A Relational Model for Confined Separation Logic

Wang Shuling
LMAM and Department of Informatics
School of Mathematical Sciences
Peking University, Beijing, China
joycy@math.pku.edu.cn

L. S. Barbosa and J. N. Oliveira
CCTC and Department of Informatics
Minho University
Portugal
{lsb,jno}@di.uminho.pt

Abstract

Confined separation logic is a new extension to separation logic designed to deal with problems involving dangling references within shared mutable structures. In particular, it allows for reasoning about confinement in object-oriented programs. In this paper, we discuss the semantics of such an extension by defining a relational model for the overall logic, parametric on the shapes of both the store *and the* heap. *This model provides a simple and elegant interpretation of the new confinement connectives and helps in seeking for duals. A number of properties of this logic are proved calculationally.*

1. Introduction

Reference aliasing is a well known problem in object oriented programming, where shared mutable structures are pervasive and access to a particular object may break another's integrity or leak sensitive information of the whole system [1]. In this domain, the extensive body of research on encapsulation mechanisms to support data abstraction is of limited help because usually such mechanisms correspond directly to language constructs, and such is not the case in reference based programs.

Confinement of objects to specific partitions of the global reference space have become, therefore, a major research issue in object-orientation. Static access modifiers found in current languages (such as the `private` and `protected` tags in JAVA) restrict only the visibility of methods, attributes, or variables, but do not constrain object references. Confined types [7], ownership [16] and universes [15] are fine-grained notions of confinement for aliasing control by enforcing static scoping of dynamic object references. However, they are either incomplete or too restrictive. This entails the need for a formal approach to confinement independent of syntactic restrictions and enabling one to assess different confinement schemes.

An interesting attempt in this direction is [4], which resorts to denotational semantics. The approach formalises type-based full encapsulation but only to address object representation independence. Moreover, a number of strong syntactic restrictions are imposed which exclude useful program idioms.

Our own contribution, partly presented in this paper, goes in a similar direction, but adopts a different approach. Our starting point is *separation logic* [24], an extension of Hoare Logic where formulæ are interpreted over suitable models of stores and heaps. In particular, it introduces a new form of conjunction, denoted $p * q$, which asserts that p and q hold for disjoint parts of the heap.

Separation logic has been extensively used to reason about pointer-based programs [24, 23, 8], fine-grained concurrency [17], and object orientation [21, 13, 14]. It can guarantee domain disjointness of object heaps and, therefore, prevent aliasing between objects laying in separated heaps. However, no attention is paid to the behavior of outgoing dangling references of separated heaps which may introduce subtle forms of *indirect* aliasing. As an illustration consider the following Hoare triple

$$\{p\}\ \ x := \mathsf{new}\ C(\ldots);\ \{p * x \mapsto \{\ldots\}\}$$

where $\{\ldots\}$ denotes the object created. C may be, for example, a node type used as an element of a linked stack. Then, the purpose of this piece of code is to allocate a node and use it as a link to a stack. Note, however, that such an object could not be put into a protection domain, because the post-condition of new does not assert the absence of any reference from the part of the heap which validates p to the new object.

The point is that, besides the domain separation of two heaps, we often need to express the restrictions upon outgoing references from heaps. In the example above, in fact we have more information about the relationship between

[1] Reference [28] reports on how the possibility of forging cryptographic authentication in a particular system arose as an unexpected consequence of a leaked reference to an internal data structure.

the two heaps, *ie.*, the original heap and the one where the new object lives. In particular, we know, that as the object has just been created, no older one has a reference to it. Therefore it is safe to put the new object into a protection domain, and not break confinement.

To express this sort of situations, we propose an extension to classical separation logic which introduces two new forms of separating conjunction: a *notIn* variant, represented by $\neg\triangleright$, asserting that no outgoing reference from the part of the heap where the first argument holds points to the part where the second argument holds, and a *In* variant, $\triangleright$, which ensures that all outgoing references from the part of the heap characterised by the first argument converge into the one where the second argument holds. With this new form of separating conjunction, the Hoare triple above can be re-written as

$$\{p\}\ \ x := \mathsf{new}\ C(\ldots)\ \ \{p \neg\triangleright x \mapsto \{\ldots\}\}$$

The new operator not only warranties heap domain disjointness, but also enforces the new post-condition.

Report [26] introduces the basic intuitions on such *confined separation logic* and discusses its application to concrete programming problems. It also shows how a number of formal schemes for confinement in the literature [9, 4, 7, 15], can be unified from a semantic point of view.

This paper goes further in this research direction by introducing a semantic model for this logic which is *generic* in the sense that it abstracts away from the specific heap structure, regarded as a mapping from a type K of references to a type parametric construction $\mathsf{F}(K)$ on K, for F a polynomial relator [6]. Typical heap models used in both classical separation logic [24] and its OO-extensions [21, 13, 14] arise by instantantiation. Moreover, instead of the usual set-theoretic semantics, we propose binary relations between heaps and stores as semantic domain for predicates. In this way, well known properties of separating conjunction (often presented without proof, as for example the existence of a formal dual implication) can be proved once and for all in the generic model. In such a setting we give compact and effective proofs of several properties of the logic connectives of confined separation logic. In particular, all conjunction variants proposed are shown to be adjoint to specific forms of implication. Due to space limitations, it is not possible to put together in a single paper the presentation of the semantic model and its application to handling specific confinement problems in object-oriented programming. Therefore, our focus in the sequel will be on formal semantics, applications being deferred to a follow-up paper [25].

The key technique in our approach is the so-called *pointfree (PF) transform* [20], which essentially means the conversion of predicate logic formulæ into binary relations by removing bound variables and quantifiers — a technique which, initiated in the 19c [2], eventually lead to what is known today as the *algebra of programming* [6, 3]. Such technique, which has been found fruitful for "theory refactoring" in other domains [19, 20], is based on the principle that *"everything is a binary relation"* once logical expressions are PF-transformed. One thereafter resorts to the powerful calculus of binary relations [1, 6] until a solution for the problem is found, which is mapped back to logics if required. At this point, another calculus — the Eindhoven quantifier calculus [3, 2] — is applied. In proceeding this way, as we expect the reader will appreciate in the sequel, elegant expressions replace lengthy formulæ and easy-to-follow calculations replace pointwise proofs with lots of "$\cdots$" notation, case analyses and natural language explanations for "obvious" steps.

