A Logical Mix of Approximation and Separation

Aquinas Hobor[1], Robert Dockins[2], and Andrew W. Appel[2]

[1] National University of Singapore hobor@comp.nus.edu.sg
[2] Princeton University {rdockins,appel}@cs.princeton.edu

Abstract. Semantic models can use indirection when the naïve semantic definitions contain a contravariant circularity, and substructure when one wishes to track resource accounting. If a model uses indirection, then its logic must reason about the resulting approximation; if a model contains substructure, then its logic often contains notations of separation. We show how to build program logics for settings involving approximation and/or separation. Our work is machine checked in Coq and available as part of the Mechanized Semantic Library.

1 Introduction

We are interested in building program logics for large-scale languages and mechanizing them in proof assistants. We are particularly interested in integrating two features which have proven useful for modeling complex language features: higher-order stores and separation. Higher-order stores are used to reason about certain kinds of recursion that involve mutable state; many common language features exhibit this kind of state-related recursion, *e.g.*, ML references, Pthreads-style locks, and function pointers. Separation is an orthogonal feature which helps reasoning about some of the complications introduced by an addressable memory such as pointer aliasing.

We model the assertion language of the program logic semantically via a Kripke semantics. That is, formulae in the assertion language identified with metalogic propositions over a set of *worlds*, which are some abstraction of the program states. This is a common approach when mechanizing program logics, even among researchers who choose to model the judgments of the program logic syntactically. [Nip02]

The classic example where one desires higher-order stores is that of ML-style references. [Ahm04] It is convenient in certain styles of type soundness proofs for languages with ML references to be able to have the program heap store not just the *dynamic* data generated by the program (i.e., the actual value in the reference) but also *static* data about the type of the reference. Then types are modeled as predicates which judge program values and program heaps. This leads directly to a situation where we want to store predicates (i.e. types) in the program heap, the hallmark of higher-order stores.

Separation logic has recently become quite popular in the programming languages community. [Rey02] Separation logic introduces a substructural connective called separating conjunction, $P * Q$, which means that P and Q hold on "disjoint" areas of the world. A simple and popular model for separation logic is to take worlds as partial maps from addresses to values (i.e., heaps) and say that two heaps are disjoint if they define disjoint sets of addresses. The formulae of separation logic are particularly

adept at describing inductively-defined data structures such as lists and trees. This ability has prompted research into shape-analysis, whereby one attempts to automatically infer datastructure invariants using the vocabulary of separation logic with the intent of deriving more precise invariants than would be possible with traditional techniques. [DOY06,GBCS07,CDOY09]

When defining a program logic, the choice of which worlds to use in the assertion semantics depends strongly on the problem domain, *i.e.*, the particular language being modeled. Much previous work has focused on constructing these complicated types of worlds and using the derived logic to prove some theorem of interest (often a soundness result). [HDA10,Ahm04,DHA09,COY07] However, the important step of building the logic on top of the worlds is often given short shrift. A reader is left with the general impression that once the underlying model is in place, building the logic on top is straightforward. Unfortunately, this is not always the case.

Here we fill in the missing piece by explaining how to build sophisticated logics on top of clean axiomatizations. We construct a general framework for defining assertion languages containing approximation and separation—that is, a logic for worlds that contain higher-order stores and substructure. For higher-order stores, we follow Appel et al. by constructing a Gödel-Löb modal logic of approximation [AMRV07] to simplify the task of using a step-indexed models. Separation appears in the form of separation logic [Rey02], and aids reasoning about languages with pointer manipulation.

In this paper, we largely abstract away from the details of any particular language, and thus we hold the choice of worlds abstract as well. Instead, we will focus on axiomatizing what features worlds must have in order to support higher-order stores and separation, and showing how one can then build a powerful assertion logic containing both these features. However, many domains of interest contain only one of these rather than both together; indeed some domains do not contain either. Our constructions should work in all four settings in a modular way to prevent unnecessary duplication since maintaining parallel code bases in a mechanized setting is painful.

One can imagine several ways to manage this modularity. We have chosen a "stacked" approach in which we first axiomatize how our worlds become more approximate in §2, and show how to satisfy our axioms for settings wherein our worlds have meaningful approximation. If the domain of interest does not have any interesting approximation behavior (*e.g.*, a basic type system or separation logic), then we give methods for adding trivial approximation behavior so that the rest of our framework will still work. After defining the basic operators of our logic in §2.4, we define a multimodal layer on top in §3 to build smooth and modular logical framework for reasoning in the presence of approximation. In §4 we explain how to model and use the equirecursive operator μ.

Once we have specified how approximation should be handled, we specify the substructural properties of our worlds by forming a separation algebra in §5 as in [DHA09,COY07]. If our worlds have no interesting separation structure, this step can be omitted, or we can alternately provide a dummy implementation.

Our primary interest is in settings that combine both approximation and separation. In §6 we characterize the relationship between these properties and prove that the standard connectives of separation logic mix well with our logic of approximation. We also show how our multimodal framework adds value in a separation logic.

