

Relational Parametricity and Separation Logic

Lars Birkedal¹ and Hongseok Yang²

¹ IT University of Copenhagen, Denmark

² Queen Mary, University of London, 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 [20, 19, 8, 17, 13, 6] 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, 7] 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 [7]. Moreover, in [14] a second-order frame rule was proved sound and in [7] a whole range of higher-order frame rules were proved sound for a separation-logic type system.

In [14] and [7] 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 k , which describes the external behavior of k . Suppose further that an actual implementation m of the module satisfies

$$\{P_1 * R\} m \{Q_1 * R\}.$$

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

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

and combine it with the specification for m to obtain

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

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 [4, 3] 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

$$\exists R. \{P_1 * R\} k \{Q_1 * R\}.$$

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 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 [18].

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 the purposes of the present paper, we have decided to 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.³ Our model validates a whole range of higher-order frame

³ 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, it appears that we cannot have the usual reading of assertions as sets of heaps because, then, the rule of consequence is not sound.

rules, as in [7], but here we achieve that for a more standard presentation of separation logic and not only for a separation-logic type system as in [7].

Technically, it has proven to be a very non-trivial problem to define a parametric model for separation logic. We describe the challenges and give an overview of the main ideas in our approach in the following section. In Section 3 we describe the programming and the assertion language we consider and in Section 4 we define our version of separation logic. In Section 5 we define the semantics of our programming language in the category of FM-cpos and we define our relational interpretation of separation logic in Section 6. 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 briefly describe an example in Section 9 and finally we conclude and discuss future work in Section 10. For reasons of space most proofs have been placed in appendix.

2 Challenges and Main Ideas

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 [22, 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 [9, 10]. 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 [9], 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 [10].

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 [2], 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 5 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 [7] 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.

3 Programs and Assertions

In this paper, we consider a higher-order language with immutable local (stack) variables. Types and preterms of the language are defined as follows:

$$\begin{array}{l}
\text{Data types } \delta ::= \text{ref} \mid \text{int} \qquad \text{Phrase types } \tau ::= \text{com} \mid \delta \rightarrow \tau \mid \tau \rightarrow \tau \\
\text{Expressions } E, F ::= i \mid \text{nil} \mid E + E \mid E - E \mid n \quad (n \in \text{Integers}) \\
\text{Terms } M, N ::= x \mid \lambda i: \delta. M \mid M E \mid \lambda x: \tau. M \mid M M \mid \text{fix } M \mid \text{if } (E=E) M M \mid M; M \\
\quad \mid \text{let } i=\text{new} \text{ in } M \mid \text{free}(E) \mid \text{let } i=E.f \text{ in } M \mid E.f := E \quad (f \in \{0, 1, 2\})
\end{array}$$

The language separates expressions E from terms M . Expressions have data types δ and are bound to *local variables* i, j, \dots , while terms have phrase types τ and are bound to *identifiers* x, y, \dots . The language also includes two kinds of abstraction and application, one for expression parameters and the other for term parameters. The main reason for this separation is to guarantee that expressions always terminate and do not access heap storage. We will use this guarantee in the language as well as in the logic. For instance, the language passes expression parameters by value, while passing term parameters by name, but it maintains a simple semantics of function type $\delta \rightarrow \tau$, which does not involve any use of monad. This clean optimization of parameter passing is possible, because of the guarantee about expressions.

To simplify presentation, we take a simple storage model where each heap cell has three fields, numbered 0, 1, and 2. The first field stores an integer, and the last two fields store locations (values of type `ref`). Command `let i =new in M` allocates such a ternary heap cell, binds the address of the cell to i , and runs M under this binding. The f 'th field ($f \in \{0, 1, 2\}$) of this newly allocated cell at address i is read by `let $j = i.f$ in N` and updated by `$i.f := E$` . The cell i is deallocated by `free(i)`.

The language uses typing judgments of the form $\Delta \vdash E : \delta$ and $\Delta \mid \Gamma \vdash M : \tau$, where Δ and Γ are standard type environments for local variables i and identifiers x , respectively. The typing rules for expressions and terms are standard, and appear in Appendix A.

We use the standard assertions from separation logic to describe properties of the heap, after adjusting the points-to predicate to ternary cells:⁴

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

The points-to predicate ($E \mapsto E_1, E_2, E_3$) means that the current heap has only one cell at address E and that the three fields of the cell contain values E_1, E_2, E_3 . 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 following, we often omit type decorations on quantified variables, when the omitted types do not matter or they can be constructed from context.

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

4 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:

$$\begin{aligned} \varphi, \psi ::= & \{P\}M\{Q\} \mid \varphi \otimes P \mid E = E \mid M = M \\ & \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \Rightarrow \varphi \mid \forall x: \tau. \varphi \mid \exists x: \tau. \varphi \mid \forall i: \delta. \varphi \mid \exists i: \delta. \varphi \end{aligned}$$

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 $\Delta \mid \Gamma \vdash \varphi : \text{Specs}$, where we overloaded **Specs** to mean the type for specifications.

The logic includes all the usual proof rules from the 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 1 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 semantics of assertions P, P', Q, Q' , which is defined in Appendix B, in order to express semantic implications between those assertions. The rules for invariant extension formalize higher-order frame rules, extending the idea in [7]. The generalized higher-order frame rule $\varphi \Rightarrow \varphi \otimes P$ adds an invariant P to specification

⁴ We omit separating implication \multimap to simplify presentation.

PROOF RULES FOR HOARE TRIPLES

$$\begin{aligned}
& (\forall i. \{P\}M\{Q\}) \Rightarrow \{\exists i. P\}M\{\exists i. Q\} \quad (\text{where } i \notin \text{fv}(M)) \\
& (\{P\}M\{Q\} \vee \{P'\}M\{Q'\}) \Rightarrow \{P \vee P'\}M\{Q \vee Q'\} \\
\{P \wedge E=E\}M\{Q\} \wedge \{P \wedge E \neq F\}N\{Q\} & \Rightarrow \{P\}\text{if } (E=F) M N\{Q\} \\
\{P\}M\{P_0\} \wedge \{P_0\}N\{Q\} & \Rightarrow \{P\}M; N\{Q\} \\
(\forall i. \{P * i \mapsto 0, \text{nil}, \text{nil}\}M\{Q\}) & \Rightarrow \{P\}\text{let } i=\text{new in } M\{Q\} \quad (\text{where } i \notin \text{fv}(P, Q)) \\
(\forall i. \{P * E \mapsto i, E_1, E_2\}M\{Q\}) & \Rightarrow \{\exists i. P * E \mapsto i, E_1, E_2\}\text{let } i=E.0 \text{ in } M\{Q\} \\
& \quad (\text{where } i \notin \text{fv}(Q)) \\
\{E \mapsto E_0, E_1, E_2\}\text{free}(E)\{\text{emp}\} & \quad \{E \mapsto E_0, E_1, E_2\}E.0 := F\{E \mapsto F, E_1, E_2\} \\
& \quad \frac{\llbracket P \rrbracket_\rho \subseteq \llbracket P' \rrbracket_\rho \text{ and } \llbracket Q' \rrbracket_\rho \subseteq \llbracket Q \rrbracket_\rho \text{ for all } \rho \in \llbracket \Delta \rrbracket}{\Delta \mid \Gamma \vdash \{P'\}M\{Q'\} \Rightarrow \{P\}M\{Q\}}
\end{aligned}$$

PROOF RULES FOR INVARIANT EXTENSION – $\otimes P$

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

RULE FOR FIXED POINT INDUCTION

$$\begin{aligned}
C & ::= [] \mid \lambda i. \delta. C \mid C E \mid \lambda x. \tau. C \mid C M \mid \text{fix } C \mid C; M \\
\gamma & ::= \{P\}C\{Q\} \mid \gamma \wedge \gamma \mid \forall x. \tau. \gamma \mid \forall i. \delta. \gamma \\
& \quad (\forall x. \gamma(x) \Rightarrow \gamma(M x)) \Rightarrow \gamma(\text{fix } M)
\end{aligned}$$

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

Fig. 1. Sample Proof Rules

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:

$$\frac{\frac{\frac{\Delta \mid \Gamma, x: \text{com} \vdash \{P\}x\{Q\} \Rightarrow \{P'\}M(x)\{Q'\}}{\Delta \mid \Gamma, x: \text{com} \vdash (\{P\}x\{Q\} \Rightarrow \{P'\}M(x)\{Q'\}) \otimes P_0}}{\Delta \mid \Gamma, x: \text{com} \vdash \{P\}x\{Q\} \otimes P_0 \Rightarrow \{P'\}M(x)\{Q'\} \otimes P_0}}{\Delta \mid \Gamma, x: \text{com} \vdash \{P * P_0\}x\{Q * P_0\} \Rightarrow \{P' * P_0\}M(x)\{Q' * P_0\}}$$

The last rule is for fixed point induction, and it relies on the restriction that a specification is of the form $\gamma(\text{fix } M)$. 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 \perp , so allowing us to omit a usual base case, “ $\gamma(\perp)$,” 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.