This paper is structured as follows. After a brief introduction to the relational calculus and the pointfree transform in section 2, section 3 introduces a generic model for confined separation logic upon which its semantic properties can be established. This is done in sections 4 and 5. Conclusions and pointers to future work are discussed in sections 6 and 7.

2 Relational calculus

This section is a self-contained introduction to the fragment of the relational calculus, and the pointfree transform, used in the paper. The reader is referred to [6, 3] for a detailed account.

Relations. Let $B \xleftarrow{R} A$ denote a binary relation on datatypes A (source) and B (target). The underlying partial order on relations is written $R \subseteq S$ (read: *"R is at most S"*), meaning that S is either more defined or less deterministic than R, that is, $b\ R\ a \Rightarrow b\ S\ a$ holds, for all a, b.

$R \cup S$ denotes the union of two relations and $\top$ is the largest relation of its type. Its dual is $\bot$, the smallest such relation (the empty one). Equality on relations is established by $\subseteq$-antisymmetry.

Relations can be combined by three basic operators: composition ($R \cdot S$), converse (R°) and meet ($R \cap S$). R°, the *converse* of R is such that $a(R^\circ)b$ iff bRa holds. Meet corresponds to set-theoretical intersection and composition is defined in the usual way: $b(R \cdot S)c$ holds wherever there exists some mediating a such that $bRa\ \wedge\ aSc$. Converse

[2] The idea of encoding predicates in terms of relations was initiated by De Morgan in the 1860s and followed by Peirce who, in the 1870s, found interesting equational laws of the calculus of binary relations [22]. The pointfree nature of the notation which emerged from this embryonic work was later further exploited by Tarski and his students [27]. In the 1980's, Freyd and Scedrov [10] developed the notion of an *allegory* (a category whose morphisms are partially ordered) which finally accommodates the binary relation calculus as special case.

commutes both with composition and with itself

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \quad \text{and} \quad (R^\circ)^\circ = R \tag{1}$$

Coreflexive relations are fragments of the identity relation which model predicates or sets. The PF-transform of a (unary) *predicate* p is the coreflexive Φ_p such that $b\ \Phi_p a \ \equiv\ (b = a)\ \wedge\ (p\ a)$ holds, that is, the relation that maps every a which satisfies p (and only such a) onto itself. The PF-meaning of a set S is $\Phi_{\lambda a.(a \in S)}$.

Taxonomy. A taxonomy of binary relations can be defined in terms of the notions of *kernel* and *image* of a relation R, resp. $\mathsf{ker}\ R = R^\circ \cdot R$ and $\mathsf{img}\ R = R \cdot R^\circ$. A relation R is said to be *entire* (or total) iff its kernel is reflexive; and *simple* (or functional) iff its image is coreflexive. Dually, R is *surjective* iff R° is entire, and R is *injective* iff R° is simple. Finally, a relation is said to be a *function* iff it is both simple and entire. A constant function, always returning the same value v, is denoted by $\underline{v}$.

The interplay between functions and relations is a rich part of the binary relation calculus. For instance, rule

$$b(f^\circ \cdot R \cdot g)a \quad \equiv \quad (f\ b)R(g\ a) \tag{2}$$

plays a prominent rôle in the PF-transform. For instance, the pointwise definition of the kernel of a function f, $b(\mathsf{ker}\ f)a \ \equiv\ f\ b \ =\ f\ a$, stems from (2), whereby it is easy to see that $\top$ is the kernel of every constant function, $1 \xleftarrow{!} A$ included. (Function ! is the unique function of its type, where 1 denotes the singleton type.)

Given two preorders $\leq$ and $\sqsubseteq$, one may relate arguments and results of pairs of functions f and g in, essentially, two ways:

$$f \cdot \sqsubseteq \quad \subseteq \quad \leq \cdot g \tag{3}$$
$$f^\circ \cdot \sqsubseteq \quad = \quad \leq \cdot g \tag{4}$$

Actually, (3) is equivalent to $\sqsubseteq \quad \subseteq \quad f^\circ \cdot \leq \cdot g$. For $f = g$, this establishes $\sqsubseteq$ to $\leq$ monotonicity, thanks to (2). Both f, g in pattern (4) are monotone and said to be *Galois connected*, f (resp. g) being referred to as the *lower* (resp. *upper*) adjoint of the connection. By introducing variables in both sides of (4) via (2), we obtain (for all a, b)

$$(f\ b) \sqsubseteq a \quad \equiv \quad b \leq (g\ a) \tag{5}$$

Galois connections in which the two preorders are relation inclusion ($\leq, \sqsubseteq\ :=\ \subseteq, \subseteq$) are particularly interesting because the two adjoints are relational combinators and the connection itself is their universal property. The following table lists a few examples:

ψ	Φ_ψ
$\langle \forall\ a, b\ ::\ b\ R\ a \Rightarrow b\ S\ a \rangle$	$R \subseteq S$
$\langle \forall\ a\ ::\ f\ a = g\ a \rangle$	$f \subseteq g$
$\langle \forall\ a\ ::\ a\ R\ a \rangle$	$id \subseteq R$
$\langle \exists\ a\ ::\ b\ R\ a\ \wedge\ a\ S\ c \rangle$	$b(R \cdot S)c$
$b\ R\ a\ \wedge\ b\ S\ a$	$b\ (R \cap S)\ a$
$b\ R\ a \vee b\ S\ a$	$b\ (R \cup S)\ a$
$(f\ b)\ R\ (g\ a)$	$b(f^\circ \cdot R \cdot g)a$
TRUE	$b \top a$
FALSE	$b \perp a$

