Relational Parametricity and Separation Logic[⋆]

Lars Birkedal[1] and Hongseok Yang[2]

[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 [16, 11, 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 [12, 7] it was shown how to extend separation logic first to languages with a simple kind of procedures [12] and then to languages also with higher-types [7]. Moreover, in [12] 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 [12] 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

[⋆] This work was supported by FUR (FIRST). Yang was supported also by EPSRC. We would like to thank Nick Benton for his insightful comments.

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 [12] 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] 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 [13]. 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 [15].

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

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 omitted; they can be found in the full version of the paper.[4]

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 [18, 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 [8, 9]. 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 [8], 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 [9].

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 [3], 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

be maintained; if we use quantification without any restrictions, as in [2], it appears that we cannot have the usual reading of assertions as sets of heaps because, then, the rule of consequence is not sound.

[4] The full version is available at the following URL: `http://www.dcs.qmul.ac.uk/~hyang/paper/fossacs07-full.pdf`.

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 stack variables. The types and terms of the languages are defined as follows:

$$\text{Types } \tau ::= \mathsf{com} \mid \mathsf{ref} \to \tau \mid \tau \to \tau \qquad \text{Expressions } E ::= i \mid \mathsf{nil}$$
$$\begin{aligned}\text{Terms } M ::= {} & x \mid \lambda i.\, M \mid M\ E \mid \lambda x{:}\,\tau.\, M \mid M\, M \mid \mathsf{fix}\, M \mid \mathsf{if}\, (E{=}E)\, M\, M \mid M; M \\ \mid {} & \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M \mid \mathsf{free}(E) \mid \mathsf{let}\ i{=}[E]\ \mathsf{in}\ M \mid [E]{:=}E\end{aligned}$$

The language separates expressions E from terms M. Expressions denote heap-independent reference values, and they 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, $\mathsf{ref} \to \tau$ 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 one field for references. Command $\mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M$ allocates such a unary heap cell, binds the address of the cell to i, and runs M under this binding. The content of this newly allocated cell at address i is read by $\mathsf{let}\ j = [i]\ \mathsf{in}\ N$ and updated by $[i] := E$. The cell i is deallocated by $\mathsf{free}(i)$.

The language uses typing judgments of the form $\Delta \vdash E(:\mathsf{ref})$ and $\Delta \mid \Gamma \vdash M : \tau$, where Δ is a finite set of stack variables and Γ is a standard type envi-

$$\frac{}{\Delta, i \vdash i} \qquad \frac{}{\Delta \vdash \mathsf{nil}}$$

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

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

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

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

$$\frac{\Delta \vdash E}{\Delta \mid \Gamma \vdash \mathsf{free}(E) : \mathsf{com}} \quad \frac{\Delta, i \mid \Gamma \vdash M : \mathsf{com} \quad \Delta \vdash E}{\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}[E]\ \mathsf{in}\ M : \mathsf{com}} \quad \frac{\Delta \vdash E \quad \Delta \vdash F}{\Delta \mid \Gamma \vdash E := F : \mathsf{com}}$$

Fig. 1. Typing Rules for Expressions and Terms

ronment for identifiers x. The typing rules for expressions and terms are shown in Figure 1.

We use the standard assertions from separation logic to describe properties of the heap:[5] $P ::= E = E \mid E \leq E \mid E \mapsto E \mid \mathsf{emp} \mid P{*}P \mid P{\wedge}P \mid \neg P \mid \exists i.\, P$. The points-to predicate $E \mapsto E'$ means that the current heap has only one cell at address E and that the content of the cell is E'. 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.

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.

