

A Fresh Look at Separation Algebras and Share Accounting*

DRAFT of March 4, 2009

Robert Dockins¹, Aquinas Hobor², and Andrew W. Appel¹

¹ Princeton University

² National University of Singapore

Abstract. *Separation Algebras* serve as models of Separation Logics; *Share Accounting* allows reasoning about concurrent-read/exclusive-write resources in Separation Logic. In designing a Concurrent Separation Logic and in mechanizing proofs of its soundness, we found previous axiomatizations of separation algebras and previous systems of share accounting to be useful but flawed. We adjust the axioms of separation algebras; we present a more powerful system of share accounting with a new, simple model; and we provide a reusable Coq development.

1 Introduction

Separation logic presents an elegant solution to the pointer aliasing problem which has long plagued Hoare logics. We have been using the framework of separation logic to examine the metatheory of Cminor enhanced with primitives for shared-memory concurrency [5]. Along the way, we have developed a generic library of constructions and proof techniques for separation logic and Hoare logic. Here we explain two basic features of our toolkit: separation algebras and share models. **Contribution 0:** Our proofs are machine checked in Coq.³

Calcagno, O’Hearn and Yang [3] present a semantics of separation logic based on structures they call “separation algebras.” But Calcagno’s definition is flawed; it is too limiting in some ways and too permissive in others. **Contribution 1:** We show several slight alterations to the definition of separation algebras that produce a class of objects with more pleasing mathematical properties and which are better suited to the task of generating useful separation logics; and we demonstrate the practical utility of an operator calculus over separation algebras.

We also revisit the idea of permission accounting. The main object of permission accounting is to allow separation logic to handle certain kinds of read sharing concurrent protocols. Share accounting allows a process to “own” some share of a memory location. If the process owns the *full* share at some location, it has read/write/deallocate permission. However, if the process has a partial share, it can only read from that location. **Contribution 2:** We present a new solution to the permission accounting problem which is superior to others presented previously by Bornat et al. [1] and by Parkinson [7].

* Supported in part by National Science Foundation grant CNS-0627650.

³ Reviewers may find a Coq development corresponding to this paper at http://www.cs.princeton.edu/~rdockins/papers/sa_account/proofs.tar.gz

2 Separation Algebras

Calcagno, O’Hearn, and Yang [3] introduced the notion of a *separation algebra*, which they defined as “a cancellative, partial commutative monoid.” More concretely, a separation algebra (SA) is a tuple $\langle A, \oplus, u \rangle$ where A is a carrier set, \oplus is a partial binary operation on A and u is an element of A satisfying the following axioms:

$$x \oplus y = y \oplus x \tag{1}$$

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z \tag{2}$$

$$u \oplus x = x \tag{3}$$

$$x_1 \oplus y = x_2 \oplus y \rightarrow x_1 = x_2 \quad \text{provided } x_1 \oplus y \text{ is defined} \tag{4}$$

The primary interest of a separation algebra is that it can be used to build a separation *logic* [9]. However, for most of this paper, we are going to consider separation algebras as first-class mathematical objects and investigate their properties.

We wish to represent and prove our models and logics in Coq. Dealing with partial functions in Coq is tricky, as the function space of Coq’s metatheory contains only total functions. Thus, we chose to take as primitive the 3-place *join relation* $J(x, y, z)$, which we usually write suggestively as $x \oplus y = z$. The partiality of the \oplus operation is now implicit in the fact that we are not guaranteed that there is some z such that $x \oplus y = z$ holds for given x and y . Reinterpreted in this relational setting, we say that $\langle A, J \rangle$, (where A is a carrier set and J is a three-place relation on A) is a separation algebra provided the following hold:⁴

$$x \oplus y = z_1 \rightarrow x \oplus y = z_2 \rightarrow z_1 = z_2 \tag{5}$$

$$x_1 \oplus y = z \rightarrow x_2 \oplus y = z \rightarrow x_1 = x_2 \tag{6}$$

$$x \oplus y = z \rightarrow y \oplus x = z \tag{7}$$

$$x \oplus y = a \rightarrow a \oplus z = b \rightarrow \exists c. y \oplus z = c \wedge x \oplus c = b \tag{8}$$

$$\exists u. \forall x. u \oplus x = x \tag{9}$$

That is, \oplus is a function (5), it is cancellative (6), commutative (7), associative (8), and has a unit u (9).

We are justified in calling this object “the” unit because the cancellation axiom guarantees that it must be unique. In addition, the unit is exactly the unique element u satisfying $u \oplus u = u$, again by the cancellative axiom.

3 The algebra of separation algebras

One of the easiest ways to build a separation algebra is to apply some operator to preexisting SAs.

⁴ Let **SA** be a category with SAs as the objects and SA homomorphisms as arrows. Let $\langle A, J_A \rangle$ and $\langle B, J_B \rangle$ be two SAs; $f : A \rightarrow B$ is a SA homomorphism from A to B iff for all $x, y, z \in A$, $J_A(x, y, z)$ implies $J_B(f(x), f(y), f(z))$. Identity arrows and composition are defined in the obvious ways.

Definition 1 (SA product). Let $\langle A, J_A \rangle$ and $\langle B, J_B \rangle$ be SAs. Then the product SA is $\langle A \times B, J \rangle$, where J is defined componentwise:

$$J((x_a, x_b), (y_a, y_b), (z_a, z_b)) \equiv J_A(x_a, y_a, z_a) \wedge J_B(x_b, y_b, z_b) \quad (10)$$

Definition 2 (SA function operator). Let A be a set and let $\langle B, J_B \rangle$ be a SA. Then the function SA is $\langle A \rightarrow B, J \rangle$, where J is defined pointwise:

$$J(f, g, h) \equiv \forall a \in A. J_B(f(a), g(a), h(a)) \quad (11)$$

The SA product and the SA function constructions are (isomorphic to) special cases of the general indexed product, or dependent function space operator.

