrbracket_\rho) * r * s) \to \mathsf{eq}(G)\big]k_1 \\
\Longrightarrow & h_0\big[\mathsf{eq}(\llbracket P \rrbracket_{\rho'}) * r * s\big]h_1 \ \wedge\ k_0\big[(\mathsf{eq}(\llbracket Q \rrbracket_{\rho'}) * r * s) \to \mathsf{eq}(G)\big]k_1 \\
\Longrightarrow & h'_0\big[\mathsf{eq}(\llbracket P * i \mapsto 0, 0 \rrbracket_{\rho'}) * r * s\big]h'_1 \ \wedge\ k_0\big[(\mathsf{eq}(\llbracket Q \rrbracket_{\rho'}) * r * s) \to \mathsf{eq}(G)\big]k_1 \\
\Longrightarrow & \llbracket M \rrbracket_{\rho',\eta_0}(h'_0, k_0)\big[\mathsf{eq}(G)\big]\llbracket M \rrbracket_{\rho',\eta_1}(h'_1, k_1) \\
\Longrightarrow & \llbracket \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \rrbracket_{\rho,\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big]\llbracket \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \rrbracket_{\rho,\eta_1}(h_1, k_1).
\end{array}$$

The first implication holds, because ρ and ρ' are different only for i but $i \notin \mathsf{fv}(P, Q)$. The second implication follows from the definition of h'_j, and the third implication from the assumption that $(\rho, \eta_0, \eta_1, r) \models \forall i.\, \{P * i \mapsto 0, 0\}M\{Q\}$. Finally, the last implication holds, because of the equation 5.1.

The second case is the axiom for lookup

$$(\forall i.\{P * E \mapsto i, E_1\}M\{Q\}) \ \Rightarrow\ \{\exists i.P * E \mapsto i, E_1\}\mathsf{let}\ i{=}E.0\ \mathsf{in}\ M\{Q\}.$$

Consider $(\rho, \eta_0, \eta_1, r)$ that satisfies $(\forall i.\{P * E \mapsto i, E_1\}M\{Q\})$, and pick arbitrary $h_0, h_1 \in \mathit{Heap}$, $k_0, k_1 \in \mathit{Cont}$ and $s \in \mathcal{R}$ such that

$$h_0\big[\mathsf{eq}(\llbracket \exists i.P * E \mapsto i, E_1 \rrbracket_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[\mathsf{eq}(\llbracket Q \rrbracket_\rho) * r * s \to \mathsf{eq}(G)\big]k_1.$$

Let l be $\llbracket E \rrbracket_\rho$ (which is well-defined since $i \notin \mathsf{fv}(E)$). By the first conjunct above, l is in $\mathsf{dom}(h_0) \cap \mathsf{dom}(h_1)$, and there exist v and ρ' such that

$$v{=}\mathsf{proj}_0(h_0(l)){=}\mathsf{proj}_0(h_1(l)), \ \ \rho'{=}\rho[i{\to}v], \ \text{ and } \ h_0\big[\mathsf{eq}(\llbracket P * E \mapsto i, E_1 \rrbracket_{\rho'}) * r * s\big]h_1.$$

Here proj_0 is the projection of the first component of pairs. The two equalities above about v and ρ' imply that for $j = 0, 1$,

$$\llbracket \mathsf{let}\ i{=}E.0\ \mathsf{in}\ M \rrbracket_{\rho,\eta_j}(h_j, k_j) = \llbracket M \rrbracket_{\rho',\eta_j}(h_j, k_j). \tag{5.2}$$

We derive the desired relationship about $\mathsf{let}\ i{=}E.0\ \mathsf{in}\ M$ as follows:

$$\begin{array}{ll}
 & k_0\big[\mathsf{eq}(\llbracket Q \rrbracket_\rho) * r * s \to \mathsf{eq}(G)\big]k_1 \ \wedge\ h_0\big[\mathsf{eq}(\llbracket P * E \mapsto i, E_1 \rrbracket_{\rho'}) * r * s\big]h_1 \\
\Longrightarrow & k_0\big[\mathsf{eq}(\llbracket Q \rrbracket_{\rho'}) * r * s \to \mathsf{eq}(G)\big]k_1 \ \wedge\ h_0\big[\mathsf{eq}(\llbracket P * E \mapsto i, E_1 \rrbracket_{\rho'}) * r * s\big]h_1 \\
\Longrightarrow & \llbracket M \rrbracket_{\rho',\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big]\llbracket M \rrbracket_{\rho',\eta_1}(h_1, k_1) \\
\Longrightarrow & \llbracket \mathsf{let}\ i{=}E.0\ \mathsf{in}\ M \rrbracket_{\rho,\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big]\llbracket \mathsf{let}\ i{=}E.0\ \mathsf{in}\ M \rrbracket_{\rho,\eta_1}(h_1, k_1).
\end{array}$$

The first implication holds because $i \notin \mathsf{fv}(Q)$, the second follows from the fact that $(\rho, \eta_0, \eta_1, r)$ satisfies the assumption of this axiom, and the last implication follows from the equation 5.2.

The third case is the axiom $\{E \mapsto \text{-}\}\mathsf{free}(E)\{\mathsf{emp}\}$. Choose arbitrary $(\rho, \eta_0, \eta_1, r)$, $h_0, h_1 \in \mathit{Heap}$, $k_0, k_1 \in \mathit{Cont}$, and $s \in \mathcal{R}$, such that

$$h_0\big[\mathsf{eq}(\llbracket E \mapsto \text{-} \rrbracket_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[\mathsf{eq}(\llbracket \mathsf{emp} \rrbracket_\rho) * r * s \to \mathsf{eq}(G)\big]k_1.$$

By the first conjunct above, there are splittings $m_0 \bullet n_0 = h_0$ and $m_1 \bullet n_1 = h_1$ such that $m_0[\mathsf{eq}([\![E \mapsto \text{-}]\!])]m_1$ and $n_0[r * s]n_1$. Note that the relationship between m_0 and m_1 implies that $[\![\mathsf{free}(E)]\!]_{\rho,\eta_j}(h_j, k_j) = k_j(n_j)$ for $j = 0, 1$. Thus, it is sufficient to show that $k_0(n_0)[\mathsf{eq}(G)]k_1(n_1)$. Note that n_0 and n_1 are already related by $r * s$, and k_0 and k_1 by $\mathsf{eq}([\![\mathsf{emp}]\!]_\rho) * r * s \to \mathsf{eq}(G)$. The conclusion follows from these two relationships, because $\mathsf{eq}([\![\mathsf{emp}]\!]_\rho) * r * s = r * s$.

The fourth case is the axiom $\{E \mapsto \text{-}, E_1\}E.0 := F\{E \mapsto F, E_1\}$. Choose arbitrary $(\rho, \eta_0, \eta_1, r)$, $h_0, h_1 \in \mathit{Heap}$, $k_0, k_1 \in \mathit{Cont}$, and $s \in \mathcal{R}$, such that

$$h_0\big[\mathsf{eq}([\![E \mapsto \text{-}, E_1]\!]_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[\mathsf{eq}([\![E \mapsto F, E_1]\!]_\rho) * r * s \to \mathsf{eq}(G)\big]k_1.$$

Because of the first conjunct, there are splittings $m_0 \bullet n_0 = h_0$ and $m_1 \bullet n_1 = h_1$ such that $m_0[\mathsf{eq}([\![E \mapsto \text{-}, E_1]\!]_\rho)]m_1$ and $n_0[r * s]n_1$. Let m' be the heap $[[\![E]\!]_\rho \to ([\![F]\!]_\rho, [\![E_1]\!]_\rho)]$. Then, we have the following two facts:

(1) $(m' \bullet n_0)\big[\mathsf{eq}([\![E \mapsto F, E_1]\!]_\rho) * r * s\big](m' \bullet n_1)$, and
(2) for all $j = 0, 1$, $[\![E.0 := F]\!]_{\rho,\eta_j}(h_j, k_j) = k_j(m' \bullet n_j)$.