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:

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

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, $(\{P\}x\{Q\} \Rightarrow \{P'\}M(x)\{Q'\}) \otimes P_0$

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

Proof Rules for Hoare Triples

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

$$\{E \mapsto F\}\mathsf{free}(E)\{\mathsf{emp}\} \qquad \{E \mapsto E'\}[E] := F\{E \mapsto F\}$$

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

Proof Rules for Invariant Extension $- \otimes P$

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

Rule for Fixed-Point Induction

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

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

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

Fig. 2. Sample Proof Rules

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 2 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. 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 φ, and the other rules distribute this added invariant all the way down to the triples. 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 $\perp$, 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 $\mathsf{perm} \times A \to A$, 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.,

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

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 $a \sqsubseteq b \Longrightarrow \pi \cdot a \sqsubseteq \pi \cdot b$ 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 3. In the figure, we use the FM-cpo *ref* of references defined by: $\mathit{ref} \stackrel{\mathit{def}}{=} \mathit{Loc} + \{\mathit{nil}\}$ with $\pi \cdot v \stackrel{\mathit{def}}{=} \mathsf{if}\ (v = \mathit{nil})\ \mathsf{then}\ \mathit{nil}\ \mathsf{else}\ \pi(v)$. The only nonstandard part is the semantics of the command type com, which we define in the continuation passing style following [17, 3]:

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

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:

$$\begin{array}{lll} \mathit{ref} \overset{\mathit{def}}{=} \mathit{Loc} + \{\mathit{nil}\} & [\![\mathsf{ref} \to \tau]\!] \overset{\mathit{def}}{=} \mathit{ref} \to [\![\tau]\!] & [\![\tau \to \tau']\!] \overset{\mathit{def}}{=} [\![\tau]\!] \to [\![\tau']\!] \\ [\![\mathsf{com}]\!] \overset{\mathit{def}}{=} \mathit{Heap} \times \mathit{cont} \to O & \multicolumn{2}{l}{(\text{where } O = \{\mathit{normal}, \mathit{err}\}_\perp \text{ and } \mathit{cont} = \mathit{Heap} \to O)} \\ [\![\varDelta]\!] \overset{\mathit{def}}{=} \prod_{i \in \varDelta} \mathit{ref} & [\![\varGamma]\!] \overset{\mathit{def}}{=} \prod_{x:\tau \in \varGamma} [\![\tau]\!]. & \end{array}$$

Fig. 3. Interpretation of Types and Typing Contexts

$$[\![\varDelta \vdash E]\!] : [\![\varDelta]\!] \to \mathit{ref} \qquad [\![\varDelta, i \vdash i]\!]_\rho \overset{\mathit{def}}{=} \rho(i) \qquad [\![\varDelta \vdash \mathsf{nil}]\!]_\rho \overset{\mathit{def}}{=} \mathit{nil}$$

Fig. 4. Interpretation of Expressions

1. $f \sqsubseteq g \overset{\mathit{def}}{\Longleftrightarrow} \mathsf{dom}(f) = \mathsf{dom}(g)$ and $f(a) \sqsubseteq g(a)$ for all $a \in \mathsf{dom}(f)$,
2. $(\pi \cdot f)(a) \overset{\mathit{def}}{=} \mathsf{if}\ (a \in \pi(\mathsf{dom}(f)))\ \mathsf{then}\ (\pi \cdot ((f \circ \pi^{-1})(a)))\ \mathsf{else}$ 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 contains a reference. 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 $\varDelta$ and $\varGamma$ is given by cartesian products: $[\![\varDelta]\!] \overset{\mathit{def}}{=} \prod_{i \in \varDelta} \mathit{ref}$ and $[\![\varGamma]\!] \overset{\mathit{def}}{=} \prod_{x:\tau \in \varGamma} [\![\tau]\!]$. The products here are taken over finite families, so they give well-defined FM-cpos.[6] We will use symbols ρ and η to denote environments in $[\![\varDelta]\!]$ and $[\![\varGamma]\!]$, respectively.

The semantics of expressions and terms is shown in Figures 4 and 5. 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 $\mathsf{supp}(h, \eta, \rho, k)$. We borrowed this interpretation from Benton and Leperchey [3].