5 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 $\text{perm} \times A \rightarrow A$, such that (1) $\text{id} \cdot a = a$ 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.,

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

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 $\text{supp}(a)$. 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* if and only if 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.

We interpret data types δ as discretely ordered FM-cpos:

$$\begin{aligned} \llbracket \text{int} \rrbracket &\stackrel{\text{def}}{=} \text{Integers} = \{\dots, -2, -1, 0, 1, 2, 3, \dots\}, & \pi \cdot n &\stackrel{\text{def}}{=} n \\ \llbracket \text{ref} \rrbracket &\stackrel{\text{def}}{=} Loc + \{\text{nil}\}, & \pi \cdot v &\stackrel{\text{def}}{=} \text{if } (v = \text{nil}) \text{ then } \text{nil} \text{ else } \pi(v) \end{aligned}$$

Note that permutations π act as the identity on $\llbracket \text{int} \rrbracket$, but they rename locations for $\llbracket \text{ref} \rrbracket$.

Phrase types are interpreted as pointed FM-cpos, using the categorical structure of the category of FM-cpos, see Figure 2. The only nonstandard part is the semantics of the command type com , which we define in the continuation passing style following [21, 2]. The command type is interpreted as follows:

$$\begin{aligned} O &\stackrel{\text{def}}{=} \{\text{normal}, \text{err}\}_\perp \text{ (with } \pi \cdot o = o) & \text{Heap} &\stackrel{\text{def}}{=} Loc \rightarrow_{\text{fin}} \llbracket \text{int} \rrbracket \times \llbracket \text{ref} \rrbracket \times \llbracket \text{ref} \rrbracket \\ \text{cont} &\stackrel{\text{def}}{=} (\text{Heap} \rightarrow O) & \llbracket \text{com} \rrbracket &\stackrel{\text{def}}{=} (\text{Heap} \times \text{cont} \rightarrow O). \end{aligned}$$

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

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

$\llbracket \text{int} \rrbracket \stackrel{\text{def}}{=} \text{Integers}$	$\llbracket \text{ref} \rrbracket \stackrel{\text{def}}{=} \text{Loc} + \{\text{nil}\}$
$\llbracket \delta \rightarrow \tau \rrbracket \stackrel{\text{def}}{=} \llbracket \delta \rrbracket \rightarrow \llbracket \tau \rrbracket$	$\llbracket \tau \rightarrow \tau' \rrbracket \stackrel{\text{def}}{=} \llbracket \tau \rrbracket \rightarrow \llbracket \tau' \rrbracket$
$\llbracket \text{com} \rrbracket \stackrel{\text{def}}{=} \text{Heap} \times \text{cont} \rightarrow O$ (where $O = \{\text{normal}, \text{err}\}_\perp$ and $\text{cont} = \text{Heap} \rightarrow O$)	
$\llbracket \Delta \rrbracket \stackrel{\text{def}}{=} \prod_{i:\delta \in \Delta} \llbracket \delta \rrbracket$	$\llbracket \Gamma \rrbracket \stackrel{\text{def}}{=} \prod_{x:\tau \in \Gamma} \llbracket \tau \rrbracket$

Fig. 2. Interpretation of Types and Typing Contexts

$\llbracket \Delta \vdash E : \delta \rrbracket : \llbracket \Delta \rrbracket \rightarrow \llbracket \delta \rrbracket$	
$\llbracket \Delta, i : \delta \vdash i : \delta \rrbracket_\rho \stackrel{\text{def}}{=} \rho(i)$	$\llbracket \Delta \vdash \text{nil} : \text{ref} \rrbracket_\rho \stackrel{\text{def}}{=} \text{nil}$
$\llbracket \Delta \vdash E + F : \text{int} \rrbracket_\rho \stackrel{\text{def}}{=} \llbracket \Delta \vdash E : \text{int} \rrbracket_\rho + \llbracket \Delta \vdash F : \text{int} \rrbracket_\rho$	$\llbracket \Delta \vdash n : \text{int} \rrbracket_\rho \stackrel{\text{def}}{=} n$
$\llbracket \Delta \vdash E - F : \text{int} \rrbracket_\rho \stackrel{\text{def}}{=} \llbracket \Delta \vdash E : \text{int} \rrbracket_\rho - \llbracket \Delta \vdash F : \text{int} \rrbracket_\rho$	

Fig. 3. Interpretation of Expressions

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

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 three fields. The third FM-cpo *cont* represents the set of continuations that consume heaps. Finally, $\llbracket \text{com} \rrbracket$ 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 $\text{dom}(h)$ and $\text{dom}(h')$, 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: $\llbracket \Delta \rrbracket \stackrel{\text{def}}{=} \prod_{i:\delta \in \Delta} \llbracket \delta \rrbracket$ and $\llbracket \Gamma \rrbracket \stackrel{\text{def}}{=} \prod_{x:\tau \in \Gamma} \llbracket \tau \rrbracket$. Note that the products here are well-defined, because the domains of Δ and Γ are finite and so, the products are taken over finite families.

The semantics of expressions and terms is shown in Figures 3 and 4. 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., $\text{supp}(h, \eta, \rho)$) as well as those that will be used by the continuation (i.e., $\text{supp}(k)$). The cps-style interpretation gives us an explicit handle on which locations are used by the continuation, and the FM domain theory ensures that $\text{supp}(h, \eta, \rho, k)$ 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 $\text{supp}(h, \eta, \rho, k)$. We borrowed this interpretation from Benton and Leperchey [2].