By the first fact, $k_0(m' \bullet n_0)[\mathsf{eq}(G)]k_1(m' \bullet n_1)$. Now, the second fact gives the required $[\![E.0 := F]\!]_{\rho,\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big][\![E.0 := F]\!]_{\rho,\eta_1}(h_1, k_1)$.

The fifth case is the rule of Consequence. Suppose that $[\![P]\!]_\rho \subseteq [\![P']\!]_\rho$ and $[\![Q']\!]_\rho \subseteq [\![Q]\!]_\rho$, and $(\rho, \eta_0, \eta_1, r) \models \{P'\}M\{Q'\}$. Consider $h_0, h_1 \in \mathit{Heap}$, $k_0, k_1 \in \mathit{Cont}$, and $s \in \mathcal{R}$, such that

$$h_0[\mathsf{eq}([\![P]\!]_\rho) * r * s]h_1 \ \wedge\ k_0[\mathsf{eq}([\![Q]\!]_\rho) * r * s \to \mathsf{eq}(G)]k_1.$$

Since eq is monotone and $*$ preserves the subset order for relations,

$$\begin{array}{rcl} \mathsf{eq}([\![P]\!]_\rho) * r * s & \subseteq & \mathsf{eq}([\![P']\!]_\rho) * r * s, \quad \text{and} \\ [\mathsf{eq}([\![Q]\!]_\rho) * r * s \to \mathsf{eq}(G)] & \subseteq & [\mathsf{eq}([\![Q']\!]_\rho) * r * s \to \mathsf{eq}(G)]. \end{array}$$

Thus, $h_0[\mathsf{eq}([\![P']\!]_\rho) * r * s]h_1$ and $k_0[\mathsf{eq}([\![Q']\!]_\rho) * r * s \to \mathsf{eq}(G)]k_1$. These two relationships imply the required $[\![M]\!]_{\rho,\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big][\![M]\!]_{\rho,\eta_1}(h_1, k_1)$, because $(\rho, \eta_0, \eta_1, r)$ satisfies $\{P'\}M\{Q'\}$.

The sixth case is the rule for introducing existential quantification for assertions:

$$(\forall i.\{P\}M\{Q\}) \Rightarrow \{\exists i.P\}M\{\exists i.Q\}.$$

Consider $(\rho, \eta_0, \eta_1, r)$ that satisfies $\forall i.\{P\}M\{Q\}$. We should show that $(\rho, \eta_0, \eta_1, r)$ satisfies $\{\exists i.P\}M\{\exists i.Q\}$, i.e.,

$$[\mathsf{eq}([\![\exists i.P]\!]_\rho) * r]([\![M]\!]_{\rho,\eta_0}, [\![M]\!]_{\rho,\eta_1})[\mathsf{eq}([\![\exists i.Q]\!]_\rho) * r].$$

Pick arbitrary $h_0, h_1 \in \mathit{Heap}$, $k_0, k_1 \in \mathit{Cont}$, and $s \in \mathcal{R}$ such that

$$h_0[\mathsf{eq}([\![\exists i.P]\!]_\rho) * r * s]h_1 \ \text{ and } \ k_0[(\mathsf{eq}([\![\exists i.Q]\!]_\rho) * r * s) \to \mathsf{eq}(G)]k_1.$$

By the definition of eq, $[\![\exists i.P]\!]$ and $[\![\exists i.Q]\!]$, these two conjuncts imply the existence of v and ρ' such that

$$\rho' = \rho[i \to v], \ \ h_0[\mathsf{eq}([\![P]\!]_{\rho'}) * r * s]h_1, \ \text{ and } \ k_0[(\mathsf{eq}([\![Q]\!]_{\rho'}) * r * s) \to \mathsf{eq}(G)]k_1.$$

From what we have just shown, we derive the conclusion as follows:

$$\begin{array}{ll} & (h_0[\mathsf{eq}([\![P]\!]_{\rho'}) * r * s]h_1) \ \wedge\ (k_0[(\mathsf{eq}([\![Q]\!]_{\rho'}) * r * s) \to \mathsf{eq}(G)]k_1) \\ \Longrightarrow & [\![M]\!]_{\rho',\eta_0}(h_0, k_0)[\mathsf{eq}(G)][\![M]\!]_{\rho',\eta_1}(h_1, k_1) \ \text{(since } (\rho, \eta_0, \eta_1, r) \models \forall i.\{P\}M\{Q\}) \\ \Longrightarrow & [\![M]\!]_{\rho,\eta_0}(h_0, k_0)[\mathsf{eq}(G)][\![M]\!]_{\rho,\eta_1}(h_1, k_1) \qquad \text{(since } i \notin \mathsf{fv}(M)). \end{array}$$

The seventh case is the disjunction rule. Suppose that $(\rho, \eta_0, \eta_1, r)$ satisfies triples $\{P\}M\{Q\}$ and $\{P'\}M\{Q'\}$. Consider $h_0, h_1 \in \mathit{Heap}$, $s \in \mathcal{R}$, and $k_0, k_1 \in \mathit{Cont}$, such that

$$h_0\big[\mathsf{eq}([\![P \vee P']\!]_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[\mathsf{eq}([\![Q \vee Q']\!]_\rho) * r * s \to \mathsf{eq}(G)\big]k_1.$$

By the definition of $\mathsf{eq}([\![P \vee P']\!]_\rho)$, heaps h_0 and h_1 are related by $\mathsf{eq}([\![P]\!]_\rho) * r * s$ or $\mathsf{eq}([\![P']\!]_\rho) * r * s$. Without loss of generality, we assume that

$$h_0\big[\mathsf{eq}([\![P]\!]_\rho) * r * s\big]h_1. \tag{5.3}$$

Since eq is monotone and $*$ preserves the subset order for relations, relation $\mathsf{eq}([\![Q \vee Q']\!]_\rho) * r * s \to \mathsf{eq}(G)$ is a subset of $\mathsf{eq}([\![Q]\!]_\rho) * r * s \to \mathsf{eq}(G)$, and so,

$$k_0\big[\mathsf{eq}([\![Q]\!]_\rho) * r * s \to \mathsf{eq}(G)\big]k_1. \tag{5.4}$$

By the supposition, $(\rho, \eta_0, \eta_1, r)$ satisfies $\{P\}M\{Q\}$. Thus, the relationships 5.3 and 5.4 imply the required

$$[\![M]\!]_{\rho,\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big][\![M]\!]_{\rho,\eta_1}(h_1, k_1).$$