Definition 3 (SA indexed product operator). Let I be a set, called the index set, and let P be a mapping from I to separation algebras. Then the indexed product SA is $\langle \prod x : I. P(x), J \rangle$ where J is defined pointwise:

$$J(f, g, h) \equiv \forall i \in I. J_{P(i)}(f(i), g(i), h(i)) \quad (12)$$

In each case, the SA axioms follow directly from the associated axioms of the underlying SAs. Pairs are isomorphic to the indexed product when the index is any two element set (such as the booleans), and the nondependent function space is isomorphic to the indexed product when P is chosen as a constant mapping.

What about the disjoint sum (coproduct) operator? Can it be constructed? Unfortunately, it cannot. Suppose we have two SAs $\langle A, J_A \rangle, \langle B, J_B \rangle$. We would like to define $\langle A + B, J \rangle$ such that J is the smallest relation satisfying:

$$J_A(x, y, z) \rightarrow J(\text{inl } x, \text{inl } y, \text{inl } z) \quad \text{for all } x, y, z \in A \quad (13)$$

$$J_B(x, y, z) \rightarrow J(\text{inr } x, \text{inr } y, \text{inr } z) \quad \text{for all } x, y, z \in B \quad (14)$$

Here $A + B$ is the disjoint union of A and B with inl and inr as the left and right injections. This structure cannot be a separation algebra under the original axioms. To see why, note that there must exist unique units in A and B ; call them u_A and u_B . Then both $\text{inl } u_A$ and $\text{inr } u_B$ satisfy $x \oplus x = x$. We noted above that the unit u is the unique element of a SA satisfying this equation. So if this structure were a SA, we would have $\text{inl } u_A = u = \text{inr } u_B$; but this results in a contradiction as inl and inr always produce unequal elements. The generalization of disjoint union to the indexed sum also fails for the same reason.

This is an unpleasant asymmetry; we can construct the product operator but not the coproduct. However, it also has practical consequences. We would like to use Hobor's stratified construction of resource maps [4] as a model for separation logic, but resource maps cannot be made a Calcagno-style separation algebra because of the requirement for unique units.

There is good news, however. We can slightly relax the axioms of separation algebras so that all these constructions can be carried out. Recall that the SA axioms above state that there must exist an identity for \oplus :

$$\exists u. \forall x. u \oplus x = x \quad (9)$$

If we simply swap the order of the quantifiers in this axiom, we get a weaker statement which says that every element of the SA has an associated unit:

$$\forall x. \exists u. u \oplus x = x \quad (15)$$

To distinguish these species of separation algebras, we call SAs defined by axioms 5, 6, 7, 8, and 9 Single-unit Separation Algebras (SSA), we call SAs defined by axioms 5, 6, 7, 8, and 15 Multi-unit Separation Algebras (MSA). Note that 9 implies 15, and thus the SSAs are a strict subset of the MSAs.

The product constructions we saw above (pairs, indexed products) as well as their duals (disjoint sum, indexed sum) are valid operators on MSAs.

Definition 4 (MSA indexed sum operator). *Let I be a set, called the index set. Let S be a mapping from I to separation algebras. Then the indexed sum MSA is $\langle \Sigma i : I. S(i), J \rangle$ such that J is the least relation satisfying:*

$$J_{S(i)}(x, y, z) \rightarrow J(\text{inj}_i(x), \text{inj}_i(y), \text{inj}_i(z)) \quad \text{for all } i \in I; x, y, z \in S(i) \quad (16)$$

Here inj_i is the injection function associated with i . The binary disjoint union operator (as with products) is isomorphic to the indexed sum when the index set contains exactly two elements.

The relaxation of the unit axiom also allows for the construction of the “discrete” separation algebra.

Definition 5 (discrete MSA). *Let A be a set. Then the discrete MSA is $\langle A, J \rangle$ where J is defined as the smallest relation satisfying:*

$$J(x, x, x) \quad \text{for all } x \in A \quad (17)$$

The discrete MSA has a trivial join relation that holds only when all three arguments are equal. Thus, *every* element of the discrete MSA is a unit. The discrete MSA is very useful for constructing MSAs over tuples where only some of the components have interesting join relations; the other components can be turned into trivial MSAs via the discrete MSA operator.

We also sometimes find use for the “lifting” operator, which removes all the units from a separation algebra and replaces them with a new, unique, unit. In essence, this operator coerces a MSA into a SSA.⁵ We’ll see this operator again in later sections.

Definition 6 (MSA lifting operator). *Let $\langle A, J_A \rangle$ be a multi-unit separation algebra. Define A^+ to be the subset of A containing all the nonunit elements, $A^+ = \{x \in A \mid \neg(x \oplus x = x)\}$. Let \perp be a distinguished element such that $\perp \notin A$. Then the lifting SA is $\langle A^+ \cup \{\perp\}, J \rangle$ where J is the least relation satisfying:*

$$J(\perp, x, x) \quad \text{for all } x \in A^+ \cup \{\perp\} \quad (18)$$

$$J(x, \perp, x) \quad \text{for all } x \in A^+ \cup \{\perp\} \quad (19)$$

$$J_A(x, y, z) \rightarrow J(x, y, z) \quad \text{for all } x, y, z \in A^+ \quad (20)$$

It is straightforward to verify the SA axioms given this definition.