$$\begin{aligned}
& \llbracket \Delta \mid \Gamma \vdash M : \tau \rrbracket : \llbracket \Delta \rrbracket \times \llbracket \Gamma \rrbracket \rightarrow \llbracket \tau \rrbracket \\
& \llbracket \Delta \mid \Gamma, x : \tau \vdash x : \tau \rrbracket_{\rho, \eta} \stackrel{def}{=} \eta(x) \\
& \llbracket \Delta \mid \Gamma \vdash \lambda i : \delta. M : \delta \rightarrow \tau \rrbracket_{\rho, \eta} \stackrel{def}{=} \lambda v : \llbracket \delta \rrbracket. \llbracket \Delta, i : \delta \mid \Gamma \vdash M : \tau \rrbracket_{\rho[i \rightarrow v], \eta} \\
& \llbracket \Delta \mid \Gamma \vdash M E : \tau \rrbracket_{\rho, \eta} \stackrel{def}{=} (\llbracket \Delta \mid \Gamma \vdash M : \delta \rightarrow \tau \rrbracket_{\rho, \eta}) \llbracket \Delta \vdash E : \delta \rrbracket_{\rho, \eta} \\
& \llbracket \Delta \mid \Gamma \vdash \lambda x : \tau'. M : \tau' \rightarrow \tau \rrbracket_{\rho, \eta} \stackrel{def}{=} \lambda m : \llbracket \tau' \rrbracket. \llbracket \Delta \mid \Gamma, x : \tau' \vdash M : \tau \rrbracket_{\rho, \eta[x \rightarrow m]} \\
& \llbracket \Delta \mid \Gamma \vdash M N : \tau \rrbracket_{\rho, \eta} \stackrel{def}{=} (\llbracket \Delta \mid \Gamma \vdash M : \tau' \rightarrow \tau \rrbracket_{\rho, \eta}) \llbracket \Delta \mid \Gamma \vdash N : \tau' \rrbracket_{\rho, \eta} \\
& \llbracket \Delta \mid \Gamma \vdash \text{fix } M : \tau \rrbracket_{\rho, \eta} \stackrel{def}{=} \text{leastfix } \llbracket \Delta \mid \Gamma \vdash M : \tau \rightarrow \tau \rrbracket_{\rho, \eta} \\
& \llbracket \Delta \mid \Gamma \vdash \text{if } (E=F) M N : \text{com} \rrbracket_{\rho, \eta} \stackrel{def}{=} \text{if } (\llbracket \Delta \vdash E : \delta \rrbracket_{\rho} = \llbracket \Delta \vdash F : \delta \rrbracket_{\rho}) \\
& \quad \text{then } \llbracket \Delta \mid \Gamma \vdash M : \text{com} \rrbracket_{\rho, \eta} \\
& \quad \text{else } \llbracket \Delta \mid \Gamma \vdash N : \text{com} \rrbracket_{\rho, \eta} \\
& \llbracket \Delta \mid \Gamma \vdash M ; N : \text{com} \rrbracket_{\rho, \eta}(h, k) \stackrel{def}{=} \text{let } k' \text{ be } \lambda h'. \llbracket \Delta \mid \Gamma \vdash N : \text{com} \rrbracket_{\rho, \eta}(h', k) \\
& \quad \text{in } \llbracket \Delta \mid \Gamma \vdash M : \text{com} \rrbracket_{\rho, \eta}(h, k') \\
& \llbracket \Delta \mid \Gamma \vdash \text{let } i = \text{new in } M : \text{com} \rrbracket_{\rho, \eta}(h, k) \stackrel{def}{=} \llbracket \Delta, i : \text{ref} \mid \Gamma \vdash M : \text{com} \rrbracket_{\rho[i \rightarrow l], \eta}(h \cdot [l \rightarrow \text{init}], k) \\
& \quad (\text{where } \text{init} = (0, \text{nil}, \text{nil}) \text{ and } l \in (\text{Loc} - \text{supp}(h, \rho, \eta, k))) \\
& \llbracket \Delta \mid \Gamma \vdash \text{free}(E) : \text{com} \rrbracket_{\rho, \eta}(h, k) \stackrel{def}{=} \text{let } l \text{ be } \llbracket \Delta \vdash E \rrbracket_{\rho} \\
& \quad \text{in if } (l \notin \text{dom}(h)) \text{ then } \text{err} \\
& \quad \quad \text{else } (k(h')) \text{ for } h' \text{ s.t. } h' \bullet [l \rightarrow h(l)] = h \\
& \llbracket \Delta \mid \Gamma \vdash \text{let } i = E.f \text{ in } M : \text{com} \rrbracket_{\rho, \eta}(h, k) \stackrel{def}{=} \text{let } l \text{ be } \llbracket \Delta \vdash E \rrbracket_{\rho} \\
& \quad \text{in if } (l \notin \text{dom}(h)) \text{ then } \text{err} \\
& \quad \quad \text{else } \llbracket \Delta, i : \delta \mid \Gamma \vdash M : \text{com} \rrbracket_{\rho[i \rightarrow \text{proj}_f(h(l))], \eta}(h, k) \\
& \llbracket \Delta \mid \Gamma \vdash E.f = F : \text{com} \rrbracket_{\rho, \eta}(h, k) \stackrel{def}{=} \text{let } l \text{ be } \llbracket \Delta \vdash E \rrbracket_{\rho} \text{ and } v \text{ be } \llbracket \Delta \vdash F \rrbracket_{\rho} \\
& \quad \text{in if } (l \notin \text{dom}(h)) \text{ then } \text{err} \text{ else } k(h[l \rightarrow h(l)][f \rightarrow v])
\end{aligned}$$

Fig. 4. Interpretation of Terms

6 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,⁵ and the accessibility relation is the preorder defined by the separating conjunction for relations:

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

⁵ A relation r is finitely supported if and only if there is $L \subseteq_{\text{fin}} \text{Loc}$ s.t. for every permutation π , if $\pi(l) = l$ for all $l \in L$, then $\forall h_0, h_1. h_0[r]h_1 \iff (\pi \cdot h_0)[r](\pi \cdot h_1)$.

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 $c_0, c_1 \in \llbracket \text{com} \rrbracket$ and $r, s \in \mathcal{R}$. For each subset D_0 of an FM-cpo D , define $\text{Eq}(D_0)$ 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 if and only if

$$\forall r' \in \mathcal{R}. \forall h_0, h_1 \in \text{Heap}. \forall k_0, k_1 \in \text{cont}. \\ (h_0[r * r']h_1 \wedge k_0[s * r' \rightarrow \text{Eq}(G)]k_1) \implies (c_0(h_0, k_0)[\text{Eq}(G)]c_1(h_1, k_1)),$$

where G is the set $O - \{\text{err}\} = \{\text{normal}, \perp\}$ of good observations. 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 usual locality condition in separation logic [20].

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

$$(\rho, \eta_0, \eta_1, r \models_{\Delta|\Gamma} \varphi) \wedge (r \sqsubseteq r') \implies (\rho, \eta_1, \eta_2, r' \models_{\Delta|\Gamma} \varphi).$$

One way to understand the satisfaction relation is to assume two machines that execute terms in the context of one specific module. 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 local variables and identifiers of terms, and the other machine uses (ρ, η_1) for the same purposes; and the internal resource invariants of the modules in those machines are related by r . The judgment $(\rho, \eta_0, \eta_1, r)$ 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 5 shows the detailed interpretation of specifications. In the figure, we make use of the semantics of assertions, which is standard [20] and included in Appendix B. We now explain three cases in the definition of \models .

The first case is implication. Our definition of 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 .

For all environments $\rho \in \llbracket \Delta \rrbracket$ and $\eta_0, \eta_1 \in \llbracket \Gamma \rrbracket$ and all worlds $r \in \mathcal{R}$,	
$(\rho, \eta_0, \eta_1, r) \models \{P\}M\{Q\}$	$\stackrel{def}{\iff} [\mathbf{Eq}(\llbracket P \rrbracket_\rho) * r](\llbracket M \rrbracket_{\rho, \eta_0}, \llbracket M \rrbracket_{\rho, \eta_1})[\mathbf{Eq}(\llbracket Q \rrbracket_\rho) * r]$
$(\rho, \eta_0, \eta_1, r) \models \varphi \otimes P$	$\stackrel{def}{\iff} (\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho)) \models \varphi$
$(\rho, \eta_0, \eta_1, r) \models E = F$	$\stackrel{def}{\iff} \llbracket E \rrbracket_\rho = \llbracket F \rrbracket_\rho$
$(\rho, \eta_0, \eta_1, r) \models M = N$	$\stackrel{def}{\iff} \llbracket M \rrbracket_{\rho, \eta_0} = \llbracket N \rrbracket_{\rho, \eta_0} \text{ and } \llbracket M \rrbracket_{\rho, \eta_1} = \llbracket N \rrbracket_{\rho, \eta_1}$
$(\rho, \eta_0, \eta_1, r) \models \varphi \Rightarrow \psi$	$\stackrel{def}{\iff} \text{for all } s \in \mathcal{R}, \text{ if } (\rho, \eta_0, \eta_1, r * s) \models \varphi, \text{ then } (\rho, \eta_0, \eta_1, r * s) \models \psi$
$(\rho, \eta_0, \eta_1, r) \models \forall i: \delta. \varphi$	$\stackrel{def}{\iff} \text{for all } v \in \llbracket \delta \rrbracket, (\rho[i \rightarrow v], \eta_0, \eta_1, r) \models \varphi$
$(\rho, \eta_0, \eta_1, r) \models \exists i: \delta. \varphi$	$\stackrel{def}{\iff} \text{there exists } v \in \llbracket \delta \rrbracket \text{ s.t. } (\rho[i \rightarrow v], \eta_0, \eta_1, r) \models \varphi$
$(\rho, \eta_0, \eta_1, r) \models \forall x: \tau. \varphi$	$\stackrel{def}{\iff} \text{for all } m, m' \in \llbracket \tau \rrbracket, (\rho, \eta_0[x \rightarrow m], \eta_1[x \rightarrow m'], r) \models \varphi$
$(\rho, \eta_0, \eta_1, r) \models \exists x: \tau. \varphi$	$\stackrel{def}{\iff} \text{there exist } m, m' \in \llbracket \tau \rrbracket \text{ s.t. } (\rho, \eta_0[x \rightarrow m], \eta_1[x \rightarrow m'], r) \models \varphi$
$(\rho, \eta_0, \eta_1, r) \models \varphi \wedge \psi$	$\stackrel{def}{\iff} (\rho, \eta_0, \eta_1, r) \models \varphi \text{ and } (\rho, \eta_0, \eta_1, r) \models \psi$
$(\rho, \eta_0, \eta_1, r) \models \varphi \vee \psi$	$\stackrel{def}{\iff} (\rho, \eta_0, \eta_1, r) \models \varphi \text{ or } (\rho, \eta_0, \eta_1, r) \models \psi$

Fig. 5. Relational Interpretation of Separation Logic

The second case is quantification. If a local 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* if and only if $(\rho, \eta_0, \eta_1, r) \models \varphi$ holds for all $(\rho, \eta_0, \eta_1, r)$. A proof rule is *sound* when it is a valid axiom or an inference rule that concludes a valid specification from valid premises.