The eighth case is the rule for conditional statement. Suppose that $(\rho, \eta_0, \eta_1, r)$ satisfies $\{P \wedge E = F\}M\{Q\}$ and $\{P \wedge E \neq F\}N\{Q\}$. Consider $h_0, h_1 \in \mathit{Heap}$, $s \in \mathcal{R}$, and $k_0, k_1 \in \mathit{Cont}$, such that

$$h_0\big[\mathsf{eq}([\![P]\!]_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[\mathsf{eq}([\![Q]\!]_\rho) * r * s \to \mathsf{eq}(G)\big]k_1.$$

We do the case analysis depending on whether $[\![E]\!]_\rho = [\![F]\!]_\rho$. Suppose that $[\![E]\!]_\rho = [\![F]\!]_\rho$. In this case, $h_0\big[\mathsf{eq}([\![P \wedge E = F]\!]_\rho) * r * s\big]h_1$, and

$$[\![\mathsf{if}\ (E{=}F)\ \mathsf{then}\ M\ \mathsf{else}\ N]\!]_{\rho,\eta_j}(h_j, k_j) = [\![M]\!]_{\rho,\eta_j}(h_j, k_j) \text{ for all } j = 0, 1. \tag{5.5}$$

Using these facts, we derive the conclusion as follows:

$$\begin{array}{ll} & h_0\big[\mathsf{eq}([\![P \wedge E = F]\!]_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[\mathsf{eq}([\![Q]\!]_\rho) * r * s \to \mathsf{eq}(G)\big]k_1 \\ \Longrightarrow & [\![M]\!]_{\rho,\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big][\![M]\!]_{\rho,\eta_1}(h_1, k_1) \\ \Longrightarrow & [\![\mathsf{if}\ (E{=}F)\ \mathsf{then}\ M\ \mathsf{else}\ N]\!]_{\rho,\eta_0}(h_0, k_0)\big[\mathsf{eq}(G)\big][\![\mathsf{if}\ (E{=}F)\ \mathsf{then}\ M\ \mathsf{else}\ N]\!]_{\rho,\eta_1}(h_1, k_1). \end{array}$$

The first implication follows from our assumption that $\{P \wedge E = F\}M\{Q\}$ is satisfied by $(\rho, \eta_0, \eta_1, r)$, and the second follows from the equation 5.5 above. The other case $[\![E]\!]_\rho \neq [\![F]\!]_\rho$ can be proved similarly, so it is omitted here.

The ninth case is the rule for sequential composition. Suppose that $(\rho, \eta_0, \eta_1, r)$ satisfies $\{P\}M\{P_0\}$ and $\{P_0\}N\{Q\}$. Consider $h_0, h_1 \in \mathit{Heap}$, $s \in \mathcal{R}$, and $k_0, k_1 \in \mathit{Cont}$, such that

$$h_0\big[\mathsf{eq}([\![P]\!]_\rho) * r * s\big]h_1 \ \wedge\ k_0\big[\mathsf{eq}([\![Q]\!]_\rho) * r * s \to \mathsf{eq}(G)\big]k_1.$$

Let k'_j be $\lambda h'_j.[\![N]\!]_{\rho,\eta_j}(h'_j, k_j)$. Since $(\rho, \eta_0, \eta_1, r) \models \{P_0\}N\{Q\}$,

$$k'_0 = \lambda h'_0.[\![N]\!]_{\rho,\eta_0}(h'_0, k_0)\big[\mathsf{eq}([\![P_0]\!]_\rho) * r * s \to \mathsf{eq}(G)\big]\lambda h'_1.[\![N]\!]_{\rho,\eta_1}(h'_1, k_1) = k'_1.$$

Since $(\rho, \eta_0, \eta_1, r) \models \{P\}M\{Q_0\}$, the above relationship between k'_0 and k'_1 implies

$$[\![M]\!]_{\rho,\eta_0}(h_0, k'_0)\big[\mathsf{eq}(G)\big][\![M]\!]_{\rho,\eta_1}(h_1, k'_1).$$

This gives the conclusion, because $[\![M; N]\!]_{\rho,\eta_j}(h_j, k_j)$ is equal to $[\![M]\!]_{\rho,\eta_j}(h_j, k'_j)$, for all $j = 0, 1$.

The last case is the rule for fixed point induction. We note two properties of C and γ.

(1) For all ρ, η, if $\eta(x) = \bot$, then $[\![C(x)]\!]_{\rho,\eta} = \bot$.
(2) For all $(\rho, \eta_0, \eta_1, r)$, the following set is admissible:

$$\{(m_0, m_1) \mid (\rho, \eta_0[x{\to}m_0], \eta_1[x{\to}m_1], r) \models \gamma(x)\}.$$

These properties can be proved by a straightforward induction on the structure of C and γ. The soundness of the induction rule follows from the second property. □

6. A General Construction

Our Kripke semantics of specifications presented in the previous section is in fact an instance of a general, abstract construction that allows one to interpret a specification logic with higher-order frame rules. In this section, we describe the general construction. The remaining part of the paper can be read and understood without reading this section, in which we assume some basic knowledge of categorical logic (see, e.g., [5] for a quick recap).

Before explaining our construction, we remind the reader of FM-cousins of monoid, preorder, Heyting algebra and complete Heyting algebra. For FM-sets A, B, C, we call an element $a_0 \in A$, a function $f : A \times B \to C$ or a relation $r \subseteq A \times B$ *equivariant* when they preserve the permutation action in the following sense: for all $a \in A$, $b \in B$ and $\pi \in \mathsf{perm}$,

$$(\pi \cdot a_0 = a_0) \ \wedge \ (\pi \cdot (f(a,b)) = f(\pi \cdot a, \pi \cdot b)) \ \wedge \ ((a,b) \in r \iff (\pi \cdot a, \pi \cdot b) \in r).$$

An FM-monoid is an FM-set M with monoid operations ($I \in M$, $* : M \times M \to M$) such that I and $*$ are equivariant, and an FM-preorder is an FM-set A with an equivariant preorder $\sqsubseteq$ on A. An FM-Heyting algebra is an FM-poset $(A, \sqsubseteq)$ with operations

$$\bot, \top \in A, \quad \text{and} \quad \sqcup, \sqcap, \Rightarrow : A \times A \to A,$$

such that all of those operations are equivariant and $(A, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ forms a Heyting algebra. Finally, an FM-complete Heyting algebra is an FM-Heyting algebra $(A, \sqsubseteq, \bot, \top, \sqcup, \sqcap, \Rightarrow)$ such that every finitely supported subset of A has a least upper bound and a greatest lower bound.

Our construction starts with an FM-monoid $(M, I, *)$ in which $*$ is commutative. The FM-monoid $(M, I, *)$ generalizes the set of finitely supported binary relations r on heaps, where the monoid unit I is the singleton relation $([], [])$ of two empty heaps and the monoid operator $*$ is the separating conjunction for relations. Intuitively, each m in M represents information about the internal resource invariants of modules, and the $*$ operator of M is used to combine two pieces of information that describe disjoint resources. Throughout this section, we assume given a fixed FM-monoid $(M, I, *)$ with $*$ commutative, and describe a construction over this FM-monoid.

First, we define a preorder $\sqsubseteq$ for M:

$$m \sqsubseteq n \iff \exists m'. m * m' = n.$$

Intuitively, $m \sqsubseteq n$ means that n is an extension of m with information about additional disjoint resources.