⁵ Considered as a functor from **MSA** to **SSA**, the lifting operator is left adjoint to the inclusion functor from **SSA** to **MSA**.

4 Characterizing units

The only difference between a multi-unit separation algebra and a single-unit separation algebra is the form of the unit axiom. It is worthwhile to pause briefly to investigate the properties of the units of a SA.

We say that an element x of a SA (whether a SSA or a MSA) is a unit provided that there exists some y such that $x \oplus y = y$, and we say that x is the *unit for y* . We say that x is an *identity* just when x is a unit for every element with which it joins; that is, provided $x \oplus y = z \rightarrow y = z$ for all y and z . Finally, we say that x is an idempotent element iff $x \oplus x = x$.

In a SSA, we know that there is a unique element u which is the unit for every element of the SA; it is thus also an identity. We also asserted earlier, without proof, that the unit of a SSA is the unique element u satisfying $u \oplus u = u$; that is, u is idempotent. The unit axiom for SSAs immediately gives the existence such a u . To see that it is unique, suppose there is some other element u' such that $u' \oplus u' = u'$. Then we have $u \oplus u' = u'$ by the unit axiom, and $u = u'$ follows from the cancellation axiom.

Therefore the units, identities and idempotent elements coincide in a SSA and characterize exactly the element whose existence is asserted in equation 9.

It is perhaps surprising that the units, identities and idempotents also coincide in the MSA setting (of course, they are no longer necessarily unique). Suppose first that x is an identity. By the unit axiom, there exists a u_x such that $u_x \oplus x = x$. Then $u_x = x$ by commutivity and the identity property, giving $x \oplus x = x$. Therefore x is an idempotent. Next suppose that x is idempotent; then it is trivially a unit (for itself). All that remains to complete the triangle is to show that every unit is an identity.

Assume that x is a unit. Then there exists some y such that $x \oplus y = y$. We wish to show that x is an identity. Therefore, assume that $x \oplus p = q$ for some p and q ; we wish to show $p = q$. By the unit axiom for MSAs, there exists some element u_x such that $u_x \oplus x = x$. By associativity and commutivity there exists some c such that $u_x \oplus y = c$ and $c \oplus x = y$. From commutivity and cancellation it follows that $c = y$, which further gives that $u_x \oplus y = y$. Another application of cancellation gives that $u_x = x$. A second use of associativity shows that there exists some d such that $u_x \oplus p = d$ and $d \oplus x = q$. Once again, commutivity and cancellation show that $d = p$; substitution demonstrates that $x \oplus p = p$. Finally we deduce that $p = q$ because \oplus is a functional relation.

5 Inducing a separation logic

The purpose of a separation algebra is to generate a separation logic, that is, a Hoare-style program logic where the assertion language is (some extension of) the logic of bunched implications (BI). Calcagno et al. demonstrated that their interpretation leads to a boolean BI algebra, a model of BI. Now that we have weakened the definition of a separation algebra by relaxing the unit axiom, how do we know that we are still generating models for the desired class of logics?

Assume we have a MSA $\langle A, J \rangle$. The following is a model of HBI (a Hilbert-style axiomatization of the logic of bunched implications). \mathbf{Prop} is the type of Coq propositions, and the right-hand sides are stated in Coq’s metalogic.

$$\begin{aligned}
\mathit{formula} &\equiv A \rightarrow \mathbf{Prop} \\
a \models p &\equiv p(a) \\
p \vdash q &\equiv \forall a. a \models p \rightarrow a \models q \\
\top &\equiv \lambda a. \mathbf{True} \\
\perp &\equiv \lambda a. \mathbf{False} \\
p \wedge q &\equiv \lambda a. a \models p \wedge a \models q \\
p \vee q &\equiv \lambda a. a \models p \vee a \models q \\
p \rightarrow q &\equiv \lambda a. a \models p \rightarrow a \models q \\
\mathbf{emp} &\equiv \lambda a. a \oplus a = a \\
p * q &\equiv \lambda a. \exists a_1. \exists a_2. a_1 \oplus a_2 = a \wedge a_1 \models p \wedge a_2 \models q \\
p \multimap q &\equiv \lambda a. \forall a_1. \forall a'. a_1 \oplus a = a' \rightarrow a_1 \models p \rightarrow a' \models q
\end{aligned}$$

Table 1. A model of HBI given a SA

We too will interpret formulae of separation logic as predicates on the elements of a separation algebra (equivalently, members of the powerset). In Coq we simply define the formulae as $\mathbf{A} \rightarrow \mathbf{Prop}$, where \mathbf{A} is the type of elements in the SA, and \mathbf{Prop} is the type of propositions in Coq’s metatheory.

We have chosen to directly link our models to the proof theory of BI by showing⁶ a soundness proof with respect to the system HBI, a Hilbert-style axiomatic system for the (propositional) logic of bunched implications [8, Table 2]. The definitions which give rise to HBI are summarized in table 1; they are quite standard, except for the definition of the empty proposition **emp**. Ordinarily, one would define **emp** as the predicate which accepts only the unit of the SSA. However, by relaxing the unit axiom we allow multiple units, each of which must be characterized. Thus, we define **emp** as the predicate which accepts any element x provided that $x \oplus x = x$. This subsumes the ordinary definition in the event that the unit is unique. More importantly, however, from this definition we can prove that **emp** is the unit for separating conjunction.

Thus we see that relaxing the unit axiom does not take us outside the class of models of the logic of bunched implications.