Figure 1. Sample of PF-transform rules

$(f\ R) \subseteq S \equiv R \subseteq (g\ S)$			
Description	f	g	**Obs.**
Converse	$(\_)^\circ$	$(\_)^\circ$	
Shunting rule	$(h \cdot)$	$(h^\circ \cdot)$	NB: h is a function
Right-division	$(\cdot R)$	$(\ /\ R)$	read "... over R"
difference	$(\_ - R)$	$(R \cup\ )$	

The main motivation for using Galois connections is their rich algebra of properties. For instance, the two adjoints f and g in a Galois connection are monotonic; lower adjoint f commutes with join and upper-adjoint g commutes with meet, wherever these exist; and two cancellation laws hold, on the left and on the right hand side:

$$b \leq g(f\ b) \qquad \text{and} \quad f\ (g\ a) \sqsubseteq a \tag{6}$$

3 A model for confined separation logic

Confined separation logic. In order to capture confinement our proposal adds to standard separation logic the following variants of conjunction to describe the behavior of dangling references of separated heaps and, therefore, to help in controlling indirect aliasing:

- The *In* variant, denoted by $p \triangleright q$, which asserts that p and q hold for disjoint parts of the heap and all references in the first part of the heap (which validates p) point into the second.
- The "dual" *notIn* variant, denoted by $p \mathrel{\neg\triangleright} q$, which asserts that p and q hold for disjoint parts of the heap and no references from the first point to the second.
- The *inBoth* variant, written $p \mathrel{\triangleleft\!\triangleright} q$, which requires that p and q hold for disjoint parts of the heap and all references from the first part are confined to either the second one or itself.

Finally, connective $\overline{p}$ asserts that p holds in an environment in which program variables do not refer to values stored in the heap where p holds. Note that, if p holds in a heap including all confined objects, $\overline{p}$ expresses that program variables can not refer to values that are confined.

The syntax of confined separation logic is given by

$$\begin{array}{rcl} p & ::= & e_1 = e_2 \mid p \vee p \mid p \wedge p \mid \forall t : T \bullet p \mid \exists t : T \bullet p \\ & & \mid \mathbf{emp} \mid v \mapsto e \mid p * p \mid p \mathrel{-\!\!*} p \\ & & \mid p \triangleright p \mid p \mathrel{\triangleleft\!\!\triangleright} p \mid \mid p \mathrel{\neg\!\triangleright} p \mid \overline{p} \end{array}$$

Recall from *e.g* [24] that singleton assertion $e_1 = e_2$ means both expressions have the same value, while $v \mapsto e$ is valid in a singleton heap which stores the value of e in the address refered by value v. Separating conjunction $*$ and separating implication $-\!\!*$ are defined as in [24].

A generic storage model. Separation logic is typically interpreted on a storage model coupling a store σ, for variables, and a heap H, as represented, for example, in the following diagram:

$$\begin{array}{ccc} Variables & \xrightarrow{\sigma} & Atom + Address \\ {\scriptstyle Aliases = \in\cdot\sigma}\downarrow & \nearrow_{\in} & \\ Address & \xrightarrow[H]{} & Atom + Address \end{array} \qquad (7)$$

where $\in$ is a membership relation which spots addresses (of type $Address$) in objects of type $Atom + Address$. The logic is, however, independent of the concrete shape either σ or H may take. This observation entails the *generic* characterisation represented in the following diagram:

$$\begin{array}{ccc} V & \xrightarrow{\sigma} & \mathsf{G}(B,K) \\ & {\scriptstyle \in_{\mathsf{G}}\cdot\sigma}\searrow & \downarrow{\scriptstyle \in_{\mathsf{G}}} \\ & & K \\ & {\scriptstyle \in_{\mathsf{F}}\cdot H}\nearrow & \uparrow{\scriptstyle \in_{\mathsf{F}}} \\ K & \xrightarrow[H]{} & \mathsf{F}(A,B,K) \end{array}$$

where V is the type of variable names and K is the type of references (addresses). As explained in section 1, *genericity* comes from the use of relators F and G to capture the *shape* of both the heap and the store information structures, respectively. Notice that parameters A, B are the *types of interest*. In the diagram σ is defined as a function from variables to values, whereas a heap H is a *simple* relation from addresses to values. Functorial membership relations $\in_{\mathsf{G}}$, $\in_{\mathsf{F}}$ [11] extract reference information from elements with parametric types stored in the store and the heap, respectively. Relation $\in_{\mathsf{F}} \cdot H$, for example, is the (immediate) reachability relation among references and fact $k(\in_{\mathsf{G}} \cdot \sigma)x$ asserts that variable x currently holds reference k. The kernel of $\in_{\mathsf{G}} \cdot \sigma$ expresses the aliasing equivalence relation. Specific instances of F and G specialise the storage model to particular classes of problems or programming paradigms. For example, a storage model for C-like programs, as above, is obtained by making $F(A,B,K) = G(B,K) = B + K$ where both variables and heap cells store either primitive values of type B or addresses in K. This is the model given in (7), for $B = Atom$ and $K = Address$. Similarly, object heaps arise by instantiating G as before and F by $F(A,B,K) = A \rightharpoonup (K + B)$ where A is the set of attribute names and the heap maps references to associations of attribute names to either values or references.