Lemma 6.1. $(M, \sqsubseteq)$ *is an FM-preorder.*

It is well known that Kripke models of intuitionistic propositional logic are obtained by taking the upwards closed subsets of a preorder and that the upwards closed subsets form a complete Heyting algebra. Thus, our next step is to form such a model over M, but in the world of FM-sets. Hence we construct an FM-complete Heyting algebra $L(M)$ whose underlying set L consists of finitely supported upwards closed subsets of M, and which is

ordered by subset inclusion, denoted $\sqsubseteq_L$. The Heyting operations for L are defined in the standard way: when $M_0, M_1 \in L(M)$,

$$\bot \stackrel{def}{=} \emptyset \qquad \top \stackrel{def}{=} M$$
$$M_0 \sqcup M_1 \stackrel{def}{=} M_0 \cup M_1 \qquad M_0 \sqcap M_1 \stackrel{def}{=} M_0 \cap M_1$$
$$M_0 \Rightarrow M_1 \stackrel{def}{=} \{m \mid \forall m'.\, (m \sqsubseteq m' \wedge m' \in M_0) \implies m' \in M_1\}.$$

Lemma 6.2. *$(L(M), \sqsubseteq_L, \bot, \top, \sqcup, \sqcap, \Rightarrow)$ is an FM-complete Heyting algebra.*

The lattice $L(M)$ has two interesting properties, which we used in our semantics of separation logic. The first property is that the $\Rightarrow$ operator involves quantification over information about disjoint resources:

Lemma 6.3. *An element m belongs to $M_0 \Rightarrow M_1$ if and only if*

$$\forall m'.\, m * m' \in M_0 \implies m * m' \in M_1.$$

The second property is about the operator that frames in information about disjoint resources. We define a binary operator $- \otimes - : L(M) \times M \to L(M)$ by

$$M_0 \otimes m \stackrel{def}{=} \{m' \mid m' * m \in M_0\}.$$

Lemma 6.4. *The function $- \otimes -$ is well-defined, and it satisfies the following three properties:*

1. *$- \otimes m$ commutes with $\Rightarrow$ and all the existing least upper bounds or greatest lower bounds of subsets of $L(M)$.*
2. *$(M_0 \otimes m) \otimes m' = M_0 \otimes (m * m')$ for all $M_0 \in L(M)$ and $m, m' \in M$.*
3. *M_0 is a subset of $M_0 \otimes m$, for every $M_0 \in L(M)$.*

In our semantics of separation logic, we used this $\otimes$ operator to interpret invariant extension $\varphi \otimes P$, and designed its proof rules, based on the general properties of $\otimes$ summarized in the above lemma.

Finally, we construct a hyperdoctrine $\mathsf{FMSet}(-, L(M))$, which can be used to interpret the specification logic, including quantifiers and invariant extensions (i.e., $\varphi \otimes P$).

Lemma 6.5. *$\mathsf{FMSet}(-, L(M))$ satisfies all the axioms for hyperdoctrines, thereby allowing the interpretation of intuitionistic predicate logic.*

For each $m \in M$, consider the fibred endo-functor

$$\mathsf{FMSet}(-, - \otimes m) \quad : \quad \mathsf{FMSet}(-, L(M)) \to \mathsf{FMSet}(-, L(M)),$$

which maps a predicate φ over X, that is, an equivariant function φ from X to $L(M)$, to $(- \otimes m) \circ \varphi$.

Lemma 6.6. *The fibred functor $\mathsf{FMSet}(-, - \otimes m)$ preserves $\bot, \top, \sqcup, \sqcap, \Rightarrow$ in each fibre and commutes with quantifiers $\exists$ and $\forall$.*

In summary, the previous two lemmas provide alternative proofs to large parts of Lemma 5.1 and Theorem 5.2. In the proof of the latter theorem in the previous section we omitted the detailed proof of soundness of the rules for predicate logic; it is a consequence of the above Lemma 6.5. Finally, we remark that the general construction actually gives us more than we use in the previous section: First, since we have a hyperdoctrine, we in fact have a model of *higher-order* specification logic in which one can also model quantification

over specifications. Second, $L(M)$ is in fact not only an FM-complete Heyting algebra but an FM-complete BI algebra. This means that we can have $*$ and $-\!\!*$ connectives also for specifications. We have not yet made use of these additional facts.

7. Properties of Semantic Quadruples

In this section, we prove two properties of semantic quadruples. The first clarifies the connection between our new interpretation of Hoare triples and the standard interpretation, and the second shows how our cps-style semantic quadruples are related to a more direct way of relating two commands.

First, we consider the relation between semantic quadruples and Hoare triples. Define an operator cps that cps-transforms a state transformer semantically:

$$\begin{array}{rcl} \mathsf{cps} & : & (\mathit{Heap} \to (\mathit{Heap} + \{\mathit{err}\})_\perp) \to (\mathit{Heap} \times \mathit{Cont} \to O) \\ \mathsf{cps}(c) & \stackrel{def}{=} & \lambda(h,k).\ \mathsf{if}\ (c(h) \notin \{\perp, \mathit{err}\})\ \mathsf{then}\ k(c(h))\ \mathsf{else}\ c(h). \end{array}$$

Proposition 7.1. *For all $p, q \subseteq \mathit{Heap}$ and all $c \in \mathit{Heap} \to (\mathit{Heap} + \{\mathit{err}\})_\perp$, the quadruple $[\mathsf{eq}(p)](\mathsf{cps}(c), \mathsf{cps}(c))[\mathsf{eq}(q)]$ holds iff the two conditions below hold:*

1. *for every h in p, either $c(h) = \perp$ or $c(h) \in q$, hence $c(h)$ cannot be err;*
2. *for every h in p and h_1 such that $h \# h_1$,*
   (a) *if $c(h) = \perp$, then $c(h \bullet h_1) = \perp$,*
   (b) *if $c(h) \neq \perp$, then $c(h) \bullet h_1$ is defined and equal to $c(h \bullet h_1)$.*

Note that the first condition is the usual meaning of Hoare triples, and the second is the locality condition of commands in separation logic restricted to heaps in p [18]. Since the locality condition merely expresses the parametricity of commands with respect to new heaps, the proposition indicates that our interpretation of triples is the usual one enhanced by an additional parametricity requirement.

Proof of Proposition 7.1. ($\Rightarrow$) Pick an arbitrary heap h in p. Let k be the continuation defined by

$$k(h) \stackrel{def}{=} \mathsf{if}\ (h \in q)\ \mathsf{then}\ \perp\ \mathsf{else}\ \mathit{err}.$$

Then, $k[\mathsf{eq}(q) \to \mathsf{eq}(G)]k$ and $h[\mathsf{eq}(p)]h$. By the assumption on the validity of the quadruple, $\mathsf{cps}(c)(h,k)[\mathsf{eq}(G)]\mathsf{cps}(c)(h,k)$. By the definition of k, this relationship on $\mathsf{cps}(c)$ implies that $\mathsf{cps}(c)(h,k) = \perp$, which in turn gives

$$(c(h) = \perp) \vee (c(h) \in \mathit{Heap} \wedge k(c(h)) = \perp).$$

The second disjunct of this disjunction is equivalent to $c(h) \in q$ because $k(h') = \perp \iff h' \in q$. So, the disjunction gives the first condition.