6 Useful restrictions of SAs

Calcagno et al.’s definition of separation algebras [3] permits very strange kinds of logics that do not correspond well to the common philosophical view that the formulae in separation logic correspond to “resources.” (This problem is unrelated to the unit issue discussed above.) We can construct more well-behaved logics if we restrict the allowable separation algebras by requiring further axioms.

⁶ The accompanying Coq development contains the full set of definitions and proofs.

Let us examine one of these badly behaved SAs. Consider the structure $\langle \{0, 1, 2\}, +_3 \rangle$, of the integers with addition modulo 3. This structure satisfies the separation algebra axioms given in the previous section and the integer 0 is the unique unit. The problem with this SA is that the following holds: $1 +_3 2 = 0$. That is, the resource denoted by 1 combines with the resource denoted by 2 to give the empty resource 0. Stated another way, we can *split* the empty resource 0 to get two *nonempty* resources 1 and 2. To make an analogy to physics, 1 and 2 act as a resource/antiresource pair that annihilate each other when combined.

This is not at all how one expects resources to behave. If one has, for example, an empty pile of bricks and splits it into two piles, one expects to have exactly two piles containing no bricks. One does *not* expect to get one pile of bricks and another pile of antibricks.

The existence of antiresources is particularly troublesome because it interacts badly with the frame rule, a ubiquitous feature of separation logic. A program with no resources can write to memory! Proof: it splits the empty permission, obtaining a write and an anti-write permission. Using the frame rule, it “frames out” the antipermission, giving it the permission to perform a write. The ability to do this defeats the entire purpose of Separation Logic.

Calcagno et al. resolve this problem by requiring that all actions be “local.” The locality condition essentially captures the requirement that actions must be compatible with the frame rule. The locality condition *indirectly* restricts the allowable SAs for a given set of actions.

We prefer to directly rule out this troublesome class of separation algebras by disallowing negative resources; instead we require that SAs be “positive” by adding the following axiom:

$$a \oplus b = c \rightarrow c \oplus c = c \rightarrow a \oplus a = a \quad (21)$$

This axiom requires that whenever two elements join to create a unit element, these joined elements must themselves be units. This axiom rules out separation algebras such as the addition-modulo-3 example above. Furthermore, this axiom is preserved by all the separation-algebra operators we examined above, which means we have not lost the ground we gained by relaxing the unit axiom.

One of most compelling reasons for including (21) in the axiom base is the observation that it is usually satisfied by SAs of interest. All the nontoy SAs known to the authors (that is, those which can be used to reason about some computational system) including all the examples listed by Calcagno et al. [3] satisfy this axiom.

The positivity axiom also allows us to make a connection to order theory. We define an ordering relation on the elements of a separation algebra:

$$a \preceq b \equiv \exists x. a \oplus x = b \quad (22)$$

This relation is reflexive and transitive, which means that it is a preorder. It turns out that \preceq is antisymmetric, and thus a partial order, if and only if (21) holds. Thus we can induce a partial order from any positive separation algebra. This

allows us to apply order theory to the study of separation algebras, a potential source of additional insight.

Although the positivity axiom rules out certain classes of badly behaved separation algebras, there is an additional property we would like to have, which we call “disjointness:” no nonempty share joins with itself. If the separation logic lacks disjointness then unusual things can happen when defining predicates about inductive data in a program heap. For example, without the disjointness property, the most straightforward definition of a formula to describe binary trees in fact describes directed acyclic graphs [1].

Disjointness is easy to axiomatize:

$$a \oplus a = b \rightarrow a = b \tag{23}$$

In short, the disjointness axiom requires that any SA element which joins with itself be a unit. Read in its contrapositive form, it says that any nonunit element cannot join with itself (is disjoint). This axiom captures essentially the same idea as Parkinson’s “disjoint” axiom; the primary difference is that our axiom is expressed on separation *algebras*, whereas Parkinson’s axiom is expressed at the level of the separation logic.

As with the positivity axiom, this axiom is preserved by the SA operators presented in the previous section. It also implies positivity. The converse does not hold; disjointness is strictly stronger than positivity.

To see why disjointness (23) implies positivity (21), consider the following proof sketch. Assume (23) and $a \oplus b = c$ and $c \oplus c = c$ for arbitrary a , b , and c ; we wish to show $a \oplus a = a$. Then the following hold (modulo some abuse of notation):

$c \oplus c$	$= c$	assumed
$(a \oplus b) \oplus (a \oplus b)$	$= a \oplus b$	subst. $a \oplus b = c$
$(a \oplus a) \oplus (b \oplus b)$	$= a \oplus b$	comm. and assoc.
$(a \oplus a) \oplus b$	$= a \oplus b$	disjointness
$a \oplus a$	$= a$	cancellation

The converse fails. Consider the structure $\langle \mathbb{N}, + \rangle$ of natural numbers with addition. This fulfills the SA axioms, including positivity. However, we can easily find counterexamples to disjointness; every natural number $i > 0$ falsifies the disjointness axiom.

In conclusion, both the positivity and the disjointness axioms can be added to the axiom base for separation algebras while still preserving the important SA operators. The positivity axiom rules out certain kinds of exotic SAs, while the disjointness axiom provides a stronger form of separating conjunction that makes reasoning about inductive heap data structures much easier. Thus, we recommend that users of the separation algebra formalism add at least the positivity axiom. If the problem domain is reasoning about programming languages with mutable memory, then the disjointness axiom is also recommended.

7 Shares

An important application of separation algebras is to model Hoare logics of programming languages with mutable memory. We generate an appropriate separation logic by choosing the correct semantic model, that is, the correct separation algebra. A natural choice is to simply take the program heaps as the elements of the separation algebra together with some appropriate join relation.