Separability. On such a generic storage model, our first step is to characterise a separability relation on heaps: notation $H_1 \parallel H_2$ denotes disjointness of H_1 and H_2. Formally,

$$H_1 \parallel H_2 \quad \stackrel{\text{def}}{=} \quad H_1 \cdot H_2^{\circ} \subseteq \bot \qquad (8)$$

because, denoting by $t\ H\ k$ the fact that *"thing t is the referent of reference k in heap H"*, we get

$$\begin{array}{ll} & \langle \forall\, b, a \ ::\ b(H_1 \cdot H_2^{\circ})a \Rightarrow \text{FALSE} \rangle \\ \equiv & \quad \{\ \text{de Morgan ; negation}\ \} \\ & \neg\langle \exists\, b, a \ ::\ b(H_1 \cdot H_2^{\circ})a \rangle \\ \equiv & \quad \{\ \text{introduce relational composition}\ \} \\ & \neg\langle \exists\, b, a \ ::\ \langle \exists\, k \ ::\ b\ H_1\ k \ \wedge\ k\ H_2^{\circ} a \rangle \rangle \\ \equiv & \quad \{\ \text{relational converse: } b\ R^{\circ} a \text{ the same as } a\ R\ b\ \} \\ & \neg\langle \exists\, b, a \ ::\ \langle \exists\, k \ ::\ b\ H_1\ k \ \wedge\ a\ H_2\ k \rangle \rangle \\ \equiv & \quad \{\ \exists\text{-nesting (Eindhoven quantifier calculus)}\ \} \\ & \neg\langle \exists\, b, a, k \ ::\ b\ H_1\ k \ \wedge\ a\ H_2\ k \rangle \end{array}$$

Actually, $\parallel$ can be extended for any pair of (not necessarily simple) relations:

$$R \parallel S \quad \stackrel{\text{def}}{=} \quad R \cdot S^{\circ} \subseteq \bot \qquad (9)$$

Properties of $\parallel$ are easily asserted by calculation. For example, we have

$$(R \cup S) \parallel T \quad \equiv \quad R \parallel T \ \wedge\ S \parallel T \qquad (10)$$

since

$$\begin{array}{ll} & (R \cup S) \parallel T \\ \equiv & \quad \{\ \text{by (9)}\ \} \\ & (R \cup S) \cdot T^{\circ} \subseteq \bot \\ \equiv & \quad \{\ \cdot\, T^{\circ} \text{ is a lower adjoint (6)}\ \} \\ & (R \cdot T^{\circ}) \cup (S \cdot T^{\circ}) \subseteq \bot \\ \equiv & \quad \{\ \cup\text{-universal}\ \} \\ & R \cdot T^{\circ} \subseteq \bot \ \wedge S \cdot T^{\circ} \subseteq \bot \\ \equiv & \quad \{\ \text{by (9)}\ \} \\ & R \parallel T \ \wedge\ S \parallel T \end{array}$$

Finally, we define relation $H \ * \ (H_1, H_2)$ to mean that H is the union of separate H_1 and H_2,

$$H \ * \ (H_1, H_2) \quad \stackrel{\text{def}}{=} \quad (H_1 \parallel H_2) \ \wedge \ (H = H_1 \cup H_2) \tag{11}$$

from which we immediately infer

$$H \ * \ (H_1, H_2) = H \ * \ (H_2, H_1) \tag{12}$$

In a similar way, one may define relations dealing with properties of dangling references. Such relations will be used in the sequel to define the new forms of separating conjunction. Again the definitions are independent of the concrete shape F of the heap. Thus,

$$H_1 \neg\triangleright H_2 \quad \stackrel{\text{def}}{=} \quad H_1 \parallel H_2 \ \wedge \ H_2 \cdot \in_{\mathsf{F}} \cdot H_1 \subseteq \bot \tag{13}$$

asserts that no outgoing reference in H_1 goes into separate H_2. Back to pointwise notation, the fact that path $H_2 \cdot \in_{\mathsf{F}} \cdot H_1$ is empty, corresponds to

$$\neg\langle \exists \, k, k' \ : \ k \in \delta \, H_1 \ \ \wedge \ \ k' \in \delta \, H_2 : \ k' \in_F (H_1 \, k)\rangle$$

where $\delta \, H \stackrel{\text{def}}{=} \mathsf{ker} \, H \cap id$ is the coreflexive representing the domain of relation H. Similarly, for the other variants:

$$H_1 \triangleright H_2 \ \stackrel{\text{def}}{=} \ H_1 \parallel H_2 \ \wedge \ \in_{\mathsf{F}} \cdot H_1 \subseteq H_2^{\circ} \cdot \top \tag{14}$$

$$H_1 \triangleleft\triangleright H_2 \ \stackrel{\text{def}}{=} \ H_1 \parallel H_2 \ \wedge \ \in_{\mathsf{F}} \cdot H_1 \subseteq (H_1 \cup H_2)^{\circ} \cdot \top \tag{15}$$

In words, $H_1 \triangleright H_2$ requires all outgoing references of H_1 go into separated H_2, and $H_1 \triangleleft\triangleright H_2$ says that all outgoing references in H_1 are confined either to H_2 or itself.

Semantics. Instead of the usual semantics of assertions in terms of predicates on pairs (σ, H), we resort to *binary relations* between heaps and stores. Thus, assertion

$$H [\![p]\!] \sigma \tag{16}$$

asserts predicate p holds on state (σ, H). So $[\![p]\!]$ is a binary relation. The semantics of elementary assertions is given as follows [3]

$$H [\![e_1 = e_2]\!] \sigma \quad \stackrel{\text{def}}{=} \quad e_1 (\mathsf{ker} \, \sigma) e_2 \tag{17}$$

$$H [\![\mathbf{emp}]\!] \sigma \quad \stackrel{\text{def}}{=} \quad H \subseteq \bot \tag{18}$$

$$H [\![v \mapsto e]\!] \sigma \quad \stackrel{\text{def}}{=} \quad H = \underline{\sigma e} \cdot \underline{\sigma v}^{\circ} \tag{19}$$