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

$$[\![\Delta \mid \Gamma \vdash M{:}\,\tau]\!] \;:\; [\![\Delta]\!] \times [\![\Gamma]\!] \to [\![\tau]\!]$$

$$\begin{aligned}
[\![\Delta \mid \Gamma, x{:}\,\tau \vdash x{:}\,\tau]\!]_{\rho,\eta} &\overset{def}{=} \eta(x)\\
[\![\Delta \mid \Gamma \vdash \lambda i.\, M{:}\,\mathsf{ref} \to \tau]\!]_{\rho,\eta} &\overset{def}{=} \lambda v{:}\,\mathit{ref}.\, [\![\Delta, i \mid \Gamma \vdash M{:}\,\tau]\!]_{\rho[i\to v],\eta}\\
[\![\Delta \mid \Gamma \vdash M\ E{:}\,\tau]\!]_{\rho,\eta} &\overset{def}{=} ([\![\Delta \mid \Gamma \vdash M{:}\,\mathsf{ref} \to \tau]\!]_{\rho,\eta})\ [\![E]\!]_{\rho}\\
[\![\Delta \mid \Gamma \vdash \lambda x{:}\,\tau'.\, M{:}\,\tau' \to \tau]\!]_{\rho,\eta} &\overset{def}{=} \lambda m{:}\,[\![\tau']\!].\, [\![\Delta \mid \Gamma, x{:}\,\tau' \vdash M{:}\,\tau]\!]_{\rho,\eta[x\to m]}\\
[\![\Delta \mid \Gamma \vdash M\ N{:}\,\tau]\!]_{\rho,\eta} &\overset{def}{=} ([\![\Delta \mid \Gamma \vdash M{:}\,\tau' \to \tau]\!]_{\rho,\eta})\ [\![\Delta \mid \Gamma \vdash N{:}\,\tau']\!]_{\rho,\eta}\\
[\![\Delta \mid \Gamma \vdash \mathsf{fix}\ M{:}\,\tau]\!]_{\rho,\eta} &\overset{def}{=} \mathit{leastfix}\ [\![\Delta \mid \Gamma \vdash M{:}\,\tau \to \tau]\!]_{\rho,\eta}\\
[\![\Delta \mid \Gamma \vdash \mathsf{if}\ (E{=}F)\ M\ N{:}\,\mathsf{com}]\!]_{\rho,\eta} &\overset{def}{=} \mathsf{if}\ [\![E]\!]_{\rho}{=}[\![F]\!]_{\rho}\ \mathsf{then}\ [\![\Delta \mid \Gamma \vdash M{:}\,\mathsf{com}]\!]_{\rho,\eta}\\
&\phantom{{}={}} \mathsf{else}\ [\![\Delta \mid \Gamma \vdash N{:}\,\mathsf{com}]\!]_{\rho,\eta}\\
[\![\Delta \mid \Gamma \vdash M; N{:}\,\mathsf{com}]\!]_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{let}\ k'\ \mathsf{be}\ \lambda h'.\, [\![\Delta \mid \Gamma \vdash N{:}\,\mathsf{com}]\!]_{\rho,\eta}(h',k)\\
&\phantom{{}={}} \mathsf{in}\ [\![\Delta \mid \Gamma \vdash M{:}\,\mathsf{com}]\!]_{\rho,\eta}(h,k')\\
[\![\Delta \mid \Gamma \vdash \mathsf{let}\ i{=}\mathsf{new}\ \mathsf{in}\ M{:}\,\mathsf{com}]\!]_{\rho,\eta}(h,k) &\overset{def}{=} [\![\Delta, i \mid \Gamma \vdash M{:}\,\mathsf{com}]\!]_{\rho[i\to l],\eta}(h \bullet [l{\to}\mathit{nil}], k)\\
&\phantom{{}={}} \qquad (\text{where } l \in (\mathit{Loc}{-}\mathsf{supp}(h,\rho,\eta,k)))\\
[\![\Delta \mid \Gamma \vdash \mathsf{free}(E){:}\,\mathsf{com}]\!]_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{if}\ [\![E]\!]_{\rho}{\notin}\mathsf{dom}(h)\ \mathsf{then}\ \mathit{err}\\
&\phantom{{}={}} \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]\ \mathsf{in}\ M{:}\,\mathsf{com}]\!]_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{if}\ [\![E]\!]_{\rho}{\notin}\mathsf{dom}(h)\ \mathsf{then}\ \mathit{err}\\
&\phantom{{}={}} \mathsf{else}\ [\![\Delta, i \mid \Gamma \vdash M{:}\,\mathsf{com}]\!]_{\rho[i\to h([\![E]\!]_{\rho})],\eta}(h,k)\\
[\![\Delta \mid \Gamma \vdash [E]{=}F{:}\,\mathsf{com}]\!]_{\rho,\eta}(h,k) &\overset{def}{=} \mathsf{if}\ [\![E]\!]_{\rho}{\notin}\mathsf{dom}(h)\ \mathsf{then}\ \mathit{err}\ \mathsf{else}\ k(h[[\![E]\!]_{\rho}{\to}[\![F]\!]_{\rho}])
\end{aligned}$$