In most of the early work in this direction, heaps were modeled as partial functions from addresses to values. In those models, two heaps join iff their domains are disjoint, the result being the union of the two heaps. However, this simple model is too restrictive, especially when one considers concurrency. It rules out useful and interesting protocols where two or more threads agree to share *read* permission to an area of memory.

There are a number of different ways to do the necessary permission accounting. Bornat et al. [1] present two different methods; one based on fractional permissions, and another based on token counting. Parkinson, in chapter 5 of his thesis [7], presents a more sophisticated system capable of handling both methods. However, this model contains a flaw, which we shall address below.

Fractional permissions are intended to handle the sorts of accounting situations that arise from concurrent divide-and-conquer algorithms. In such algorithms, a worker thread has read-only permission to the input dataset and it needs to divide this permission among any child threads it may spawn. When a child thread finishes its work, it returns its permission to its parent. Child threads may, in turn, need to split their permissions among their own children and so on. In order to handle any possible pattern of divide-and-conquer, splitting must be possible to an unbounded depth.

The token-counting method is intended to handle the accounting problem that arises from reader-writer locks. When a reader acquires a lock, it receives a “share token,” which it will later return when it unlocks. The lock itself tracks how many readers there are with an integer which is incremented when a reader locks and decremented when a reader unlocks. When the reader count is positive there are outstanding read tokens; when it is zero there are no outstanding readers and a writer may acquire the lock.

Here we will show how each of the above accounting systems arises from the choice of a “share model,” and we present our own share model which can handle both accounting methods and avoids a pitfall found in Parkinson’s model.

Suppose we have a separation algebra $\langle S, J_S \rangle$ of *shares*. Let L and V be sets of addresses and values, respectively. Now we can generically define a SA over heaps as follows:

$$H \equiv L \rightarrow (S \times V_{\perp})_{\perp} \tag{24}$$

This equation is quite concise, but it conceals some subtle points. First, the operators in this equation are the operators on SAs defined in section 3. We let V_{\perp} be the “discrete” SA over values (i.e., values V with the trivial join relation). Then $S \times V_{\perp}$ is the SA over pairs of shares and values, where the join is defined when the share components join and the value components are equal. Next we

construct the “lifted” SA $(S \times V_{\perp})_{\perp}$, which removes the unit values and adds a new, distinguished unit \perp . This has the effect of requiring values to be paired only with nonempty shares. Then finally, $L \rightarrow (S \times V_{\perp})_{\perp}$ builds the function space SA. Thus, heaps are essentially partial functions from locations to pairs of nonunit shares and values (with out-of-domain addresses mapped to \perp).

Note that this construction using a composition of orthogonal operators was made possible when we weakened the unit axiom. The ability to create interesting, complex SAs by building them up out of smaller ones is one of the main advantages of our approach.

Once we have done this, we can define the points-to operator of separation logic as:

$$\ell \mapsto_s v \equiv \lambda h. h(\ell) = (s, v) \wedge \forall \ell'. \ell \neq \ell' \rightarrow h(\ell') = \perp \quad (25)$$

Here, $\ell \in L$ is an address, $v \in V$ is a value, and $s \in S^+$ is a nonunit share. Rendered in English, $\ell \mapsto_s v$ means “the memory location at address ℓ contains v , and I have share s at this location, and I have no permission at any other locations.” Now the exact behavior of the points-to operator is dependent only on the properties of the share model S .

We can produce a separation logic very similar to the ones studied by Reynolds [9] and by Ishtiaq and O’Hearn [6] by choosing S to be the SA over booleans with the smallest join relation such that “false” is the unique unit.

Definition 7 (Boolean share model). *The boolean share model is $\langle \{\circ, \bullet\}, J \rangle$ where J is the least relation satisfying $J(\circ, x, x)$ and $J(x, \circ, x)$ for all $x \in \{\circ, \bullet\}$.*

We let \circ stand for the “false” boolean value, and \bullet for “true.” This share model is quite unsophisticated; one either has total, unrestricted permission at a location, or one has no permission at all. Note that when this share model is lifted the element \circ is removed, leaving \bullet as the only legal share annotation. This justifies omitting the annotation, leaving the more familiar notation, $\ell \mapsto v$.

A more sophisticated model is the one proposed by Boyland [2], which takes shares as fractions in the interval $[0, 1]$. Although Boyland works in the reals, the rationals suffice and we adopt them here.

Definition 8 (Fractional share model). *The fractional share model is given by $\langle [0, 1] \cap \mathbb{Q}, + \rangle$, where $+$ is the restriction of addition to a partial operation on the unit interval.*

The main advantage of the fractional share model is that it allows one to *split* a permission into two pieces that can later be recombined. One can derive a separation logic containing the axiom,

$$\ell \mapsto_{s_1+s_2} v \leftrightarrow \ell \mapsto_{s_1} v * \ell \mapsto_{s_2} v \quad (26)$$

Notice that for some given nonzero s , we can always find appropriate nonzero s_1 and s_2 so that we can go from the left side of this formula to the right side. A simple algorithm is to take $s_1 = s_2 = s/2$.

The fractional share model satisfies the positivity axiom; however, it does not satisfy the disjointness axiom. This leads to the problems noticed by Bornat et al. [1, section 13.1].

The other model of read-sharing examined by Bornat et al. is the *token factory* model. The idea is that some central authority starts with all the permission and doles out tokens, which it later collects back. The authority counts the number of outstanding tokens; when the count is 0, all the tokens are collected.