Notice that (19) is equivalent, by (1), to $H = \sigma \cdot \underline{e} \cdot \underline{v}^{\circ} \cdot \sigma^{\circ}$. First-order connectives are easy to specify in terms of relations, for example, $[\![p \ \wedge \ q]\!] \stackrel{\text{def}}{=} [\![p]\!] \cap [\![q]\!]$ or $[\![p \vee q]\!] \stackrel{\text{def}}{=} [\![p]\!] \cup [\![q]\!]$. Preorder $\rightarrow$ on assertions is defined by

$$p \rightarrow q \ \stackrel{\text{def}}{=} \ [\![p]\!] \subseteq [\![q]\!] \tag{20}$$

so that it can be distinguished from standard logic implication $\Rightarrow$. Its anti-symmetric closure will be denoted by symbol $\leftrightarrow$. The definition of *separating conjunction* resorts to the separability relation and relational *split* (defined by $(a, b) \langle R, S \rangle c$ iff aRc and bSc),

$$[\![p * q]\!] \quad \stackrel{\text{def}}{=} \quad (*) \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{21}$$

[3] PF-expression $\underline{k}$ denotes the (polymorphic) everywhere k constant function.

which is the PF-transform of

$$H [\![p * q]\!] \sigma \quad \stackrel{\text{def}}{=} \quad \langle \exists \, H_0, H_1 \ :: \ H * (H_0, H_1) \ \wedge \ H_0 [\![p]\!] \sigma \ \wedge \ H_1 [\![q]\!] \sigma \rangle$$

On the other hand, the *notIn* variant appears as

$$[\![p \neg\triangleright q]\!] \quad \stackrel{\text{def}}{=} \quad (*) \cdot \Phi_{\neg\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{22}$$

(Recall from section 2, that $\Phi_{\neg\triangleright}$ is the coreflexive associated to predicate $\neg\triangleright$ (13) on heap pairs.) A consequence of $\Phi_{\neg\triangleright}$ being coreflexive is that $p \neg\triangleright q \rightarrow p * q$ holds. Carrying on, we define:

$$[\![p \triangleright q]\!] \quad \stackrel{\text{def}}{=} \quad (*) \cdot \Phi_{\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{23}$$

$$[\![p \triangleleft\triangleright q]\!] \quad \stackrel{\text{def}}{=} \quad (*) \cdot \Phi_{\triangleleft\triangleright} \cdot \langle [\![p]\!], [\![q]\!] \rangle \tag{24}$$

Clearly $\triangleright \subseteq \triangleleft\triangleright$ and therefore, $p \triangleright q \ \rightarrow \ p \triangleleft\triangleright q$ holds. Moreover, since $\Phi_{\triangleright}$, $\Phi_{\triangleleft\triangleright}$ are coreflexive, we have $p \triangleleft\triangleright q \rightarrow p * q$ and $p \triangleright q \rightarrow p * q$. Finally, the relational semantics of $\overline{p}$ is given by $[\![\overline{p}]\!] \stackrel{\text{def}}{=} \text{NA} \cap [\![p]\!]$, where relation $H \ \text{NA} \ \sigma$ (read "H is not accessible from σ") is $H \ \text{NA} \ \sigma \stackrel{\text{def}}{=} H \cdot \in_{\mathsf{G}} \cdot \sigma \subseteq \bot$.

4 Conjunction and implication

The classical case. In this section we take advantage of our relational model and establish upper adjoints for each form of conjunction considered in confined separation logic. We begin by dealing with the well-known (but usually stated without a formal proof) fact that $(p*)$ and $(p{-\!\!*})$ constitute a Galois connection (GC). Our method differs from the standard practice in that, instead of *postulating* the definition of $(p{-\!\!*})$ and then verifying that it adjoins with $(p*)$, we actually *calculate* the definition by regarding the GC itself as an equation whose unknown is the upper adjoint. So, our starting point is *equation*

$$(p * x) \rightarrow y \equiv x \rightarrow (p \mathrel{-\!\!*} y) \tag{25}$$

where we know everything apart from $(p{-\!\!*})$. At PF-level, the calculation is quite simple and stems from a more basic GC where relational *split* performs the role of lower adjoint,

$$\langle R, S \rangle \subseteq X \quad \equiv \quad S \subseteq R \blacktriangleright X \tag{26}$$

The intuition behind relational combinator $\blacktriangleright$ is captured by its pointwise expansion [4]

$$b(R \blacktriangleright S)a \quad \equiv \quad \langle \forall \, c \ : \ c \, R \, a : \ (c, b) \, S \, a \rangle \tag{27}$$

[4] The quantified expression stems from the pointfree $\pi_2^{\circ} \setminus (\pi_1^{\circ} \cdot R \Rightarrow S)$.

We reason:

$$
\begin{array}{ll}
& (p * x) \to y \\
\equiv & \quad \{\ (20)\ \} \\
& [\![p * x]\!] \subseteq [\![y]\!] \\
\equiv & \quad \{\ (21) \text{ following by GC of left division } \} \\
& \langle [\![p]\!], [\![x]\!] \rangle \subseteq (*) \setminus [\![y]\!] \\
\equiv & \quad \{\ (26)\ \} \\
& [\![x]\!] \subseteq [\![p]\!] \blacktriangleright ((*) \setminus [\![y]\!]) \\
\equiv & \quad \{\ \text{introduce } p \mathrel{-\!\!*} y \text{ st } [\![p \mathrel{-\!\!*} y]\!] = [\![p]\!] \blacktriangleright ((*) \setminus [\![y]\!])\ \} \\
& [\![x]\!] \subseteq [\![p \mathrel{-\!\!*} y]\!] \\
\equiv & \quad \{\ (20)\ \} \\
& x \to (p \mathrel{-\!\!*} y)
\end{array}
$$