Theorem 1. *All the proof rules in our logic are sound.*

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:

$$\begin{aligned} \text{cps}_D & : (\text{Heap} \rightarrow (\text{Heap} + \{\text{err}\})_\perp) \rightarrow (\text{Heap} \times \text{cont} \rightarrow O) \\ \text{cps}_D(f) & \stackrel{def}{=} \lambda(h, k). \text{if } (f(h) \notin \{\perp, \text{err}\}) \text{ then } k(f(h)) \text{ else } f(h). \end{aligned}$$

Proposition 1. For all $p, q \subseteq \text{Heap}$ and all $c \in \text{Heap} \rightarrow (\text{Heap} \times D + \{\text{err}\})_{\perp}$, the quadruple $[\text{Eq}(p)](\text{cps}(c), \text{cps}(c))[\text{Eq}(q)]$ holds if and only if the following two conditions 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 the locality condition of commands in separation logic restricted to heaps in p [20]. 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.

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 $(-)^{\perp\perp}$, and also called perping. For each FM-cpo D and relation $r \subseteq D \times D$, we define two relations, r^{\perp} on $[D \rightarrow O]$ and $r^{\perp\perp}$ on D , as follows:

$$\begin{aligned} k_1[r^{\perp}]k_2 &\stackrel{\text{def}}{\iff} \forall d_1, d_2 \in D. (d_1[r]d_2 \implies k_1(d_1)[\text{Eq}(G)]k_2(d_2)), \\ d_1[r^{\perp\perp}]d_2 &\stackrel{\text{def}}{\iff} \forall k_1, k_2 \in [D \rightarrow O]. (k_1[r^{\perp}]k_2 \implies k_1(d_1)[\text{Eq}(G)]k_2(d_2)). \end{aligned}$$

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

Proposition 2. Let r, s be relations in \mathcal{R} , and let c_1, c_2 be functions of type $\text{Heap} \rightarrow (\text{Heap} + \{\text{err}\})_{\perp}$. A quadruple $[r](\text{cps}(c_1), \text{cps}(c_2))[s]$ holds, iff

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

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.

8 Abstraction Theorem

The abstraction theorem below formalizes that well-specified programs (specified in separation logic with implicit quantification over internal resource invariants qua 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 2 (Abstraction Theorem). If $\Delta \mid \Gamma \vdash \varphi$ is provable in the logic, then for all $(\rho, \eta_0, \eta_1, r) \in \llbracket \Delta \rrbracket \times \llbracket \Gamma \rrbracket^2 \times \mathcal{R}$, we have that $(\rho, \eta_0, \eta_1, r) \models \varphi$.

Proof. By Theorem 1, we get that $\Delta \mid \Gamma \vdash \varphi$ is valid, which is just what the conclusion expresses. \square

Corollary 1. *Suppose that $\Delta \mid x:\text{com} \vdash \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\}$ is provable in the logic. Then for all (ρ, c_0, c_1, r) , if $[\text{Eq}(\llbracket P_1 \rrbracket_\rho) * r](c_0, c_1)[\text{Eq}(\llbracket Q_1 \rrbracket_\rho) * r]$ holds, then $[\text{Eq}(\llbracket P \rrbracket_\rho) * r](\llbracket M \rrbracket_{[x \rightarrow c_0]}, \llbracket M \rrbracket_{[x \rightarrow c_1]})[\text{Eq}(\llbracket Q \rrbracket_\rho) * r]$ 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 .

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

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

By Theorem 2, 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:

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

\square

9 Example

For reasons of space we only include one very simple example (but at least it does involve ownership transfer).

We will consider a mutable abstract data type that is a buffer of size one. It has operations `put` and `get`. Intuitively, `put(i)` stores the value found at $i.0$ in the buffer and `get(i)` retrieves the value stored in the buffer and stores it at $i.0$. Let $P_1 \equiv i \mapsto j, -, -,$ and $Q_1 \equiv \text{emp}$, and $P_2 \equiv i \mapsto -, -, -,$ and $Q_2 \equiv i \mapsto -, -, -,$ where $-$ denotes existentially quantified variables. We assume the following abstract specifications of this mutable abstract data type:

$$\{P_1\}\text{put}(i)\{Q_1\} \quad \text{and} \quad \{P_2\}\text{get}(i)\{Q_2\}.$$

Figure 6 shows two implementations of the buffer and a client. The figure also includes the concrete specifications for the implementation and a specification for the buffer. Note that the first implementation just uses one cell for the buffer and that the implementation follows the intuitive description given above. The

$$\begin{aligned}
\text{put}_1 &\equiv \left(\lambda i. \text{let } j = i.0 \text{ in } (\text{free}(i); k.0 := j) \right) & \text{get}_1 &\equiv \left(\lambda i. \text{let } j = k.0 \text{ in } i.0 := j \right) \\
&\quad \{i \mapsto j, -, - * k \mapsto -, -, -\} \text{put}_1(i) \{k \mapsto -, -, -\} \\
&\quad \{i \mapsto -, -, - * k \mapsto -, -, -\} \text{get}_1(i) \{i \mapsto -, -, - * k \mapsto -, -, -\} \\
\text{put}_2 &\equiv \left(\lambda i. \text{let } k' = k.1 \text{ in } (\text{free}(k'); k.1 := i) \right) & \text{get}_2 &\equiv \left(\lambda i. \text{let } k' = k.1 \text{ in let } j = k'.0 \text{ in } i.0 := j \right) \\
&\quad \{i \mapsto j, -, - * \exists k'. k \mapsto -, k', - * k' \mapsto -, -, -\} \text{put}_2(i) \{\exists k'. k \mapsto -, k', - * k' \mapsto -, -, -\} \\
&\quad \{i \mapsto -, -, - * \exists k'. k \mapsto -, k', - * k' \mapsto -, -, -\} \text{get}_2(i) \{i \mapsto -, -, - * \exists k'. k \mapsto -, k', - * k' \mapsto -, -, -\} \\
c &\equiv \left(\text{let } i'' = \text{new in } i''.0 := 5; \text{put}(i''); \text{get}(i'') \right) \\
\Delta \mid \Gamma \vdash & (\forall i. \{P_1\} \text{put}(i) \{Q_1\} \wedge \{P_2\} \text{get}(i) \{Q_2\}) \Rightarrow \{i' \mapsto -, -, -\} c \{i' \mapsto -, -, -\} \\
& \text{(where } \Delta = \{i': \text{ref}, k: \text{ref}\} \text{ and } \Gamma = \{\text{put}: \text{ref} \rightarrow \text{com}, \text{get}: \text{ref} \rightarrow \text{com}\})
\end{aligned}$$

Fig. 6. Two Implementations of a Buffer and a Simple Client

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 $\text{put}_2(i)$, 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 $\rho \in \llbracket \{i': \text{ref}, k: \text{ref}\} \rrbracket$, and define $f_1, f_2, g_1, g_2, c_1, c_2$ as follows:

$$f_i \stackrel{\text{def}}{=} \llbracket \text{put}_i \rrbracket_{\rho, []}, \quad g_i \stackrel{\text{def}}{=} \llbracket \text{get}_i \rrbracket_{\rho, []}, \quad c_i \stackrel{\text{def}}{=} \llbracket c \rrbracket_{\rho, [\text{put} \rightarrow f_i, \text{get} \rightarrow g_i]}.$$

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

$$\begin{aligned}
& (\forall v \in \llbracket \text{ref} \rrbracket. [\text{Eq}(\llbracket P_1 \rrbracket_{\rho[i \mapsto v]} * r)](f_1(v), f_2(v)) [\text{Eq}(\llbracket Q_1 \rrbracket_{\rho[i \mapsto v]} * r)] \wedge \\
& \quad [\text{Eq}(\llbracket P_2 \rrbracket_{\rho[i \mapsto v]} * r)](g_1(v), g_2(v)) [\text{Eq}(\llbracket Q_2 \rrbracket_{\rho[i \mapsto v]} * r)] \quad (1) \\
& \Rightarrow [\text{Eq}(\llbracket i' \mapsto -, -, - \rrbracket_{\rho} * r)](c_1, c_2) [\text{Eq}(\llbracket i' \mapsto -, -, - \rrbracket_{\rho} * r)].
\end{aligned}$$