A slight modification of the construction given by Bornat et al. yields a suitable share model:

Definition 9 (Counting share model). *The counting share model is given by $\langle \mathbb{Z} \cup \{\perp\}, J \rangle$ where J is defined as the least relation satisfying:*

$$J(\perp, x, x) \quad \text{for all } x \in \mathbb{Z} \cup \{\perp\} \quad (27)$$

$$J(x, \perp, x) \quad \text{for all } x \in \mathbb{Z} \cup \{\perp\} \quad (28)$$

$$(x < 0 \vee y < 0) \wedge ((x + y \geq 0) \vee (x < 0 \wedge y < 0)) \rightarrow J(x, y, x + y) \quad (29)$$

for all $x, y \in \mathbb{Z}$

This definition sets up the nonnegative integers as token factories and negative integers as tokens. To absorb a token back into a factory, the integers are simply added. The token factory has collected all its tokens when its share is 0.

This share model validates the following logical axioms:

$$\ell \mapsto_n v \leftrightarrow (\ell \mapsto_{n+m} v * \ell \mapsto_{-m} v) \quad \text{for } n \geq 0 \text{ and } m > 0 \quad (30)$$

$$\ell \mapsto_{-(n+m)} v \leftrightarrow (\ell \mapsto_{-n} v * \ell \mapsto_{-m} v) \quad \text{for } n, m > 0 \quad (31)$$

$$(\ell \mapsto_0 v * \ell \mapsto_n v) \leftrightarrow \text{false} \quad (32)$$

Like the fractional model, the counting model satisfies positivity but not the disjointness axiom.

In his thesis, Parkinson goes on to define a more sophisticated share model that can support both the splitting and the token counting use cases.

Definition 10 (Parkinson's named share model). *Parkinson's named share model is given by $\langle \mathcal{P}(\mathbb{N}), \uplus \rangle$, where $\mathcal{P}(\mathbb{N})$ is the set of subsets of the natural numbers and \uplus is disjoint union.*

This share model satisfies the disjointness axiom, and is therefore positive. It also satisfies a version of the splitting axiom (26), with $+$ replaced by \uplus .

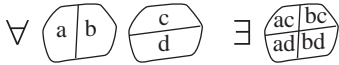
In order to support the token-counting use case, Parkinson constructs an embedding of the finite and cofinite subsets of \mathbb{N} into the infinite subsets of \mathbb{N} . These sets can be related to the counting model given above by considering the cardinality of the set (or set complement, for cofinite sets). We will see the details of this embedding later.

However, notice that this model does not guarantee that we can find a suitable splitting. That is, the splitting axiom holds whenever a splitting exists, but not every share can be split. In particular, singleton sets cannot be split into

two disjoint nonempty sets which partition the original. We cannot define a total function which calculates the splitting of a share in this model, and this makes it quite difficult to support the parallel divide-and-conquer use case.

We can fix this problem by restricting the model to include only the *infinite* subsets of \mathbb{N} (and the empty set). One simple method for finding a splitting of an infinite set works by “unzipping” the set. Simply enumerate the elements of the set, generating s_1 from the elements in even positions and s_2 from the elements in odd positions. Then s_1 and s_2 are infinite, disjoint, and partition the original set, as desired.

Unfortunately, restricting the model to infinite subsets means it is no longer closed under set intersection, and the following “cross split” axiom fails:

$$a \oplus b = z \wedge c \oplus d = z \rightarrow \exists ac \ ad \ bc \ bd. \ ac \oplus ad = a \wedge bc \oplus bd = b \wedge ac \oplus bc = c \wedge ad \oplus bd = d \quad (33)$$


That is, if a share can be split in two different ways, then there are 4 subshares which partition the original share and respect the splittings.

To see why this axiom fails, consider splitting \mathbb{N} into the primes/nonprimes and the even/odd numbers. All four sets are infinite, but the set $\{2\}$ of even primes is finite and thus not in the share model.

The failure of this axiom is not critical, but it is inconvenient. It sometimes happens in technical arguments about complicated schemes of share accounting that a string of tedious steps reasoning about associativity can be shortcut by applying the cross split axiom.

It might be possible to further restrict the model by more crisply characterizing just those subsets of interest and thus recover closure under set intersection and the cross split axiom. However, such a characterization is not immediately apparent. Instead, we have developed an alternate share model which avoids this difficulty while retaining the the ability to support both the token counting and the splitting use cases.

8 Binary tree share model

Before giving the explicit construction of our share model, we shall take a short detour to show how we can induce a separation algebra from a boolean algebra.

Definition 11 (Boolean separation algebra). *Let $\langle A, \sqsubseteq, \sqcap, \sqcup, 0, 1, \neg \rangle$ be a boolean algebra. Then, $\langle A, J \rangle$ is a separation algebra where J is defined as:*

$$J(x, y, z) \equiv x \sqcup y = z \wedge x \sqcap y = 0 \quad (34)$$

This structure satisfies the separation algebra axioms, including the disjointness axiom. It also has a unique unit. The cross split axiom follows directly from the existence of greatest lower bounds.

It is interesting to note that Parkinson’s model (without the restriction to infinite subsets) is the separation algebra derived from the standard powerset

boolean algebra. The restriction allows unbounded splitting, but destroys part of the structure of the boolean algebra.

Trees. Now we can restate our goal; we wish to construct a Boolean algebra which supports splitting and token counting. This means we must support a splitting function and we must be able to embed the finite and cofinite subsets of the naturals. We can build a model of shares supporting all these operations by starting with a very simple data structure: the humble binary tree. We consider binary trees with boolean-valued leaves and unlabeled internal nodes.

$$\tau ::= \circ \mid \bullet \mid \widehat{\tau \tau} \quad (35)$$

We use an empty circle \circ to represent a “false” leaf and the filled circle \bullet to represent a “true” leaf. Thus \bullet is a tree with a single leaf, $\widehat{\circ \bullet}$ is a tree with with one internal node, etc.