Fig. 5. 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,[7] and the accessibility relation is the preorder defined by the separating conjunction for relations:

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

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.

[7] 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 $\forall h_0, h_1.\, h_0[r]h_1 \iff (\pi \cdot h_0)[r](\pi \cdot h_1)$.

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

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

where G is the set $O - \{\mathit{err}\} = \{\mathit{normal}, \bot\}$ of good observations. 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 [16].

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:

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

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 stack 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 6 shows the detailed interpretation of specifications. In the figure, we make use of the standard semantics of assertions [16]. We now explain three cases in the definition of $\models$.

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.

For all environments $\rho \in \llbracket \Delta \rrbracket$ and $\eta_0, \eta_1 \in \llbracket \Gamma \rrbracket$ and all worlds $r \in \mathcal{R}$,

$$\begin{array}{lcl}
(\rho, \eta_0, \eta_1, r) \models \{P\}M\{Q\} & \stackrel{def}{\Longleftrightarrow} & [\mathsf{eq}(\llbracket P \rrbracket_\rho) * r](\llbracket M \rrbracket_{\rho,\eta_0}, \llbracket M \rrbracket_{\rho,\eta_1})[\mathsf{eq}(\llbracket Q \rrbracket_\rho) * r] \\
(\rho, \eta_0, \eta_1, r) \models \varphi \otimes P & \stackrel{def}{\Longleftrightarrow} & (\rho, \eta_0, \eta_1, r * \mathsf{eq}(\llbracket P \rrbracket_\rho)) \models \varphi \\
(\rho, \eta_0, \eta_1, r) \models E = F & \stackrel{def}{\Longleftrightarrow} & \llbracket E \rrbracket_\rho = \llbracket F \rrbracket_\rho \\
(\rho, \eta_0, \eta_1, r) \models M = N & \stackrel{def}{\Longleftrightarrow} & \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}{\Longleftrightarrow} & \text{for all } s \in \mathcal{R}, \text{ if } (\rho, \eta_0, \eta_1, r * s) \models \varphi, \\
 & & \quad \text{then } (\rho, \eta_0, \eta_1, r * s) \models \psi \\
(\rho, \eta_0, \eta_1, r) \models \forall i.\, \varphi & \stackrel{def}{\Longleftrightarrow} & \text{for all } v \in \mathit{ref},\ (\rho[i \to v], \eta_0, \eta_1, r) \models \varphi \\
(\rho, \eta_0, \eta_1, r) \models \exists i.\, \varphi & \stackrel{def}{\Longleftrightarrow} & \text{there exists } v \in \mathit{ref} \text{ s.t. } (\rho[i \to v], \eta_0, \eta_1, r) \models \varphi \\
(\rho, \eta_0, \eta_1, r) \models \forall x{:}\, \tau.\, \varphi & \stackrel{def}{\Longleftrightarrow} & \text{for all } m, n \in \llbracket \tau \rrbracket,\ (\rho, \eta_0[x \to m], \eta_1[x \to n], r) \models \varphi \\
(\rho, \eta_0, \eta_1, r) \models \exists x{:}\, \tau.\, \varphi & \stackrel{def}{\Longleftrightarrow} & \text{there exist } m, n \in \llbracket \tau \rrbracket \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}{\Longleftrightarrow} & (\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}{\Longleftrightarrow} & (\rho, \eta_0, \eta_1, r) \models \varphi \text{ or } (\rho, \eta_0, \eta_1, r) \models \psi
\end{array}$$