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

$$r = \{(h_1, h_2) \mid \exists j. h_1 = [k \rightarrow j, -, -] \wedge \exists k'. h_2 = [k \rightarrow -, k', -] \bullet [k' \rightarrow j, -, -]\}.$$

Then one can verify that the antecedent of the implication in (1) holds, and thus conclude that

$$[\text{Eq}(\llbracket i' \mapsto -, -, - \rrbracket_{\rho} * r)](c_1, c_2) [\text{Eq}(\llbracket i' \mapsto -, -, - \rrbracket_{\rho} * r)]$$

holds. Take $(h_1, h_2) \in \text{Eq}(\llbracket i' \mapsto -, -, - \rrbracket_{\rho} * r)$, and denote the result of running c_1 on h_1 by h'_1 , and the result of running c_2 on h_2 by h'_2 . We then conclude that h'_1 will be of the form $h'_{11} \bullet h'_{12}$ and that h'_2 will be of the form $h'_{21} \bullet h'_{22}$ with $(h'_{12}, h'_{22}) \in r$ and with $(h'_{11}, h'_{21}) \in \text{Eq}(\llbracket i' \mapsto -, -, - \rrbracket_{\rho})$.

Thus the relation between the internal resource invariants is maintained and, for the visible part, c_1 and c_2 both produce the *same heap* with exactly one cell.

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, 5]. To this end, we also expect to make use of the ideas of relational separation logic in [23] 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 local 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 4, the conjunction rule is not sound in our model. This is a consequence of our cps-style interpretation. 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., [11, 1]).

References

1. M. Barnett and D. Naumann. Towards imperative modules: Reasoning about invariants and sharing of mutable state. In *Proc. of LICS’04*, 2004.
2. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. of TLCA’05*, pages 88–101, Nara, Japan, 2005.
3. B. Biering, L. Birkedal, and N. Torp-Smith. BI hyperdoctrines, higher-order separation logic, and abstraction. ITU-TR-2005-69, IT University of Copenhagen.
4. 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.
5. L. Birkedal and R. Møgelberg. Categorical models for Abadi-Plotkin’s logic for parametricity. *Mathematical Structures in Computer Science*, 15:709–772, 2005.
6. 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.
7. 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.
8. S. Ishtiaq and P. W. O’Hearn. BI as an assertion language for mutable data structures. In *Proc. of POPL’01*, pages 14–26, London, UK, 2001.

9. I. Mijajlović, N. Torp-Smith, and P. O’Hearn. Refinement and separation context. In *Proc. of FSTTCS’04*, pages 421–433, Chennai, India, 2004.
10. I. Mijajlović and H. Yang. Data refinements with low-level pointer operations. In *Proc. of APLAS’05*, pages 19–36, Tsukuba, Japan, 2005.
11. A. Nanevski, A. Ahmed, G. Morrisett, and L. Birkedal. Abstract predicates and mutable adts in hoare type theory. TR-14-06, Harvard University, 2006.
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. U. Reddy and H. Yang. Correctness of data representations involving heap data structures. *Science of Computer Programming*, 50(1):129–160, 2004.
18. J. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
19. J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In J. Davies, B. Roscoe, and J. Woodcock, editors, *Millennial Perspectives in Computer Science*, pages 303–321. Palgrave, Houndsmill, Hampshire, 2000.
20. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS’02*, pages 55–74, Copenhagen, Denmark, 2002.
21. M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342:28–55, 2005.
22. N. Torp-Smith. *Advances in Separation Logic — A Study of Logics for Reasoning about Stateful Programs*. PhD thesis, IT University of Copenhagen, 2005.
23. H. Yang. Relational separation logic. *Theoretical Comput. Sci.*, 2005. (to appear).

A Typing Rules for Expressions E and Terms M

$$\begin{array}{c}
\frac{}{\Delta, i : \delta \vdash i : \delta} \quad \frac{}{\Delta \vdash \text{nil} : \text{ref}} \\
\frac{\Delta \vdash E : \text{int} \quad \Delta \vdash F : \text{int}}{\Delta \vdash E + F : \text{int}} \quad \frac{\Delta \vdash E : \text{int} \quad \Delta \vdash F : \text{int}}{\Delta \vdash E - F : \text{int}} \quad \frac{}{\Delta \vdash n : \text{int}} \\
\frac{}{\Delta \mid \Gamma, x : \tau \vdash x : \tau} \quad \frac{\Delta, i : \delta \mid \Gamma \vdash M : \tau}{\Delta \mid \Gamma \vdash \lambda i : \delta. M : \delta \rightarrow \tau} \\
\frac{\Delta \mid \Gamma \vdash M : \delta \rightarrow \tau \quad \Delta \vdash E : \delta}{\Delta \mid \Gamma \vdash M E : \tau} \quad \frac{\Delta \mid \Gamma, x : \tau \vdash M : \tau'}{\Delta \mid \Gamma \vdash \lambda x : \tau. M : \tau \rightarrow \tau'} \\
\frac{\Delta \mid \Gamma \vdash M : \tau' \rightarrow \tau \quad \Delta \mid \Gamma \vdash N : \tau'}{\Delta \mid \Gamma \vdash M N : \tau} \quad \frac{\Delta \mid \Gamma \vdash M : \tau \rightarrow \tau}{\Delta \mid \Gamma \vdash \text{fix } M : \tau} \\
\frac{\Delta \vdash E : \delta \quad \Delta \vdash F : \delta \quad \Delta \mid \Gamma \vdash M : \text{com} \quad \Delta \mid \Gamma \vdash N : \text{com}}{\Delta \mid \Gamma \vdash \text{if } (E=F) M N : \text{com}} \\
\frac{\Delta \mid \Gamma \vdash M : \text{com} \quad \Delta \mid \Gamma \vdash N : \text{com}}{\Delta \mid \Gamma \vdash M; N : \text{com}} \quad \frac{\Delta, i : \text{ref} \mid \Gamma \vdash M : \text{com}}{\Delta \mid \Gamma \vdash \text{let } i = \text{new in } M : \text{com}} \\
\frac{\Delta \vdash E : \text{ref}}{\Delta \mid \Gamma \vdash \text{free}(E) : \text{com}} \quad \frac{\Delta, i : \text{int} \mid \Gamma \vdash M : \text{com} \quad \Delta \vdash E : \text{ref}}{\Delta \mid \Gamma \vdash \text{let } i = E.0 \text{ in } M : \text{com}} \\
\frac{\Delta, i : \text{ref} \mid \Gamma \vdash M : \text{com} \quad \Delta \vdash E : \text{ref}}{\Delta \mid \Gamma \vdash \text{let } i = E.f \text{ in } M : \text{com}} \quad f \in \{1, 2\} \quad \frac{\Delta \vdash E : \text{ref} \quad \Delta \vdash F : \text{int}}{\Delta \mid \Gamma \vdash E.0 := F : \text{com}} \\
\frac{\Delta \vdash E : \text{ref} \quad \Delta \vdash F : \text{ref}}{\Delta \mid \Gamma \vdash E.f := F : \text{com}} \quad f \in \{1, 2\}
\end{array}$$

B Semantics of Assertions

$$\begin{array}{l}
\llbracket E = F \rrbracket_\rho \stackrel{\text{def}}{=} \text{if } (\llbracket E \rrbracket_\rho = \llbracket F \rrbracket_\rho) \text{ then } \text{Heap} \text{ else } \emptyset \\
\llbracket E \leq F \rrbracket_\rho \stackrel{\text{def}}{=} \text{if } (\llbracket E \rrbracket_\rho = \llbracket F \rrbracket_\rho) \text{ then } \text{Heap} \text{ else } \emptyset \\
\llbracket E \mapsto E_0, E_1, E_2 \rrbracket_\rho \stackrel{\text{def}}{=} \{h \mid \text{dom}(h) = \{\llbracket E \rrbracket_\rho\} \wedge h(\llbracket E \rrbracket_\rho) = (\llbracket E_0 \rrbracket_\rho, \llbracket E_1 \rrbracket_\rho, \llbracket E_2 \rrbracket_\rho)\} \\
\llbracket \text{emp} \rrbracket_\rho \stackrel{\text{def}}{=} \{h \mid \text{dom}(h) = \emptyset\} \\
\llbracket P * Q \rrbracket_\rho \stackrel{\text{def}}{=} \{h \mid \exists h_1, h_2. h = h_1 \bullet h_2 \wedge h_1 \models \llbracket P \rrbracket_\rho \wedge h_2 \models \llbracket Q \rrbracket_\rho\} \\
\llbracket P \wedge Q \rrbracket_\rho \stackrel{\text{def}}{=} \llbracket P \rrbracket_\rho \cap \llbracket Q \rrbracket_\rho \\
\llbracket \neg P \rrbracket_\rho \stackrel{\text{def}}{=} \text{Heap} - \llbracket P \rrbracket_\rho \\
\llbracket \exists i : \delta. P \rrbracket_\rho \stackrel{\text{def}}{=} \bigcup \{\llbracket P \rrbracket_{\rho[i \rightarrow v]} \mid v \in \llbracket \delta \rrbracket\}
\end{array}$$

C Proof of Theorem 1

Lemma 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 $(\rho, \eta_0, \eta_1, r)$.