For the second condition, consider h, h_1 such that $h \in p$ and $h \# h_1$. Let r be the relation $\{([], h_1)\}$ on heaps, and define three continuations k_0, k_1, k_2 as follows:

$$\begin{array}{rcl} k_0(h') & \stackrel{def}{=} & \mathit{normal}, \\ k_1(h') & \stackrel{def}{=} & \mathsf{if}\ (h' = c(h))\ \mathsf{then}\ \mathit{normal}\ \mathsf{else}\ \perp, \\ k_2(h') & \stackrel{def}{=} & \mathsf{if}\ (c(h) \in \mathit{Heap} \wedge h' = c(h) \bullet h_1)\ \mathsf{then}\ \mathit{normal}\ \mathsf{else}\ \perp. \end{array}$$

By the definition of r and k_i, we have that

$$h[\mathsf{eq}(p) * r](h \bullet h_1), \quad k_0[\mathsf{eq}(q) * r \to \mathsf{eq}(G)]k_0, \quad \text{and} \quad k_1[\mathsf{eq}(q) * r \to \mathsf{eq}(G)]k_2.$$

To see why the third relationship holds, note that if $h_1'[\mathsf{eq}(q) * r]h_2'$, then $h_1' \bullet h_1$ is defined and $h_2' = h_1' \bullet h_1$. Thus, $h_1' = c(h)$ holds precisely when $c(h) \in \mathit{Heap} \wedge h_2' = c(h) \bullet h_1$ holds. This implies that $k_1(h_1') = \mathit{normal}$ iff $k_2(h_2') = \mathit{normal}$. Now, by the assumption on the validity of the quadruple, we have that

$$\mathsf{cps}(c)(h, k_0)[\mathsf{eq}(G)]\mathsf{cps}(c)(h \bullet h_1, k_0) \text{ and } \mathsf{cps}(c)(h, k_1)[\mathsf{eq}(G)]\mathsf{cps}(c)(h \bullet h_1, k_2).$$

The first conjunct about k_0 implies that if $c(h) = \bot$, then $c(h \bullet h_1) = \bot$, and the second conjunct about k_1, k_2 implies that if $c(h) \neq \bot$, then $c(h \bullet h_1) = c(h) \bullet h_1$.

($\Leftarrow$) Consider a relation r on heaps and pick heaps h_1, h_2 and continuations k_1, k_2 such that

$$h_1[\mathsf{eq}(p) * r]h_2 \text{ and } k_1[\mathsf{eq}(q) * r \to \mathsf{eq}(G)]k_2.$$

Then, there exist two splittings $h_1' \bullet h_1'' = h_1$ and $h_2' \bullet h_2'' = h_2$ such that $h_1' = h_2' \in p$ and $h_1''[r]h_2''$. If $c(h_1) = \bot$, then $c(h_1') = \bot$ by the condition (2-b) of the assumption, and $c(h_2') = \bot$ by the condition (2-a) of the assumption. Thus, in this case, we have $\mathsf{cps}(c)(h_1, k_1) = \mathsf{cps}(c)(h_2, k_2) = \bot$ and $\mathsf{cps}(c)(h_1, k_1)[\mathsf{eq}(G)]\mathsf{cps}(c)(h_2, k_2)$, as desired. Otherwise, i.e., if $c(h_1) \neq \bot$, then $c(h_1') \neq \bot$ by the condition (2-a). Thus, by the condition (2-b), we have that $c(h_1) = c(h_1') \bullet h_1''$ and $c(h_2) = c(h_1') \bullet h_2''$. Since $c(h_1') \in q$ by the condition (1),

$$c(h_1) = c(h_1') \bullet h_1''[\mathsf{eq}(q) * r]c(h_1') \bullet h_2'' = c(h_2).$$

This implies $\mathsf{cps}(c)(h_1, k_1)[\mathsf{eq}(G)]\mathsf{cps}(c)(h_2, k_2)$, as desired. □

Next, we relate our cps-style notion of semantic quadruples to the direct-style alternative. The notion underlying this relationship is the observation closure, denoted $(-)^{\perp\!\!\perp}$. For each FM-cpo D and relation $r \subseteq D \times D$, we define two relations, $r^{\perp}$ on $[D \to O]$ and $r^{\perp\!\!\perp}$ on D, as follows:

$$\begin{aligned} k_1[r^{\perp}]k_2 &\overset{def}{\iff} \forall d_1, d_2 \in D.\ (d_1[r]d_2 \implies k_1(d_1)[\mathsf{eq}(G)]k_2(d_2)), \\ d_1[r^{\perp\!\!\perp}]d_2 &\overset{def}{\iff} \forall k_1, k_2 \in [D \to O].\ (k_1[r^{\perp}]k_2 \implies k_1(d_1)[\mathsf{eq}(G)]k_2(d_2)). \end{aligned}$$

Operator $(-)^{\perp}$ dualizes a relation on D to one on observations on D, and $(-)^{\perp\!\!\perp}$ closes a given relation r under observations.

Proposition 7.2. *Let r, s be relations in $\mathcal{R}$. Consider functions c_1, c_2 from Heap to $(\mathit{Heap} + \{\mathit{err}\})_\perp$. A quadruple $[r](\mathsf{cps}(c_1), \mathsf{cps}(c_2))[s]$ holds, iff*

$$\forall (r', h_1, h_2).\ h_1[r * r']h_2 \implies (c_1(h_1) = c_2(h_2) = \bot \vee c_1(h_1)[(s * r')^{\perp\!\!\perp}]c_2(h_2)).$$

This proposition shows that our semantic quadruples are close to what one might expect at first for relating two commands parametrically. The only difference is that our quadruple always closes the post-relation $s * r'$ under observations.

Proof of Proposition 7.2. ($\Rightarrow$) Consider r', h_1, h_2 such that $h_1[r * r']h_2$. We first show that

$$c_1(h_1) = \bot \iff c_2(h_2) = \bot.$$

Let k be the continuation $\lambda h'.\mathit{normal}$. Then, $k[s * r' \to \mathsf{eq}(G)]k$. By the assumption on the quadruple for $\mathsf{cps}(c_1), \mathsf{cps}(c_2)$, we have that

$$\mathsf{cps}(c_1)(h_1, k)[\mathsf{eq}(G)]\mathsf{cps}(c_2)(h_2, k).$$

This relationship implies that $c_1(h_1) = \bot \iff c_2(h_2) = \bot$, because $c_i(h_i) = \bot \iff \mathsf{cps}(c_i)(h_i, k) = \bot$ by the choice of k.

Next, we prove that if $c_1(h_1) \neq \bot$ or $c_2(h_2) \neq \bot$, then $c_1(h_1)[(s * r')^{\perp\!\!\perp}]c_2(h_2)$. By what we have just shown, $c_1(h_1) \neq \bot$ iff $c_2(h_2) \neq \bot$. We will assume that neither $c_1(h_1)$ nor $c_2(h_2)$ is $\bot$. Take two continuations k_1, k_2 such that $k_1[(s*r')^{\perp}]k_2$, i.e., $k_1[s*r' \to \mathsf{eq}(G)]k_2$. Since the quadruple $[r](\mathsf{cps}(c_1), \mathsf{cps}(c_2))[s]$ holds by assumption and $h_1[r * r']h_2$, we have that

$$\mathsf{cps}(c_1)(h_1, k_1)[\mathsf{eq}(G)]\mathsf{cps}(c_2)(h_2, k_2).$$

Since both $c_1(h_1)$ and $c_2(h_2)$ are different from $\bot$, the relationship is equivalent to

$$k_1(c_1(h_1))[\mathsf{eq}(G)]k_2(c_2(h_2)).$$

We have just shown that $c_1(h_1)[(s * r')^{\perp\!\!\perp}]c_2(h_2)$.