Fig. 6. Relational Interpretation of Separation Logic

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

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

Proposition 1. *For all $p, q \subseteq \mathit{Heap}$ and all $c \in \mathit{Heap} \to (\mathit{Heap} \times D + \{\mathit{err}\})_\perp$, quadruple $[\mathsf{eq}(p)](\mathsf{cps}(c), \mathsf{cps}(c))[\mathsf{eq}(q)]$ holds iff the below 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 $c(h) = \bot$, then $c(h \bullet h_1) = \bot$,
(b) if $c(h) \neq \bot$, 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 [16]. 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 $(-)^{\bot\bot}$. For each FM-cpo D and relation $r \subseteq D \times D$, we define two relations, $r^{\bot}$ on $[D \to O]$ and $r^{\bot\bot}$ on D, as follows:

$$k_1[r^{\bot}]k_2 \overset{def}{\iff} \forall d_1, d_2 \in D.\ (d_1[r]d_2 \Longrightarrow k_1(d_1)[\mathsf{eq}(G)]k_2(d_2)),$$
$$d_1[r^{\bot\bot}]d_2 \overset{def}{\iff} \forall k_1, k_2 \in [D \to O].\ (k_1[r^{\bot}]k_2 \Longrightarrow k_1(d_1)[\mathsf{eq}(G)]k_2(d_2)).$$

Operator $(-)^{\bot}$ dualizes a relation on D to one on observations on D, and $(-)^{\bot\bot}$ 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 $Heap \to (Heap + \{err\})_{\bot}$. A quadruple $[r](\mathsf{cps}(c_1), \mathsf{cps}(c_2))[s]$ holds, iff*

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

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 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 2 (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 1, we get that $\Delta \mid \Gamma \vdash \varphi$ is valid, which is just what the conclusion expresses. □

Corollary 1. *Suppose that $\Delta \mid x{:}\mathsf{com} \vdash \{P_1\}x\{Q_1\} \Rightarrow \{P\}M\{Q\}$ 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.*

$$\mathsf{put}_1 \equiv (\lambda i.\, \mathsf{let}\ j = [i]\ \mathsf{in}\ (\mathsf{free}(i); [k] := j) \qquad \mathsf{get}_1 \equiv (\lambda i.\, \mathsf{let}\ j = [k]\ \mathsf{in}\ [i] := j)$$
$$\{i \mapsto j * k \mapsto \text{-}\}\mathsf{put}_1(i)\{k \mapsto \text{-}\} \qquad \{i \mapsto \text{-} * k \mapsto \text{-}\}\mathsf{get}_1(i)\{i \mapsto \text{-} * k \mapsto \text{-}\}$$
$$\mathsf{put}_2 \equiv (\lambda i.\, \mathsf{let}\ k'{=}[k]\ \mathsf{in}\ (\mathsf{free}(k'); [k]{:=}i)) \quad \mathsf{get}_2 \equiv (\lambda i.\, \mathsf{let}\ k'{=}[k]\ \mathsf{in}\ \mathsf{let}\ j{=}[k']\ \mathsf{in}\ [i]{:=}j)$$
$$\{i \mapsto j * \exists k'.k \mapsto k' * k' \mapsto \text{-}\}\mathsf{put}_2(i)\{\exists k'.k \mapsto k' * k' \mapsto \text{-}\}$$
$$\{i \mapsto \text{-} * \exists k'.k \mapsto k' * k' \mapsto \text{-}\}\mathsf{get}_2(i)\{i \mapsto \text{-} * \exists k'.k \mapsto k' * k' \mapsto \text{-}\}$$
$$c \equiv (\mathsf{let}\ i''{=}\mathsf{new}\ \mathsf{in}\ [i'']{:=}i'; \mathsf{put}(i''); \mathsf{get}(i'))$$
$$\Delta \mid \Gamma \vdash (\forall i.\{P_1\}\mathsf{put}(i)\{Q_1\} \wedge \{P_2\}\mathsf{get}(i)\{Q_2\}) \Rightarrow \{i' \mapsto \text{-}\}c\{i' \mapsto \text{-}\}$$
$$\text{(where } \Delta = \{i', k\} \text{ and } \Gamma = \{\mathsf{put}\colon \mathsf{ref} \to \mathsf{com}, \mathsf{get}\colon \mathsf{ref} \to \mathsf{com}\})$$