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, $(\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho)) \models \varphi$. 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:

$$\begin{aligned}
& (\rho, \eta_0, \eta_1, r) \models \{P\}M\{Q\} \otimes P_0 \\
\iff & (\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P_0 \rrbracket_\rho)) \models \{P\}M\{Q\} \quad (\text{by the semantics of } \otimes P) \\
\iff & [\mathbf{Eq}(\llbracket P \rrbracket_\rho) * \mathbf{Eq}(\llbracket P_0 \rrbracket_\rho) * r](\llbracket M \rrbracket_{\rho, \eta_0}, \llbracket M \rrbracket_{\rho, \eta_1})[\mathbf{Eq}(\llbracket Q \rrbracket_\rho) * \mathbf{Eq}(\llbracket P_0 \rrbracket_\rho) * r] \\
\iff & [\mathbf{Eq}(\llbracket P * P_0 \rrbracket_\rho) * r](\llbracket M \rrbracket_{\rho, \eta_0}, \llbracket M \rrbracket_{\rho, \eta_1})[\mathbf{Eq}(\llbracket Q * P_0 \rrbracket_\rho) * r] \\
\iff & (\rho, \eta_0, \eta_1, r) \models \{P * P_0\}M\{Q * P_0\} \quad (\text{by the semantics of triples}).
\end{aligned}$$

The second equivalence is by the semantics of triples, and the third equivalence holds because \mathbf{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 $(\rho, \eta_0, \eta_1, r)$. 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

$$(\rho, \eta_0, \eta_1, r) \models \varphi \iff (\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho)) \models \varphi \iff (\rho, \eta_0, \eta_1, r) \models \varphi \otimes P.$$

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:

$$\begin{aligned}
& (\rho, \eta_0, \eta_1, r) \models \exists i: \delta. \varphi \otimes P \\
\iff & \text{there exists } v \in \llbracket \delta \rrbracket \text{ s.t. } (\rho[i \mapsto v], \eta_0, \eta_1, r) \models \varphi \otimes P \\
\iff & \text{there exists } v \in \llbracket \delta \rrbracket \text{ s.t. } (\rho[i \mapsto v], \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_{\rho[i \mapsto v]})) \models \varphi \\
\iff & \text{there exists } v \in \llbracket \delta \rrbracket \text{ s.t. } (\rho[i \mapsto v], \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho)) \models \varphi \quad (\text{since } i \notin \text{fv}(P)) \\
\iff & (\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho)) \models \exists i: \delta. \varphi \\
\iff & (\rho, \eta_0, \eta_1, r) \models (\exists i: \delta. \varphi) \otimes P.
\end{aligned}$$

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:

$$\begin{aligned}
& (\rho, \eta_0, \eta_1, r) \models (\varphi \Rightarrow \psi) \otimes P \\
\iff & (\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho)) \models \varphi \Rightarrow \psi \\
\iff & \forall s. ((\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho) * s) \models \varphi) \implies ((\rho, \eta_0, \eta_1, r * \mathbf{Eq}(\llbracket P \rrbracket_\rho) * s) \models \psi) \\
\iff & \forall s. ((\rho, \eta_0, \eta_1, r * s) \models \varphi \otimes P) \implies ((\rho, \eta_0, \eta_1, r * s) \models \psi \otimes P) \\
\iff & (\rho, \eta_0, \eta_1, r) \models (\varphi \otimes P) \Rightarrow (\psi \otimes P).
\end{aligned}$$

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

Theorem 1. *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 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 $(\rho, \eta_0, \eta_1, r)$.

The first case is the rule for memory allocation:

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

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.,

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

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

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

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

$$\llbracket \text{let } i = \text{new in } M \rrbracket_{\rho, \eta_j}(h_j, k_j) = \llbracket M \rrbracket_{\rho[i \mapsto l], \eta_j}(h_j \bullet [l \mapsto 0, \text{nil}, \text{nil}], k_j). \quad (2)$$

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

$$\begin{aligned} & h_0[\text{Eq}(\llbracket P \rrbracket_\rho) * r * s]h_1 \wedge k_0[(\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s) \rightarrow \text{Eq}(G)]k_1 \\ \Rightarrow & h_0[\text{Eq}(\llbracket P \rrbracket_{\rho'}) * r * s]h_1 \wedge k_0[(\text{Eq}(\llbracket Q \rrbracket_{\rho'}) * r * s) \rightarrow \text{Eq}(G)]k_1 \\ \Rightarrow & h'_0[\text{Eq}(\llbracket P * i \mapsto 0, \text{nil}, \text{nil} \rrbracket_{\rho'}) * r * s]h'_1 \wedge k_0[(\text{Eq}(\llbracket Q \rrbracket_{\rho'}) * r * s) \rightarrow \text{Eq}(G)]k_1 \\ \Rightarrow & \llbracket M \rrbracket_{\rho', \eta_0}(h'_0, k_0)[\text{Eq}(G)]\llbracket M \rrbracket_{\rho', \eta_1}(h'_1, k_1) \\ \Rightarrow & \llbracket \text{let } i = \text{new in } M \rrbracket_{\rho, \eta_0}(h_0, k_0)[\text{Eq}(G)]\llbracket \text{let } i = \text{new in } M \rrbracket_{\rho, \eta_1}(h_1, k_1). \end{aligned}$$

The first implication holds, because ρ and ρ' are different only for i but $i \notin \text{fv}(P, Q)$. The second implication follows from the definition of h'_j , and the third implication from the assumption that $(\rho, \eta_0, \eta_1, r) \models \forall i. \{P * i \mapsto 0, \text{nil}, \text{nil}\} M\{Q\}$. Finally, the last implication holds, because of the equation 2.

The second case is the axiom for lookup

$$(\forall i. \{P * E \mapsto i, E_0, E_1\} M\{Q\}) \Rightarrow \{\exists i. P * E \mapsto i, E_0, E_1\} \text{let } i = E.0 \text{ in } M\{Q\}.$$

Consider $(\rho, \eta_0, \eta_1, r)$ that satisfies $(\forall i. \{P * E \mapsto i, E_0, E_1\} M\{Q\})$, and pick arbitrary $h_0, h_1 \in \text{Heap}$, $k_0, k_1 \in \text{cont}$ and $s \in \mathcal{R}$ such that

$$h_0[\text{Eq}(\llbracket \exists i. P * E \mapsto i, E_0, E_1 \rrbracket_\rho) * r * s]h_1 \wedge k_0[\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]k_1.$$

Let l be $\llbracket E \rrbracket_\rho$. By the first conjunct above, l is in $\text{dom}(h_0) \cap \text{dom}(h_1)$, and there exist v and ρ' such that

$$v = \text{proj}_0(h_0(l)) = \text{proj}_0(h_1(l)), \quad \rho' = \rho[i \mapsto v], \quad \text{and} \quad h_0[\text{Eq}(\llbracket P * E \mapsto i, E_0, E_1 \rrbracket_{\rho'}) * r * s] h_1.$$

The two equalities above about v and ρ' imply that for $j = 0, 1$,

$$\llbracket \text{let } i = E.0 \text{ in } M \rrbracket_{\rho, \eta_j}(h_j, k_j) = \llbracket M \rrbracket_{\rho', \eta_j}(h_j, k_j). \quad (3)$$

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

$$\begin{aligned} & k_0[\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)] k_1 \wedge h_0[\text{Eq}(\llbracket P * E \mapsto i, E_0, E_1 \rrbracket_{\rho'}) * r * s] h_1 \\ \implies & k_0[\text{Eq}(\llbracket Q \rrbracket_{\rho'}) * r * s \rightarrow \text{Eq}(G)] k_1 \wedge h_0[\text{Eq}(\llbracket P * E \mapsto i, E_0, E_1 \rrbracket_{\rho'}) * r * s] h_1 \\ \implies & \llbracket M \rrbracket_{\rho', \eta_0}(h_0, k_0) [\text{Eq}(G)] \llbracket M \rrbracket_{\rho', \eta_1}(h_1, k_1) \\ \implies & \llbracket \text{let } i = E.0 \text{ in } M \rrbracket_{\rho, \eta_0}(h_0, k_0) [\text{Eq}(G)] \llbracket \text{let } i = E.0 \text{ in } M \rrbracket_{\rho, \eta_1}(h_1, k_1). \end{aligned}$$

The first implication holds because $i \notin \text{fv}(Q)$, 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 3.