To spell out the pointwise meaning of $p \mathrel{-\!\!*} y$ we resort to the Eindhoven quantifier calculus [3]:

$$
\begin{array}{ll}
& H[\![p \mathrel{-\!\!*} y]\!]\sigma \\
\equiv & \quad \{\ \text{above}\ \} \\
& H([\![p]\!] \blacktriangleright ((*) \setminus [\![y]\!]))\sigma \\
\equiv & \quad \{\ (27)\ \} \\
& \langle \forall\ H_0\ :\ H_0[\![p]\!]\sigma :\ (H_0, H)((*) \setminus [\![y]\!])\sigma \rangle \\
\equiv & \quad \{\ \text{left division (pointwise)}\ \} \\
& \langle \forall\ H_0\ :\ H_0[\![p]\!]\sigma :\ \langle \forall\ H_1\ :\ H_1 * (H_0, H) :\ H_1[\![y]\!])\sigma \rangle \rangle \\
\equiv & \quad \{\ \text{nesting: (4.21) of [3]}\ \} \\
& \langle \forall\ H_0, H_1\ :\ H_0[\![p]\!]\sigma\ \wedge\ H_1 * (H_0, H) :\ H_1[\![y]\!])\sigma \rangle \\
\equiv & \quad \{\ \star \text{ definition (11) and one-point rule (4.24) of [3]}\ \} \\
& \langle \forall\ H_0\ :\ H_0[\![p]\!]\sigma\ \wedge\ H_0 \parallel H :\ (H_0 \cup H)[\![y]\!])\sigma \rangle \\
\equiv & \quad \{\ \text{trading: (4.28) of [3]}\ \} \\
& \langle \forall\ H_0\ :\ H_0 \parallel H :\ H_0[\![p]\!]\sigma \Rightarrow (H_0 \cup H)[\![y]\!])\sigma \rangle
\end{array}
$$

Summing up, we've calculated

$$
\begin{array}{l}
H[\![p \mathrel{-\!\!*} q]\!]\sigma \quad \stackrel{\text{def}}{=} \\
\quad \langle \forall\ H_0\ :\ H_0 \parallel H :\ H_0[\![p]\!]\sigma \Rightarrow (H \cup H_0)[\![q]\!]\sigma \rangle
\end{array} \tag{28}
$$

which is, in fact, the standard definition [24] Once the Galois connection is established, a number of properties come for free. First of all, equation (12) leads to

$$
(x * p) \to y \equiv x \to (p \mathrel{-\!\!*} y) \tag{29}
$$

which, with (25), corresponds to the currying and decurrying rules in [24]. Moreover, being lower adjoints, both $(p*)$ or $(*p)$ are monotonic and distribute over disjunction:

$$
p * (x_1 \vee x_2) \quad \leftrightarrow \quad (p * x_1) \vee (p * x_2) \tag{30}
$$

$$
(x_1 \vee x_2) * p \quad \leftrightarrow \quad (x_1 * p) \vee (x_2 * p) \tag{31}
$$

Similarly, as upper adjoint, $p \mathrel{-\!\!*}$ is monotonic and distributes over conjunction,

$$
p \mathrel{-\!\!*} (x_1\ \wedge\ x_2) \quad \leftrightarrow \quad (p \mathrel{-\!\!*} x_1)\ \wedge\ (p \mathrel{-\!\!*} x_2) \tag{32}
$$

Cancellation laws such as

$$
x \to (p \mathrel{-\!\!*} (p * x)) \tag{33}
$$

$$
p * (p \mathrel{-\!\!*} y) \to y \tag{34}
$$

hold. On the other hand, monotonicity of $(p*)$ and $(*p)$ entails

$$
p_1 \to p_2\ \text{ and }\ q_1 \to q_2 \quad \Rightarrow \quad (p_1 * q_1 \to p_2 * q_2) \tag{35}
$$

which corresponds to the inference rule showing that separating conjunction is monotone with respect to implication in [24].

The following properties are also direct consequences of adjointness, which, however, are not usually mentioned in the literature:

$$
\textbf{emp} \quad \to \quad p \mathrel{-\!\!*} p \tag{36}
$$

$$
p * x \quad \leftrightarrow \quad p * (p \mathrel{-\!\!*} (p * x)) \tag{37}
$$

$$
p \mathrel{-\!\!*} x \quad \leftrightarrow \quad p \mathrel{-\!\!*} (p * (p \mathrel{-\!\!*} x)) \tag{38}
$$

This provides evidence of the usefulness of our approach to 'discover' new, underlying laws.

Galois connections for confined separating logic. Similarly, the three forms of separating conjuntion for confined separation logic can be shown to take part in their own Galois connections. This is where our calculational techniques pay off. If we compare (22, 23, 24) to the standard case (21), we realize that the difference resides in an extra coreflexive (resp. $\Phi_{\neg\triangleright}$, $\Phi_{\triangleright}$ and $\Phi_{\triangleleft\triangleright}$) mediating separate union $(*)$ and the split of relations which capture the semantics of arguments p and q. This means that our calculation of the upper adjoint $(p \mathrel{-\!\!*})$ in the standard case can be re-used by sticking such a coreflexive to $(*)$ and carrying on. For the first form of confined separating conjunction, this leads immediately to the following upper adjoint for $(p \mathrel{-\!\!\neg\triangleright})$:

$$
\begin{array}{l}
H[\![p \mathrel{-\!\!\neg\triangleright} y]\!]\sigma \quad \stackrel{\text{def}}{=} \\
\quad \langle \forall\ H_0\ :\ H_0 \neg\triangleright H :\ H_0[\![p]\!]\sigma \Rightarrow (H_0 \cup H)[\![y]\!]\sigma \rangle
\end{array} \tag{39}
$$

Expression $p \mathrel{-\!\!\neg\triangleright} q$ asserts that, if the current heap is extended with a disjoint part in which p holds, and dangling references do not point into the current one, then q will hold in the whole heap.