($\Leftarrow$) Pick an arbitrary relation r', heaps h_1, h_2 and continuations k_1, k_2 such that $h_1[r * r']h_2$ and $k_1[s * r' \to \mathsf{eq}(G)]k_2$ (i.e., $k_1[(s * r')^{\perp}]k_2$.) By the assumption of this if direction, either $c_1(h_1) = c_2(h_2) = \bot$ or $c_1(h_1)[(s * r')^{\perp\!\!\perp}]c_2(h_2)$. In the first case,

$$\mathsf{cps}(c_1)(h_1, k_1) = \bot\,[\mathsf{eq}(G)]\,\bot = \mathsf{cps}(c_2)(h_2, k_2),$$

and in the second case, both $c_1(h_1)$ and $c_2(h_2)$ are in *Heap*, so that

$$\mathsf{cps}(c_1)(h_1, k_1) = k_1(c_1(h_1))\,[\mathsf{eq}(G)]\,k_2(c_2(h_2)) = \mathsf{cps}(c_2)(h_2, k_2).$$

The conclusion follows from these two relationships. □

8. Abstraction Theorem

The abstraction theorem below formalizes that well-specified programs (specified in separation logic with implicit quantification over internal resource invariants by frame rules) behave relationally parametrically in internal resource invariants. The easiest way to understand this intuition may be from the corollary following the theorem.

Some readers might feel that it is too much to call the abstraction theorem a "theorem" since it really is a trivial corollary of the soundness theorem — but that is just as it should be: the semantics was defined to achieve that.

Theorem 8.1 (Abstraction Theorem)**.** *If $\Delta \mid \Gamma \vdash \varphi$ is provable in the logic, then for all $(\rho, \eta_0, \eta_1, r) \in [\![\Delta]\!] \times [\![\Gamma]\!]^2 \times \mathcal{R}$, we have that $(\rho, \eta_0, \eta_1, r) \models \varphi$.*

Proof. By Theorem 5.2, we get that $\Delta \mid \Gamma \vdash \varphi$ is valid, which is just what the conclusion expresses. □

Corollary 8.2. *Suppose that $\Delta \mid x\colon \mathsf{com} \vdash \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\}$ is provable in the logic. Then for all (ρ, c_0, c_1, r), if $[\mathsf{eq}([\![P_1]\!]_\rho) * r](c_0, c_1)[\mathsf{eq}([\![Q_1]\!]_\rho) * r]$ holds, then*

$$[\mathsf{eq}([\![P]\!]_\rho) * r]([\![M]\!]_{[x\to c_0]}, [\![M]\!]_{[x\to c_1]})[\mathsf{eq}([\![Q]\!]_\rho) * r]$$

holds as well.

Intuitively, x corresponds to a module with a single operation, and M a client of the module. This corollary says that if we prove a property of the client M, assuming only an abstract external specification $\{P_1\}x\{Q_1\}$ of the module, the client cannot tell apart two different implementations c_0, c_1 of the module, as long as c_0, c_1 have identical external behavior. The four instances of eq in the proposition formalize that the external behaviors of c_0, c_1 are identical and that the client M behaves the same externally regardless of whether it is used with c_0 or c_1. The relation r is a simulation relation for internal resource invariants of c_0 and c_1.

$$\forall j.\ \{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, j\}\ \mathsf{inc}_0\ \{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, (j{+}1)\}$$
$$\forall j.\ \{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, j * g \mapsto \text{-}\}\mathsf{read}_0\{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, j * g \mapsto \text{-}, j\}$$

$$\begin{array}{rcl} \mathsf{inc}_0 & \equiv & \mathsf{let}\ i{=}c.0\ \mathsf{in}\ (\mathsf{let}\ j{=}i.1\ \mathsf{in}\ i.1 := j{+}1) \\ \mathsf{read}_0 & \equiv & \mathsf{let}\ i{=}c.0\ \mathsf{in}\ (\mathsf{let}\ j{=}i.1\ \mathsf{in}\ g.1 := j) \end{array}$$

$$\forall j.\ \{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, j\}\ \mathsf{inc}_1\ \{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, (j{-}1)\}$$
$$\forall j.\ \{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, j * g \mapsto \text{-}\}\mathsf{read}_1\{\exists i.\, c \mapsto i, \text{-} * i \mapsto \text{-}, j * g \mapsto \text{-}, (-j)\}$$

$$\begin{array}{rcl} \mathsf{inc}_1 & \equiv & \mathsf{let}\ i{=}c.0\ \mathsf{in}\ (\mathsf{let}\ j{=}i.1\ \mathsf{in}\ i.1 := j{-}1) \\ \mathsf{read}_1 & \equiv & \mathsf{let}\ i{=}c.0\ \mathsf{in}\ (\mathsf{let}\ j{=}i.1\ \mathsf{in}\ g.1 := -j) \end{array}$$

$$\Delta \mid \Gamma \ \vdash\ (\{\mathsf{emp}\}\mathsf{inc}\{\mathsf{emp}\} \wedge \{g \mapsto \text{-}\}\mathsf{read}\{g \mapsto \text{-}\}) \ \Rightarrow\ \{g \mapsto \text{-}\}\mathsf{inc}; \mathsf{read}\{g \mapsto \text{-}\}$$
$$(\text{where } \Delta \equiv \{g, c\} \text{ and } \Gamma \equiv \{\mathsf{inc}\colon \mathsf{com}, \mathsf{read}\colon \mathsf{com}\})$$

Figure 8: Two Implementations of a Counter and a Simple Client

Proof of Corollary 8.2. Define environments η_0, η_1 and heap sets p, p_1, q, q_1 as follows:

$$\eta_0 = [x{\to}c_0],\ \eta_1 = [x{\to}c_1],\ \text{and}\ (p_1, q_1, p, q) = (\llbracket P_1 \rrbracket_\rho, \llbracket Q_1 \rrbracket_\rho, \llbracket P \rrbracket_\rho, \llbracket Q \rrbracket_\rho).$$

By Theorem 8.1, we have, for any r, that $(\rho, \eta_0, \eta_1, r) \models \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\}$. From this, we derive the conclusion of the proposition:

$$\begin{array}{ll} & (\rho, \eta_0, \eta_1, r) \models \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\} \\ \Longrightarrow & (\forall s \in \mathcal{R}.\ (\rho, \eta_0, \eta_1, r * s) \models \{P_1\}x\{Q_1\} \implies (\rho, \eta_0, \eta_1, r * s) \models \{P\}M\{Q\}) \\ \Longrightarrow & ((\rho, \eta_0, \eta_1, r) \models \{P_1\}x\{Q_1\} \implies (\rho, \eta_0, \eta_1, r) \models \{P\}M\{Q\}) \\ \Longrightarrow & ([\mathsf{eq}(p_1) * r](c_0, c_1)[\mathsf{eq}(q_1) * r] \implies [\mathsf{eq}(p) * r](\llbracket M \rrbracket_{\eta_0}, \llbracket M \rrbracket_{\eta_1})[\mathsf{eq}(q) * r]). \end{array}$$

□

9. Examples

Our first example is the two implementations of a counter in the introduction and the simple client ($\mathsf{inc}; \mathsf{read}$) in Example 3.1. We remind the reader of the implementations and the specification of the client in Figure 8 (here we use the formally correct 0 and 1 for the fields named data and next in the introduction for readability). The figure also shows the concrete specifications of the implementations. Note that the concrete specifications describe that both implementations use an internal cell $c.0$ to keep the value of the counter, and that the second implementation stores the negated value of the counter in this internal cell.