The third case is the axiom $\{E \mapsto E_0, E_1, E_2\} \text{free}(E) \{\text{emp}\}$. Choose arbitrary $(\rho, \eta_0, \eta_1, r)$, $h_0, h_1 \in \text{Heap}$, $k_0, k_1 \in \text{cont}$, and $s \in \mathcal{R}$, such that

$$h_0[\text{Eq}(\llbracket E \mapsto E_0, E_1, E_2 \rrbracket_\rho) * r * s] h_1 \wedge k_0[\text{Eq}(\llbracket \text{emp} \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)] k_1.$$

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 $m_0[\text{Eq}(\llbracket F \mapsto E_0, E_1, E_2 \rrbracket)] m_1$ and $n_0[r * s] n_1$. Note that the relationship between m_0 and m_1 implies that $\llbracket \text{free}(E) \rrbracket_{\rho, \eta_j}(h_j, k_j) = k_j(n_j)$ for $j = 0, 1$. Thus, it is sufficient to show that $k_0(n_0)[\text{Eq}(G)] k_1(n_1)$. Note that n_0 and n_1 are already related by $r * s$, and k_0 and k_1 by $\text{Eq}(\llbracket \text{emp} \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)$. The conclusion follows from these two relationships, because $\text{Eq}(\llbracket \text{emp} \rrbracket_\rho) * r * s = r * s$.

The fourth case is the axiom $\{F \mapsto E_0, E_1, E_2\} F.0 := F' \{F \mapsto F', E_1, E_2\}$. Choose arbitrary $(\rho, \eta_0, \eta_1, r)$, $h_0, h_1 \in \text{Heap}$, $k_0, k_1 \in \text{cont}$, and $s \in \mathcal{R}$, such that

$$h_0[\text{Eq}(\llbracket F \mapsto E_0, E_1, E_2 \rrbracket_\rho) * r * s] h_1 \wedge k_0[\text{Eq}(\llbracket F \mapsto F', E_1, E_2 \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)] k_1.$$

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 $m_0[\text{Eq}(\llbracket F \mapsto E_0, E_1, E_2 \rrbracket_\rho)] m_1$ and $n_0[r * s] n_1$. Let m' be the heap $\llbracket \llbracket F \rrbracket_\rho \rightarrow (\llbracket F' \rrbracket_\rho, \llbracket E_1 \rrbracket_\rho, \llbracket E_2 \rrbracket_\rho) \rrbracket$. Then, we have the following two facts:

1. $(m' \bullet n_0) [\text{Eq}(\llbracket F \mapsto F', E_1, E_2 \rrbracket_\rho) * r * s] (m' \bullet n_1)$, and
2. for all $j = 0, 1$, $\llbracket F.0 := F' \rrbracket_{\rho, \eta_j}(h_j, k_j) = k_j(m' \bullet n_j)$.

By the first fact, $k_0(m' \bullet n_0) [\text{Eq}(G)] k_1(m' \bullet n_1)$. Now, the second fact gives the required $\llbracket F.0 := F' \rrbracket_{\rho, \eta_0}(h_0, k_0) [\text{Eq}(G)] \llbracket F.0 := F' \rrbracket_{\rho, \eta_1}(h_1, k_1)$.

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

$$h_0[\text{Eq}(\llbracket P \rrbracket_\rho) * r * s] h_1 \wedge k_0[\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)] k_1.$$

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

$$\begin{aligned} \text{Eq}(\llbracket P \rrbracket_\rho) * r * s &\subseteq \text{Eq}(\llbracket P' \rrbracket_\rho) * r * s, \quad \text{and} \\ \llbracket \text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G) \rrbracket &\subseteq \llbracket \text{Eq}(\llbracket Q' \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G) \rrbracket. \end{aligned}$$

Thus, $h_0[\text{Eq}(\llbracket P \rrbracket_\rho) * r * s]h_1$ and $k_0[\text{Eq}(\llbracket Q' \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]k_1$. These two relationships imply the required $\llbracket M \rrbracket_{\rho, \eta_0}(h_0, k_0)[\text{Eq}(G)]\llbracket M \rrbracket_{\rho, \eta_1}(h_1, k_1)$, 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.,

$$\llbracket \text{Eq}(\llbracket \exists i. P \rrbracket_\rho) * r \rrbracket (\llbracket M \rrbracket_{\rho, \eta_0}, \llbracket M \rrbracket_{\rho, \eta_1}) [\text{Eq}(\llbracket \exists i. Q \rrbracket_\rho) * r].$$

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

$$h_0[\text{Eq}(\llbracket \exists i. P \rrbracket_\rho) * r * s]h_1 \quad \text{and} \quad k_0[(\text{Eq}(\llbracket \exists i. Q \rrbracket_\rho) * r * s) \rightarrow \text{Eq}(G)]k_1.$$

By the definition of Eq , $\llbracket \exists i. P \rrbracket$ and $\llbracket \exists i. Q \rrbracket$, these two conjuncts imply the existence of v and ρ' such that

$$\rho' = \rho[i \rightarrow v], \quad h_0[\text{Eq}(\llbracket P \rrbracket_{\rho'}) * r * s]h_1, \quad \text{and} \quad k_0[(\text{Eq}(\llbracket Q \rrbracket_{\rho'}) * r * s) \rightarrow \text{Eq}(G)]k_1.$$

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

$$\begin{aligned} &(h_0[\text{Eq}(\llbracket P \rrbracket_{\rho'}) * r * s]h_1) \wedge (k_0[(\text{Eq}(\llbracket Q \rrbracket_{\rho'}) * r * s) \rightarrow \text{Eq}(G)]k_1) \\ \Rightarrow &\llbracket M \rrbracket_{\rho', \eta_0}(h_0, k_0)[\text{Eq}(G)]\llbracket M \rrbracket_{\rho', \eta_1}(h_1, k_1) \quad (\text{since } (\rho, \eta_0, \eta_1, r) \models \forall i. \{P\}M\{Q\}) \\ \Rightarrow &\llbracket M \rrbracket_{\rho, \eta_0}(h_0, k_0)[\text{Eq}(G)]\llbracket M \rrbracket_{\rho, \eta_1}(h_1, k_1) \quad (\text{since } i \notin \text{fv}(M)). \end{aligned}$$

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 $h_0, h_1 \in \text{Heap}$, $s \in \mathcal{R}$, and $k_0, k_1 \in \text{cont}$, such that

$$h_0[\text{Eq}(\llbracket P \vee P' \rrbracket_\rho) * r * s]h_1 \quad \wedge \quad k_0[\text{Eq}(\llbracket Q \vee Q' \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]k_1.$$

By the definition of $\text{Eq}(\llbracket P \vee P' \rrbracket_\rho)$, heaps h_0 and h_1 are related by $\text{Eq}(\llbracket P \rrbracket_\rho) * r * s$ or $\text{Eq}(\llbracket P' \rrbracket_\rho) * r * s$. Without loss of generality, we assume that

$$h_0[\text{Eq}(\llbracket P \rrbracket_\rho) * r * s]h_1. \tag{4}$$

Since Eq is monotone and $*$ preserves the subset order for relations, relation $\text{Eq}(\llbracket Q \vee Q' \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)$ is a subset of $\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)$, and so,

$$k_0[\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]k_1. \tag{5}$$

By the supposition, $(\rho, \eta_0, \eta_1, r)$ satisfies $\{P\}M\{Q\}$. Thus, the relationships 4 and 5 imply the required

$$\llbracket M \rrbracket_{\rho, \eta_0}(h_0, k_0)[\text{Eq}(G)]\llbracket M \rrbracket_{\rho, \eta_1}(h_1, k_1).$$