We define the ordering on trees as the least relation \sqsubseteq satisfying:

$$\circ \sqsubseteq \circ \quad (36)$$

$$\circ \sqsubseteq \bullet \quad (37)$$

$$\bullet \sqsubseteq \bullet \quad (38)$$

$$x_1 \sqsubseteq x_2 \quad \rightarrow \quad y_1 \sqsubseteq y_2 \quad \rightarrow \quad \widehat{x_1 y_1} \sqsubseteq \widehat{x_2 y_2} \quad (39)$$

$$\circ \cong \widehat{\circ \circ} \quad (40)$$

$$\bullet \cong \widehat{\bullet \bullet} \quad (41)$$

$x \cong y$ is defined to mean $x \sqsubseteq y \wedge y \sqsubseteq x$. The intuitive meaning is that $x \sqsubseteq y$ holds iff x has a \circ in at least every position y does once we expand leaf nodes using the congruence rules until the trees are the same shape.

This relation is reflexive and transitive; however it is not antisymmetric because of the structural congruence rules. We get around this by working only with the “canonical” trees. A tree is canonical if it is the tree with the fewest nodes in the equivalence class generated by \cong . Canonical trees always exist and are unique, and the ordering relation is antisymmetric on the domain of canonical trees. Therefore we can build a partial order using the canonical boolean-labeled binary trees with the above ordering relation.

The details of canonicalization are somewhat tedious, so in what follows we will work informally up to congruence. In the formal Coq development, however, we give a full account of canonicalization and show all the required properties.

Our first task is to implement the Boolean algebra signature. The trees \circ and \bullet are the least and greatest element of the poset, respectively. The least upper bound of two trees is calculated as the pointwise disjunction of booleans (expanding the trees as necessary to make them the same shape). For example, $\widehat{\bullet \circ} \sqcup \widehat{\circ \circ} \cong \widehat{\bullet \bullet \circ \circ} \sqcup \widehat{\circ \circ \circ \circ} \cong \widehat{\bullet \bullet \circ \circ} \cong \widehat{\bullet \bullet \circ \circ}$. Likewise, the greatest lower bound is found by pointwise conjunction, so that $\widehat{\bullet \circ} \sqcap \widehat{\circ \circ} \cong \widehat{\bullet \bullet \circ \circ} \sqcap \widehat{\circ \circ \circ \circ} \cong \widehat{\bullet \circ \circ \circ} \cong \widehat{\bullet \circ \circ \circ}$. Finally complement is pointwise

boolean complement: $\neg \bullet \widehat{\circ} \cong \widehat{\circ} \bullet$. The Boolean algebra axioms can be verified by simple inductive arguments over the structure of the trees.

In addition to the Boolean algebra operations, we also require an operation to split trees. Given some tree s , we wish to find two trees s_1 and s_2 such that $s_1 \sqcup s_2 \cong s$ and $s_1 \sqcap s_2 \cong \circ$ and both $s_1 \not\cong \circ$ and $s_2 \not\cong \circ$ provided that $s \not\cong \circ$. We can calculate s_1 by recursively replacing each \bullet leaf in s with $\bullet \widehat{\circ}$, and likewise s_2 by replacing \bullet with $\widehat{\circ} \bullet$.

We can usefully generalize this procedure by defining the “relativization” operator $x \rtimes y$, which replaces every \bullet leaf in x with the tree y . This operator is associative with identity \bullet . It distributes over \sqcup and \sqcap on the left, and is injective for non- \circ arguments.

$$x \rtimes \bullet = x = \bullet \rtimes x \quad (42)$$

$$x \rtimes \circ = \circ = \circ \rtimes x \quad (43)$$

$$x \rtimes (y \rtimes z) = (x \rtimes y) \rtimes z \quad (44)$$

$$x \rtimes (y \sqcup z) = (x \rtimes y) \sqcup (x \rtimes z) \quad (45)$$

$$x \rtimes (y \sqcap z) = (x \rtimes y) \sqcap (x \rtimes z) \quad (46)$$

$$x \rtimes y_1 = x \rtimes y_2 \rightarrow x = \circ \vee y_1 = y_2 \quad (47)$$

$$x_1 \rtimes y = x_2 \rtimes y \rightarrow x_1 = x_2 \vee y = \circ \quad (48)$$

Given this operator, we can more succinctly define the split of x as returning the pair containing $x \rtimes \bullet \widehat{\circ}$ and $x \rtimes \widehat{\circ} \bullet$. If this were the only use of the relativization, however, it would hardly be worthwhile to define it. Instead, the main purpose of this operator is to allow us to glue together arbitrary methods for partitioning permissions. In particular, we can split or perform token counting on any nonempty permission we obtain, no matter how it was originally generated.

In addition, we only have to concentrate on how to perform accounting of the full permission \bullet because we can let the \rtimes operator handle relativizing to some other permission of interest. This will make our development of a system for performing token counting considerably easier.