Pick a location $l \in \mathit{Loc}$ and an environment $\rho \in \llbracket \{c, g\} \rrbracket$ with $\rho(c) = l$, and define $f_0, f_1, g_0, g_1, b_0, b_1$ as follows:

$$f_i \stackrel{def}{=} \llbracket \mathsf{inc}_i \rrbracket_{\rho,[]},\quad g_i \stackrel{def}{=} \llbracket \mathsf{read}_i \rrbracket_{\rho,[]},\quad b_i \stackrel{def}{=} \llbracket \mathsf{inc}; \mathsf{read} \rrbracket_{\rho,[\mathsf{inc}\to f_i, \mathsf{read}\to g_i]}.$$

Now, by the Abstraction Theorem, we get that, for all r,

$$\begin{array}{c} \big([\mathsf{eq}(\llbracket \mathsf{emp} \rrbracket_\rho) * r](f_0, f_1)[\mathsf{eq}(\llbracket \mathsf{emp} \rrbracket_\rho) * r] \ \wedge\ [\mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho) * r](g_0, g_1)[\mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho) * r]\big) \\ \Rightarrow \\ [\mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho) * r](b_0, b_1)[\mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho) * r]. \end{array} \tag{9.1}$$

$$\forall i, v.\ \{i \mapsto \text{-}, v * k \mapsto \text{-}\}\mathsf{put}_0(i)\{k \mapsto \text{-}, v\}$$
$$\forall j, v.\ \{j \mapsto \text{-} * k \mapsto \text{-}, v\}\mathsf{get}_0(j)\{j \mapsto \text{-}, v * k \mapsto \text{-}, v\}$$

$$\begin{array}{rcl} \mathsf{put}_0 & \equiv & \lambda i.\, \mathsf{let}\ v = i.1\ \mathsf{in}\ (\mathsf{free}(i); k.1 := v) \\ \mathsf{get}_0 & \equiv & \lambda j.\, \mathsf{let}\ v = k.1\ \mathsf{in}\ j.1 := v \end{array}$$

$$\forall i, v.\ \{i \mapsto \text{-}, v * (\exists k'.\ k \mapsto k', \text{-} * k' \mapsto \text{-})\}\mathsf{put}_1(i)\{\exists k'.\ k \mapsto k', \text{-} * k' \mapsto \text{-}, v\}$$
$$\forall j, v.\ \{j \mapsto \text{-} * (\exists k'.\ k \mapsto k', \text{-} * k' \mapsto \text{-}, v)\}\mathsf{get}_1(j)\{j \mapsto \text{-}, v * (\exists k'.\ k \mapsto k', \text{-} * k' \mapsto \text{-}, v)\}$$

$$\begin{array}{rcl} \mathsf{put}_1 & \equiv & \lambda i.\, \mathsf{let}\ k'{=}k.0\ \mathsf{in}\ (\mathsf{free}(k'); k.0{:=}i) \\ \mathsf{get}_1 & \equiv & \lambda j.\, \mathsf{let}\ k'{=}k.0\ \mathsf{in}\ \mathsf{let}\ v{=}k'.1\ \mathsf{in}\ j.1{:=}v \end{array}$$

$$\Delta \mid \Gamma \ \vdash\ (\forall i.\{i \mapsto \text{-}\}\mathsf{put}(i)\{\mathsf{emp}\}) \wedge (\forall j.\{j \mapsto \text{-}\}\mathsf{get}(j)\{j \mapsto \text{-}\}) \ \Rightarrow\ \{j \mapsto \text{-}\}c\{j \mapsto \text{-}\}$$
$$(\text{where } \Delta \equiv \{j, k\} \text{ and } \Gamma \equiv \{\mathsf{put}\colon \mathsf{val} \to \mathsf{com}, \mathsf{get}\colon \mathsf{val} \to \mathsf{com}\})$$
$$c \ \equiv\ \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ (i.1{:=}5; \mathsf{put}(i); \mathsf{get}(j))$$

Figure 9: Two Implementations of a Buffer and a Simple Client

We now sketch a consequence of this result; for brevity we allow ourselves to be a bit informal. Let r be the following simulation relation between the two implementations:

$$r \stackrel{def}{=} \{\, (h_0, h_1) \mid \exists i \in \mathit{Loc}.\, \exists n \in \mathit{Int}.\, \exists v_0, v_1, v_0', v_1' \in \mathit{Val}.\ i \neq l \wedge h_0 = [c{\to}i, v_0] \bullet [i{\to}v_0', n] \wedge h_1 = [c{\to}i, v_1] \bullet [i{\to}v_1', -n] \,\}.$$

Then one can verify that the antecedent of the implication in (9.1) holds, and thus conclude that

$$[\mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho) * r](b_0, b_1)[\mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho) * r]$$

holds. Take $(h_0, h_1) \in \mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho) * r$, and denote the result of running b_0 on h_0 by h_0', and the result of running b_1 on h_1 by h_1'. We then conclude that h_0' will be of the form $h_{00}' \bullet h_{01}'$ and that h_1' will be of the form $h_{10}' \bullet h_{11}'$ with $(h_{01}', h_{11}') \in r$ and with $(h_{00}', h_{10}') \in \mathsf{eq}(\llbracket g \mapsto \text{-} \rrbracket_\rho)$.

Thus the relation between the internal resource invariants is maintained and, for the visible part, b_0 and b_1 both produce the *same heap* with exactly one cell.

The next example is a buffer of size one, and it illustrates the ownership transfer. Our buffer has operations put and get. Intuitively, $\mathsf{put}(i)$ stores the value found at i in the buffer, and $\mathsf{get}(j)$ retrieves the value stored in the buffer and stores it at j. We assume the following abstract specifications of this mutable abstract data type:

$$(\forall i.\, \{i \mapsto \text{-}\}\mathsf{put}(i)\{\mathsf{emp}\}) \quad \text{and} \quad (\forall j.\, \{j \mapsto \text{-}\}\mathsf{get}(j)\{j \mapsto \text{-}\}.$$

Figure 9 shows two implementations of the buffer and a client, as well as the concrete specifications for the implementations and the specification for the client. Note that the first implementation just uses one cell for the buffer and that the implementation follows the intuitive description given above. The second implementation uses two cells for the buffer. The additional cell is used to hold the cell pointed to by i itself. Note that this additional cell is transferred from the caller of $\mathsf{put}_2(i)$, i.e., a client of the buffer. Finally, the specification of the client describes the safety property of c, assuming the abstract specification for the buffer.