The eighth 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 $h_0, h_1 \in \text{Heap}$, $s \in \mathcal{R}$, and $k_0, k_1 \in \text{cont}$, such that

$$h_0[\text{Eq}(\llbracket P \rrbracket_\rho) * r * s]h_1 \wedge k_0[\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]k_1.$$

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

$$k'_0 = \lambda h'_0. \llbracket N \rrbracket_{\rho, \eta_0}(h'_0, k_0)[\text{Eq}(\llbracket P_0 \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]\lambda h'_1. \llbracket N \rrbracket_{\rho, \eta_1}(h'_1, k_1) = k'_1.$$

Since $(\rho, \eta_0, \eta_1, r) \models \{P\}M\{Q_0\}$, the above relationship between k'_0 and k'_1 implies $\llbracket M \rrbracket_{\rho, \eta_0}(h_0, k'_0)[\text{Eq}(G)]\llbracket M \rrbracket_{\rho, \eta_1}(h_1, k'_1)$. This gives the conclusion, because $\llbracket M; N \rrbracket_{\rho, \eta_j}(h_j, k_j)$ is equal to $\llbracket M \rrbracket_{\rho, \eta_j}(h_j, k'_j)$, for all $j = 0, 1$.

The ninth 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 $h_0, h_1 \in \text{Heap}$, $s \in \mathcal{R}$, and $k_0, k_1 \in \text{cont}$, such that

$$h_0[\text{Eq}(\llbracket P \rrbracket_\rho) * r * s]h_1 \wedge k_0[\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]k_1.$$

We do the case analysis depending on whether $\llbracket E \rrbracket_\rho = \llbracket F \rrbracket_\rho$. Suppose that $\llbracket E \rrbracket_\rho = \llbracket F \rrbracket_\rho$. In this case, $h_0[\text{Eq}(\llbracket P \wedge E = F \rrbracket_\rho) * r * s]h_1$, and

$$\llbracket \text{if}(E=F)MN \rrbracket_{\rho, \eta_j}(h_j, k_j) = \llbracket M \rrbracket_{\rho, \eta_j}(h_j, k_j) \text{ for all } j = 0, 1. \quad (6)$$

Using these facts, we derive the conclusion as follows:

$$\begin{aligned} & h_0[\text{Eq}(\llbracket P \wedge E = F \rrbracket_\rho) * r * s]h_1 \wedge k_0[\text{Eq}(\llbracket Q \rrbracket_\rho) * r * s \rightarrow \text{Eq}(G)]k_1 \\ \implies & \llbracket M \rrbracket_{\rho, \eta_0}(h_0, k_0)[\text{Eq}(G)]\llbracket M \rrbracket_{\rho, \eta_1}(h_1, k_1) \\ \implies & \llbracket \text{if}(E=F)MN \rrbracket_{\rho, \eta_0}(h_0, k_0)[\text{Eq}(G)]\llbracket \text{if}(E=F)MN \rrbracket_{\rho, \eta_1}(h_1, k_1). \end{aligned}$$

The first implication follows from our assumption that $(\rho, \eta_0, \eta_1, r)$ satisfies $\{P \wedge E = F\}M\{Q\}$, and the second follows from the equation 6 above. The other case that $\llbracket E \rrbracket_\rho \neq \llbracket F \rrbracket_\rho$ can be proved similarly, so it is omitted here.

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

1. For all ρ, η , if $\eta(k) = \perp$, then $\llbracket C(k) \rrbracket_{\rho, \eta} = \perp$.
2. For all $(\rho, \eta_0, \eta_1, r)$, the following set is admissible:

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

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. \square

D Proofs of Propositions 1 and 2

Proposition 1. *For all $p, q \subseteq \text{Heap}$ and all $c \in \text{Heap} \rightarrow (\text{Heap} \times D + \{\text{err}\})_\perp$, the quadruple $[\text{Eq}(p)](\text{cps}(c), \text{cps}(c))[\text{Eq}(q)]$ holds if and only if the following two conditions 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 $c(h \bullet h_1)$.

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

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

Then, $k[\text{Eq}(q) \rightarrow \text{Eq}(G)]k$ and $h[\text{Eq}(p)]h$. By the assumption on the validity of the quadruple, $\text{cps}(c)(h, k)[\text{Eq}(G)]\text{cps}(c)(h, k)$. By the definition of k , this relationship on $\text{cps}(c)$ implies that $\text{cps}(c)(h, k) = \perp$, which in turn gives $(c(h) = \perp) \vee (c(h) \in \text{Heap} \wedge k(c(h)) = \perp)$. 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:

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

By the definition of r and k_i , we have that

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

To see why the third relationship holds, note that if $h'_1[\text{Eq}(q) * r]h'_2$, then $h'_1 \bullet h_1$ is defined and $h'_2 = h'_1 \bullet h_1$. Thus, $h'_1 = c(h)$ holds precisely when $c(h) \in \text{Heap} \wedge h'_2 = c(h) \bullet h_1$ holds. This implies that $k_1(h'_1) = \textit{normal}$ iff $k_2(h'_2) = \textit{normal}$. Now, by the assumption on the validity of the quadruple, we have that

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

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

$$h_1[\text{Eq}(p) * r]h_2 \quad \text{and} \quad k_1[\text{Eq}(q) * r \rightarrow \text{Eq}(G)]k_2.$$

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 $c(h_1) = \perp$, then $c(h'_1) = \perp$ by the condition (2-b) of the assumption, and $c(h'_2) = \perp$ by the condition (2-a) of the assumption. Thus, in this case, we have $\text{cps}(c)(h_1, k_1) = \text{cps}(c)(h_2, k_2) = \perp$ and $\text{cps}(c)(h_1, k_1)[\text{Eq}(G)]\text{cps}(c)(h_2, k_2)$, as desired. Otherwise, i.e., if $c(h_1) \neq \perp$, then $c(h'_1) \neq \perp$ 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),

$$c(h_1) = c(h'_1) \bullet h''_1[\text{Eq}(q) * r]c(h'_1) \bullet h''_2 = c(h_2).$$

This implies $\text{cps}(c)(h_1, k_1)[\text{Eq}(G)]\text{cps}(c)(h_2, k_2)$, as desired. \square

Proposition 2. *Let r, s be relations in \mathcal{R} , and let c_1, c_2 be functions of type $\text{Heap} \rightarrow (\text{Heap} \times D + \{\text{err}\})_{\perp}$. A quadruple $[r](\text{cps}(c_1), \text{cps}(c_2))[s]$ holds, iff*

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

Proof. (\implies) Consider r', h_1, h_2 such that $h_1[r * r']h_2$. We first show that $c_1(h_1) = \perp \iff c_2(h_2) = \perp$. Let k be the continuation $\lambda h'. \text{normal}$. Then, $k[s * r' \rightarrow \text{Eq}(G)]k$. By the assumption on the quadruple for $\text{cps}(c_1), \text{cps}(c_2)$, we have that

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

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

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

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

Since both $c_1(h_1)$ and $c_2(h_2)$ are different from \perp , the relationship is equivalent to $k_1(c_1(h_1))[\text{Eq}(G)]k_2(c_2(h_2))$. We have just shown that $c_1(h_1)[(s * r')^{\perp\perp}]c_2(h_2)$.

(\impliedby) Pick arbitrary relation r' , heaps h_1, h_2 and continuations k_1, k_2 such that $h_1[r * r']h_2$ and $k_1[s * r' \rightarrow \text{Eq}(G)]k_2$ (i.e., $k_1[(s * r')^{\perp}]k_2$.) By the assumption of the if direction, either $c_1(h_1) = c_2(h_2) = \perp$ or $c_1(h_1)[(s * r')^{\perp\perp}]c_2(h_2)$. In the first case,

$$\text{cps}(c_1)(h_1, k_1) = \perp [\text{Eq}(G)] \perp = \text{cps}(c_2)(h_2, k_2),$$

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

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

The conclusion follows from these two relationships. \square