Following Parkinson, we will consider finite and cofinite sets of the natural numbers to support token counting. This structure has several nice properties. First, it is closed under set intersection, set union and set complement and it contains \mathbb{N} and \emptyset ; in other words, it forms a sub-Boolean algebra of the powerset Boolean algebra over \mathbb{N} . There also exists a homomorphism from the (co)finite subsets of \mathbb{N} into the boolean-labeled binary trees. We shall examine this homomorphism shortly. Finally, the cardinality of these sets can be mapped to the integers in following way:

$$\llbracket p \rrbracket_{\mathbb{Z}} = \begin{cases} -|p| & \text{when } p \text{ is finite and nonempty} \\ |\mathbb{N} \setminus p| & \text{when } p \text{ is cofinite} \end{cases} \quad (49)$$

It turns out that the cardinalities of disjoint (co)finite sets combine in exactly the way defined by the counting share model (equation 29). Later we will use this property to prove generalizations of the token counting axioms.

The homomorphism from (co)finite sets to boolean-labeled binary trees relies on encoding such sets as right-biased trees (trees where the left subtree of each internal node is always a leaf). Such trees are a list of booleans together with one extra boolean, the rightmost leaf in the tree. Then the i th boolean in the list encodes whether the natural number i is in the set. The final terminating boolean stands for all the remaining naturals. If it is \circ , the set is finite and does not contain the remaining naturals, and if it is \bullet the set is infinite and contains all the remaining naturals.

For example, the finite set $\{0, 2\}$ is encoded in tree form as $\bullet \circ \bullet \circ$. The coset $\mathbb{N} \setminus \{0, 2\}$ is encoded as $\circ \bullet \circ \bullet$. And, of course, $\bullet \circ \bullet \circ \oplus \circ \bullet \circ \bullet = \bullet$.

This encoding is in fact a Boolean algebra homomorphism; GLBs, LUBs, complements and the top and bottom elements are preserved. We write $\llbracket p \rrbracket_\tau = s$ when s is the tree encoding the (co)finite set p .

Now we can define a more sophisticated points-to operator which allows us to incorporate token counting along with permission splitting.

$$\ell \mapsto_{s,n} v \equiv \lambda h. h(\ell) = (s \bowtie \llbracket p \rrbracket_\tau, v) \wedge \llbracket p \rrbracket_Z = n \wedge \forall \ell'. \ell \neq \ell' \rightarrow h(\ell') = \perp \quad (50)$$

Then $\ell \mapsto_{s,n} v$ means that ℓ contains value v and we have a portion of the permission s indexed by n . If n is zero, we have all of s . If n is positive, we have a token factory over s with n tokens missing, and if n is negative, we have a token of s (of size $-n$).

This points-to operator satisfies the following axioms, which are generalizations of the splitting and counting axioms examined above:

$$(\ell \mapsto_{s,0} v * \ell \mapsto_{s,n} v) \leftrightarrow \text{false} \quad (51)$$

$$s_1 \oplus s_2 = s \rightarrow ((\ell \mapsto_{s_1,0} v * \ell \mapsto_{s_2,0} v) \leftrightarrow \ell \mapsto_{s,0}) \quad (52)$$

$$n_1 \oplus n_2 = n \rightarrow ((\ell \mapsto_{s,n_1} v * \ell \mapsto_{s,n_2} v) \leftrightarrow \ell \mapsto_{s,n} v) \quad (53)$$

$$\ell \mapsto_{s,n} v \rightarrow \exists! s'. \ell \mapsto_{s,n} v \leftrightarrow \ell \mapsto_{s',0} v \quad (54)$$

In equation (53), $n_1 \oplus n_2 = n$ refers to the token counting join relation on integers defined earlier in equation 29.

9 Conclusion

We have presented a new formulation of multi-unit separation algebras which overcomes some flaws in the original definition by Calcagno et al. [3]. The original definition is both too restrictive (it rules out desirable constructions, including coproducts) and too permissive (it allows badly-behaved “exotic” SAs). We examined a variety of operators over separation algebras that allow us to easily construct complicated separation algebras from simpler ones, and have shown an example of their utility.

We have also constructed a new solution to the permission accounting problem. Our share model based on boolean-labeled binary trees fully supports both

the splitting and token counting use cases for read sharing, and yet still validates the cross split axiom. No previously published system for share accounting has all these properties. Parkinson’s model [7] comes closest, but suffers from the inability to find splittings for some shares.

We have implemented the constructions discussed in this paper and proved their relevant properties using the proof assistant Coq. We will release the final version of our proof development along with the conference version of this paper.³

References

1. R. Bornat, C. Calcagno, P. O’Hearn, and M. Parkinson. Permission accounting in separation logic. In *POPL ’05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 259–270, New York, NY, USA, 2005. ACM.
2. J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis: 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 55–72, Berlin, Heidelberg, New York, 2003. Springer.
3. C. Calcagno, P. W. O’Hearn, and H. Yang. Local action and abstract separation logic. In *LICS ’07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 366–378, Washington, DC, USA, 2007. IEEE Computer Society.
4. A. Hobor. *Oracle Semantics*. PhD thesis, Princeton University, 2008.
5. A. Hobor, A. W. Appel, and F. Zappa Nardelli. Oracle semantics for concurrent separation logic. In *Proc. European Symp. on Programming (ESOP 2008)*, volume 4960/2008, pages 353–367, 2008.
6. S. S. Ishtiaq and P. W. O’Hearn. Bi as an assertion language for mutable data structures. In *POPL ’01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 14–26, New York, NY, USA, 2001. ACM.
7. M. Parkinson. *Local Reasoning for Java*. PhD thesis, University of Cambridge, 2005.
8. D. J. Pym, P. W. O’Hearn, and H. Yang. Possible worlds and resources: the semantics of BI. *Theor. Comput. Sci.*, 315(1):257–305, 2004.
9. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS ’02: Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, Washington, DC, USA, 2002. IEEE Computer Society.