Also in a similar way, but now replacing $\Phi_{\neg\triangleright}$ by either $\Phi_{\triangleleft\triangleright}$ or $\Phi_{\triangleright}$ respectively, we establish $\triangleleft\triangleright$, $\triangleright$ as lower adjoints for corresponding forms of implication. Actually one is lead to the following definitions of $p \mathrel{-\!\!\triangleleft\triangleright} y$ and $p \mathrel{-\!\!\triangleright} y$ as the upper adjoints of $(p \triangleleft\triangleright)$ and $(p \triangleright)$, respectively.

$$
\begin{array}{l}
H[\![p \mathrel{-\!\!\triangleleft\triangleright} y]\!]\sigma \quad \stackrel{\text{def}}{=} \\
\quad \langle \forall H_0 : H_0 \triangleleft\triangleright H : H_0[\![p]\!]\sigma \Rightarrow (H_0 \cup H)[\![y]\!]\sigma \rangle
\end{array} \tag{40}
$$

$$
\begin{array}{l}
H[\![p \mathrel{-\!\!\triangleright} y]\!]\sigma \quad \stackrel{\text{def}}{=} \\
\quad \langle \forall H_0 : H_0 \triangleright H : H_0[\![p]\!]\sigma \Rightarrow (H_0 \cup H)[\![y]\!]\sigma \rangle
\end{array} \tag{41}
$$

When compared with standard separated implication, all of the above place extra restrictions on the augmented heap, in terms of how dangling references are handled. Note that all properties derived for $(p*)$ hold *for free* for all its confined versions, thanks to the 'machinery' of Galois connections.

Intuitionistic p:	$[\![p]\!] = \supseteq \cdot [\![p]\!]$
Strictly-exact p:	$[\![p]\!]$ is *simple*, that is $[\![p]\!] \cdot [\![p]\!]^\circ \subseteq id$
Domain-exact p:	$\delta \leq [\![p]\!]^\circ$
Pure p:	$[\![p]\!]$ is a *right-condition*

Figure 2. Reynolds' classes re-visited.

5 Modelling and reasoning

The strength of a semantic model is assessed through both its expressive power and suitability for formal reasoning. This section illustrates how properties in separation logic, either in the standard or confined variants, can be formulated and established by calculating their interpretations in the relational model. It further illustrates how reasoning is conducted within the model in a calculational way. We just give two examples. (The interested reader is referred to report [25] for a full account.)

The first example checks the semantics of confinement against what happens to standard property

$$\mathbf{emp} * p \;\leftrightarrow\; p \;\leftrightarrow\; p * \mathbf{emp} \tag{42}$$

In the confined variants semantic rules entail

$$H[\![p]\!]S \;\wedge\; \Phi_\alpha(H, \bot) \quad\equiv\quad H[\![p]\!]S$$

or

$$H[\![p]\!]S \;\wedge\; \Phi_\alpha(\bot, H) \quad\equiv\quad H[\![p]\!]S$$

where α ranges over the three given variants. Checking $\Phi_\alpha(\bot, H)$ and $\Phi_\alpha(H, \bot)$ for $\alpha := \triangleright$ leads to:

$$\mathbf{emp} \triangleright p \quad\leftrightarrow\quad p$$

and

$$p \triangleright \mathbf{emp} \;\leftrightarrow\; p \quad\Leftarrow\quad p \rightarrow \mathbf{emp}$$

as the reader can easily verify. Furthermore, it is immediate to conclude that the two other variants trivially preserve the standard rule.

As exemplified above, confined variants of separating conjunction behave in particular ways even wrt some standard properties. In [25] we prove, for example, that $\triangleright$ is only semi-associative, *ie.*,

$$(p_1 \triangleright p_2) \triangleright p_3 \rightarrow p_1 \triangleright (p_2 \triangleright p_3) \tag{43}$$

For further illustrating the potential of the relational model to express and reasoning in separation logic, our second example re-visits J. Reynolds characterization of classes of assertions in [24]. An alternative, but equivalent, characterization is introduced in Figure 2, where $\leq$ denotes the *injectivity* preorder on relations [12]. Notice that a relation R is a *right-condition* iff it can be expressed as $R = \top \cdot \Phi$ for some coreflexive Φ.

Finally, as a calculational example, we prove the following theorem which involves confined *In* conjunction and a side condition on the assertions involved:

$$(p \;\wedge\; q) \triangleright r \leftrightarrow p \;\wedge\; (q \triangleright r) \qquad \text{when } p \text{ is pure} \tag{44}$$

The proof reads

$$\begin{array}{ll}
& [\![p \;\wedge\; (q * r)]\!] \\
= & \quad \{\; p := \top \cdot \Phi \text{ since } p \text{ is pure} \;\} \\
& \top \cdot \Phi \cap (*) \cdot \Phi_\triangleright \cdot \langle [\![q]\!], [\![r]\!] \rangle \\
= & \quad \{\; \text{right-conditions: } \Phi \cdot R = R \cap \Phi \cdot \top \text{ [3]} \;\} \\
& (*) \cdot \Phi_\triangleright \cdot \langle [\![q]\!], [\![r]\!] \rangle \cdot \Phi \\
= & \quad \{\; \text{splits: } \langle R, S \rangle \cdot \Phi = \langle R, S \cdot \Phi \rangle \equiv \Phi \text{ coreflexive [12]} \;\} \\
& (*) \cdot \Phi_\triangleright \cdot \langle [\![q]\!] \cdot \Phi, [\![r]\!] \rangle \\
= & \quad \{\; \text{right-conditions: } \Phi \cdot R = R \cap \Phi \cdot \top \text{ [3]} \;\} \\
& (*) \cdot \Phi_\triangleright \cdot \langle \top \cdot \Phi \cap [\![q]\!], [\![r]\!] \rangle \\
= & \quad \{\; \top \cdot \Phi := p \text{ ; definitions} \;\} \\
& [\![(p \;\wedge\; q) * r]\!]
\end{array}$$

This is an example of a result which extends from standard separating conjunction to all confined variants.

6 Conclusions

This paper's contribution is twofold. On the one hand it provides a semantic characterisation of a new extension to separation logic designed to reason about *confinement* of references in programming models. On-going work includes the encoding of an object-oriented programming language in the semantic model proposed here. Preliminary research (in [26] and [25]) suggests the potential of confined separation logic to express and reason about confinement problems in object-oriented programs. We believe this may be an interesting alternative or complement to the range of syntactic mechanisms recently proposed to achieve confinement resorting to, *eg.*, static annotations or extended type systems.