Fig. 7. Two Implementations of a Buffer and a Simple Client

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:

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

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:

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

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, $\mathsf{put}(i)$ stores the value found at i in the buffer and $\mathsf{get}(i)$ retrieves the value stored in the buffer and stores it at i. Let $P_1 \equiv i \mapsto j$, and $Q_1 \equiv \mathsf{emp}$, 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: $\{P_1\}\mathsf{put}(i)\{Q_1\}$ and $\{P_2\}\mathsf{get}(i)\{Q_2\}$.

Figure 7 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 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 $\rho \in [\![\{i', k\}]\!]$, and define $f_1, f_2, g_1, g_2, c_1, c_2$ as follows:

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

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

$$\begin{aligned}\big(\forall v \in \mathit{ref}.\ &[\mathsf{eq}([\![P_1]\!]_{\rho[i\to v]}) * r](f_1(v), f_2(v))[\mathsf{eq}([\![Q_1]\!]_{\rho[i\to v]}) * r] \wedge \\ &[\mathsf{eq}([\![P_2]\!]_{\rho[i\to v]}) * r](g_1(v), g_2(v))[\mathsf{eq}([\![Q_2]\!]_{\rho[i\to v]}) * r]\big) \\ \Rightarrow\ &[\mathsf{eq}([\![i' \mapsto \text{-},\text{-}]\!]_\rho) * r](c_1, c_2)[\mathsf{eq}([\![i' \mapsto \text{-},\text{-}]\!]_\rho) * r].\end{aligned} \tag{1}$$

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: $r = \{(h_1, h_2) \mid \exists j.\, h_1 = [k\to j] \wedge \exists k'.\, h_2 = [k\to k'] \bullet [k'\to j]\}$. Then one can verify that the antecedent of the implication in (1) holds, and thus conclude that $[\mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho) * r](c_1, c_2)[\mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho) * r]$ holds. Take $(h_1, h_2) \in \mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho) * r$, 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 $(h'_{11}, h'_{21}) \in \mathsf{eq}([\![i' \mapsto \text{-}]\!]_\rho)$.

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 [14, 5]. To this end, we also expect to make use of the ideas of relational separation logic in [19] 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 [10]. 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 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., [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. Abstracting Allocation:The New new Thing. In *Proc. of CSL'06*, 2006.
3. N. Benton and B. Leperchey. Relational reasoning in a nominal semantics for storage. In *Proc. of TLCA'05*, pages 88–101, Nara, Japan, 2005.
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. I. Mijajlović, N. Torp-Smith, and P. O'Hearn. Refinement and separation context. In *Proc. of FSTTCS'04*, pages 421–433, Chennai, India, 2004.
9. I. Mijajlović and H. Yang. Data refinements with low-level pointer operations. In *Proc. of APLAS'05*, pages 19–36, Tsukuba, Japan, 2005.
10. P. O'Hearn and J. Reynolds. From Algol to polymorphic linear lambda-calculus. *Journal of the ACM*, 47(1):167–223, 2000.
11. 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.
12. 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.
13. M. Parkinson and G. Bierman. Separation logic and abstraction. In *Proc. of POPL'05*, pages 247–258, Long Beach, CA, USA, 2005.
14. G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Proc. of TLCA'93*, pages 361–375, Utrecht, Netherlands, 1993.
15. J. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
16. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. of LICS'02*, pages 55–74, Copenhagen, Denmark, 2002.
17. M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342:28–55, 2005.
18. N. Torp-Smith. *Advances in Separation Logic — A Study of Logics for Reasoning about Stateful Programs.* PhD thesis, IT University of Copenhagen, 2005.
19. H. Yang. Relational separation logic. *Theoretical Comput. Sci.*, 2005. (to appear).