Pick $\rho \in [\![\{j,k\}]\!]$, and define $f_0, f_1, g_0, g_1, c_0, c_1$ by

$$f_i \stackrel{def}{=} [\![\mathsf{put}_i]\!]_{\rho,[]}, \quad g_i \stackrel{def}{=} [\![\mathsf{get}_i]\!]_{\rho,[]}, \quad c_i \stackrel{def}{=} [\![c]\!]_{\rho,[\mathsf{put}\to f_i,\mathsf{get}\to g_i]}.$$

Our Abstraction Theorem gives that, for all r,

$$\begin{array}{l}(\forall v \in \mathit{Val}.\ [\mathsf{eq}([\![i \mapsto \text{-}]\!]_{\rho[i\to v]}) * r](f_0(v), f_1(v))[\mathsf{eq}([\![\mathsf{emp}]\!]_{\rho[i\to v]}) * r]) \wedge \\ (\forall v \in \mathit{Val}.\ [\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho[j\to v]}) * r](g_0(v), g_1(v))[\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho[j\to v]}) * r]) \\ \quad \Rightarrow \quad [\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho}) * r](c_0, c_1)[\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho}) * r].\end{array} \tag{9.2}$$

This result implies that the client behaves the same no matter whether we run it with the first or second implementation of the buffer. To see this, let l be $\rho(k)$ and define a simulation relation r between the two implementations:

$$\begin{array}{rl} r \stackrel{def}{=} & \{\,(h_0,h_1) \mid \exists l' \in \mathit{Loc}.\, \exists n, v_0, v_1, v_1' \in \mathit{Val}. \\ & \qquad l \neq l' \ \wedge\ h_0 = [l{\to}v_0, n] \ \wedge\ h_1 = [l{\to}l', v_1] \bullet [l'{\to}v_1', n]\,\}. \end{array}$$

For this relation r, one can verify that the antecedent of the implication in (9.2) holds, and thus conclude that

$$[\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho}) * r](c_0, c_1)[\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho}) * r]$$

holds. This quadruple says, in particular, that c_0 and c_1 map $\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho}) * r$-related heaps to $\mathsf{eq}([\![j \mapsto \text{-}]\!]_{\rho}) * r$-related heaps, which means that they behave the same for cell j and preserve the r relation for the internal resource invariants of the two implementations.

10. Conclusion and Future Work

We have succeeded in defining the first relationally parametric model of separation logic. The model captures the informal idea that well-specified clients of mutable abstract data types should behave parametrically in the internal resource invariants of the abstract data type.

We see our work as a first step towards devising a logic for reasoning about mutable abstract data types, similar in spirit to Abadi and Plotkin's logic for parametricity [16, 6]. To this end, we also expect to make use of the ideas of relational separation logic in [21] for reasoning about relations between different programs syntactically. The logic should include a link between separation logic and relational separation logic so that one could get a syntactic representation of the semantic Abstraction Theorem and its corollary presented above.

One can also think of our work as akin to the O'Hearn-Reynolds model for idealized algol based on translation into a relationally parametric polymorphic linear lambda calculus [12]. In *loc. cit.* O'Hearn and Reynolds show how to provide a better model of stack variables for idealized algol by making a formal connection to parametricity. Here we provide a better model for the more unwieldy world of heap storage by making a formal connection to parametricity.

As mentioned in Section 3, the conjunction rule is not sound in our model. This is a consequence of our interpretation, which "bakes-in" the frame rule by quantifying over all relations r'. Indeed, using the characterization given by Proposition 7.2, one sees that for the conjunction rule

$$\big([r_1](\mathsf{cps}(c_1), \mathsf{cps}(c_2))[s_1] \wedge [r_2](\mathsf{cps}(c_1), \mathsf{cps}(c_2))[s_2]\big) \Longrightarrow [r_1 \wedge r_2](\mathsf{cps}(c_1), \mathsf{cps}(c_2))[s_1 \wedge s_2]$$

to hold, we would need something like $(r_1 \wedge r_2) * r = (r_1 * r) \wedge (r_2 * r)$ to hold. We "bake-in" the frame rule in order to get a model that validates a wide range of higher-order frame rules and it is known that already for second-order frame rules, the conjunction rule is not sound without some restrictions on the predicates involved [14]. We don't know whether it is possible to develop a parametric model in which the conjunction rule is sound.

Future work further includes developing a parametric model for the higher-order version of separation logic with explicit quantification over internal resource invariants. Finally, we hope that ideas similar to those presented here can be used to develop parametric models for other recent approaches to mutable abstract data types (e.g., [2]).

Acknowledgments

We would like to thank Nick Benton, Jacob Thamsborg and the anonymous referees for their insightful comments. This work was supported by FUR (FIRST). Yang was supported also by EPSRC.

References

[1] A. Banerjee and D. Naumann. Ownership Confinement Ensures Representation Independence for Object-oriented Programs. *Journal of the ACM*, 52(6):894–960, 2005.
[2] M. Barnett and D. Naumann. Towards imperative modules: Reasoning about invariants and sharing of mutable state. In *Proc. of LICS'04*, 2004.
[3] N. Benton. Abstracting Allocation:The New new Thing. In *Proc. of CSL'06*, 2006.
[4] N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. of TLCA'05*, pages 88–101, Nara, Japan, 2005.
[5] B. Biering, L. Birkedal, and N. Torp-Smith. BI-hyperdoctrines and higher order separation logic. In *Proc. of ESOP'05*, pages 233–247, Edinburgh, UK, 2005.
[6] L. Birkedal and R. Møgelberg. Categorical models for Abadi-Plotkin's logic for parametricity. *Mathematical Structures in Computer Science*, 15:709–772, 2005.
[7] L. Birkedal, N. Torp-Smith, and J. C. Reynolds. Local reasoning about a copying garbage collector. In *Proc. of POPL'04*, pages 220–231, Venice, Italy, 2004.
[8] L. Birkedal, N. Torp-Smith, and H. Yang. Semantics of separation-logic typing and higher-order frame rules. In *Proc. of LICS'05*, pages 260–269, 2005.
[9] L. Birkedal and H. Yang. Relational Parametricity and Separation Logic. In *Proc. of FOSSACS'07*, pages 93–107, 2007.
[10] I. Mijajlović, N. Torp-Smith, and P. O'Hearn. Refinement and separation context. In *Proc. of FSTTCS'04*, pages 421–433, Chennai, India, 2004.
[11] I. Mijajlović and H. Yang. Data refinements with low-level pointer operations. In *Proc. of APLAS'05*, pages 19–36, Tsukuba, Japan, 2005.
[12] P. O'Hearn and J. Reynolds. From Algol to polymorphic linear lambda-calculus. *Journal of the ACM*, 47(1):167–223, 2000.
[13] P. W. O'Hearn, H. Yang, and J. C. Reynolds. Local reasoning about programs that alter data structures. In *Proc. of CSL'01*, pages 1–19, Paris, France, 2001.
[14] P. W. O'Hearn, H. Yang, and J. C. Reynolds. Separation and information hiding. In *Proc. of POPL'04*, pages 268–280, Venice, Italy, 2004.
[15] M. Parkinson and G. Bierman. Separation logic and abstraction. In *Proc. of POPL'05*, pages 247–258, Long Beach, CA, USA, 2005.
[16] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of TLCA'93*, pages 361–375, Utrecht, Netherlands, 1993.
[17] J. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.

[18] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS'02*, pages 55–74, Copenhagen, Denmark, 2002.
[19] M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342:28–55, 2005.
[20] N. Torp-Smith. *Advances in Separation Logic — A Study of Logics for Reasoning about Stateful Programs*. PhD thesis, IT University of Copenhagen, 2005.
[21] H. Yang. Relational separation logic. *Theoretical Comput. Sci.*, 2005. (to appear).