On the other hand, the paper illustrates how the calculus of binary relations can be regarded as a handy alternative for carrying out calculational proofs which are succint and easy to follow, nicely complemented by the Eindhoven quantifier calculus in mapping relational expressions to logics back and forth. The 'discovery' of new operators by calculation along Galois connections — namely the *notIn* and *inBoth* implications — is particularly instructive. Moreover, since heaps and stores are modeled also as binary relations, we take double advantage of the PF-transform which, as in [18], works smoothly across logic and data semantics.

7 Related and Future Work

Confined separation logic belongs to a family of recent extensions to standard separation logic already cited in the introduction.

Reference [29], and before that [5], focus on obtaining a good language for specifying how two pointer programs are related, and effective rules for proving such specifications. Actually, the starting point of [29] is the observation that specifications in mainstream separation logic, given by a Hoare triple, are appropriate for specifying the input and output relation of a single command,

but not for the equivalence between two programs. Contrary to what may be suggested by the title of [29], our use of relational methods has a completely different aim: to provide an alternative to the usual set-theoretic semantics for (variants of) classical separation logic in terms of binary relationships between heaps and stores as semantic domains for predicates.

Both the logic and the relational approach to its semantics discussed here are relevant to other application domains. One of them is concerned with data representation. In particular, our current research targets a theory of refinement of polynomial data structures into heap-based implementations [18].

Acknowledgements. We thank Qiu Zongyan for fruitful discussions on confined separation logic, and comments about a draft version of this paper.

References

[1] C. Aarts, R. C. Backhouse, P. Hoogendijk, E. Voermans, and J. van der Woude. *A relational theory of datatypes*. 1992.

[2] R. Backhouse and D. Michaelis. Exercises in quantifier manipulation. In T. Uustalu, editor, *MPC'06*, pages 70–81. Springer Lect. Notes Comp. Sci. (4014), 2006.

[3] R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004.

[4] A. Banerjee and D. A. Naumann. Representation independence, confinement and access control[extended abstract]. In *Proceedings of POPL'02*, pages 166–177. ACM Press, New York, NY, 2002.

[5] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *Proc. of 31th ACM Symposium on Principles of Programming Languages*, pages 14–25, Venice, 2004.

[6] R. Bird and O. Moor. *The Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997.

[7] B. Bokowski and J. Vitek. Confined types. In *Proceedings of OOPSLA'99*, pages 82–96. ACM Press, New York, NY, USA, 1999.

[8] C. Calcagno, D. Distefano, P. O'Hearn, and H. Yang. Beyond reachability: Shape abstraction in the presence of pointer arithmetic. In *Proceedings of 13th SAS*, pages 386–400. Springer Lect. Notes in Comp. Sci. (4134), 2006.

[9] D. Clarke, J. Potter, and J. Noble. Ownership types for flexible alias protection. In *Proceedings of OOPSLA'98*, pages 48–64. ACM Press, New York, NY, 1998.

[10] P.J. Freyd and A. Ščedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North-Holland, 1990.

[11] P. F. Hoogendijk. *A generic theory of datatypes*. PhD thesis, Department of Computing Science, Eindhoven University of Technology, 1996.

[12] J.N.Oliveira. Pointfree foundations for lossless decomposition. Draft of paper in preparation, 2007.

[13] Quan Long, Qiu Zongyan, and Wang Shuling. A weakest precondition semantics for OO languages: An OO-Separation Logic approach. Technical report, School of Math., Peking University, 2006.

[14] R. Middelkoop, K. Huizing, and R. Kuiper. A separation logic proof system for a class-based language. In *Proceedings of LRPP*, 2004.

[15] P. Müller. *Modular specification and verification of object-oriented programs*. PhD thesis, FernUniversit at Hagen, 2002.

[16] J. Noble, J. Vitek, and J. Potter. Flexible alias protection. In *Proceedings of ECOOP'98*, volume 1445, 1998.

[17] P. O'Hearn. Resources, concurrency and local reasoning. *Theor. Comp. Sci.*, 375(1-3):271–307, 2007.

[18] J.N. Oliveira. Data transformation by calculation, June 2007. Tutorial notes for GTTSE'07 (to appear in Springer LNCS).

[19] J.N. Oliveira and C.J. Rodrigues. Transposing relations: from maybe functions to hash tables. In *Proc. of MPC'04*, pages 334–356. Springer Lect. Notes Comp. Sci. (3125), 2004.

[20] J.N. Oliveira and C.J. Rodrigues. Pointfree factorization of operation refinement. In *Proc. FM'2006*, pages 236–251. Springer Lect. Notes Comp. Sci. (4085), 2006.

[21] M. Parkinson and G. Bierman. Separation logic and abstraction. In *Proceedings of POPL'05*, pages 247–258. ACM Press, New York, NY, 2005.

[22] V. Pratt. Origins of the calculus of binary relations. In *Proc. of the 7th Annual IEEE Symp. on Logic in Computer Science*, pages 248–254, Santa Cruz, CA, 1992. IEEE Comp. Soc.

[23] U. Reddy and H. Yang. Correctness of data representations involving heap data structures. *Science of Computer Programming*, 50(1):129–160, 2004.

[24] J.C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of LICS'02*, pages 55–74, Los Alamitos, California, 2002. Springer-Verlag. Invited paper.

[25] Wang Shuling, L. S. Barbosa, J. N. Oliveira, and Qiu Zongyan. Confined separation logic applied to object confinement (relationally!). Technical report, (in preparation), 2008.

[26] Wang Shuling and Qiu Zongyan. Towards a semantic model of confinement with confined separation logic. Technical report, TR 2007-045, School of Math., Peking University, 2007.

[27] Alfred Tarski and Steven Givant. *A Formalization of Set Theory without Variables*. American Mathematical Society, 1987. AMS Colloquium Publications, volume 41.

[28] J. Vitek and B. Bokowski. Confined types in Java. *Software: Practice and Experience*, 31(6):507–532, 2001.

[29] H. Yang. Relational separation logic. *Theor. Comp. Sci.*, 375(2):308–334, 2007.