In §7, we show how one can use indirection theory to satisfy all of our approximation and separation axioms simultaneously in a nontrivial context.

Implementation. Our constructions and proofs are machine-checked in Coq, and made freely available as part of the Mechanized Semantic Library. Our mechanization contains a certain amount of "black magic Coqery" (*e.g.*, typeclasses, implicit coercions) to ensure that it slides together smoothly and works cleanly from the perspective of using the logic. From time to time we will mention a few design choices that enable simpler mechanical definitions/proofs, but readers particularly interested in this aspect of the result should consult the mechanization. Our results are available at:

`http://msl.cs.princeton.edu/`

Numbering convention. In this presentation we present three classes of equations: *definitions*, numbered with roman numerals; Coq-verified *theorems*, which we enumerate with arabic numerals; and *axioms in a given interface*, enumerated with letters. Many models can satisfy a given interface; one must prove the axioms from its construction.

Constructiveness. In addition to the base axioms of CiC, our framework depends only on the axioms of dependent and propositional extensionality; we do not require, *e.g.*, the axiom of choice. In our presentation we will sweep such issues under the rug.

2 A Logic of Approximation

Here we present the framework of our Gödel-Löb logic of approximation. The formulae of the logic will be identified with predicates on worlds that are *hereditary* with respect to an approximation relation. This simple base will allow us to build a powerful intuitionistic logic into which we can later fit the modal and substructural features.

2.1 Hereditary scaffolding

We assume the existence of a set of *worlds* $\mathbb{W}$, whose precise construction depends on the domain of interest; see [HDA10, §2] for seven examples drawn from various program logics. Given a function P from worlds $\mathbb{W}$ to truth values $\mathbb{T}$ (*e.g.*, $\mathbb{T} \equiv \texttt{Prop}$ in Coq) and a relation R between worlds, we say that P is *hereditary over* R when, if P holds on some world w, then it also holds on all worlds reachable from w through R:

$$\mathsf{hereditary}(P, R) \quad \equiv \quad \forall w, w'.\ P(w) \to (wRw') \to P(w') \qquad (i)$$

That is, P is hereditary over R when P is stable under R. We assume that our worlds come with two operations for axiomatizing approximation: "level" $|w| : \mathbb{W} \to \mathbb{N}$ and "approximate" $w \rightsquigarrow w' : \mathbb{W} \rightharpoonup \mathbb{W}$. The intuition is that $|w| = n$ quantifies the "amount of information" in the world w, and approximating w into w' erases (*i.e.*, approximates) some information in w to make it "fit" into level $n-1$. A $\mathsf{predicate}$ $P \in \mathbb{P}$ is a function from states to truth values $\mathbb{T}$ that is hereditary over the approximation relation:

$$\mathbb{P} \quad \equiv \quad \left\{ P \in \mathbb{W} \to \mathbb{T} \;\middle|\; \mathsf{hereditary}(P, \rightsquigarrow) \right\} \qquad (ii)$$

In Coq, we define this type as a dependent pair and use implicit coercions that allow us to use the pair as a function when desired. We introduce the notation $w \models P$ when we wish to emphasize that we are thinking of P as an assertion rather than a function:

$$w \models P \qquad \equiv \qquad P(w) \qquad (iii)$$

We say P entails Q, written $P \vdash Q$, when the truth of P forces the truth of Q:

$$P \vdash Q \qquad \equiv \qquad \forall w.\, (w \models P) \to (w \models Q) \qquad (iv)$$

We write $\rightsquigarrow^*$ and $\rightsquigarrow^+$ for the reflexive and irreflexive transitive closure of the approximate relation, respectively. We say that two worlds w and w' are *fashionable**, written $w \sim w'$, if they contain the same amount of information, *i.e.*, if $|w| = |w'|$.

Connection to intuitionistic logic. Our framework has much in common with Kripke models of intuitionistic logic in that predicates are hereditary over some relation between worlds. We develop this connection further in, *e.g.*, our model for implication.

2.2 Axiomatization of Approximation

What kinds of properties do we require the approximation operations $\rightsquigarrow$ and $|\cdot|$ to have? In fact, our categorization for approximation is quite simple:**

Approximation functional:	$(w \rightsquigarrow w_1') \;\to\; (w \rightsquigarrow w_2') \;\to\; w_1' = w_2'$	(a)
Level total and functional:	$\exists! n.\; \lvert w\rvert = n$	(b)
Level of bottom:	$(\nexists w'.\, w \rightsquigarrow w') \;\to\; \lvert w\rvert = 0$	(c)
Level of approximation:	$(w \rightsquigarrow w') \;\to\; \lvert w\rvert = \lvert w'\rvert + 1$	(d)
Weak unapproximation:	$(\exists w.\, \lvert w\rvert = \lvert w'\rvert + 1) \;\to\; \exists w.\, w \rightsquigarrow w'$	(e)

We require that approximation be functional *(a)* and that the level operation be a total function *(b)*. If the world w cannot be further approximated, the level of w must be 0 *(c)*. If the world w is approximated to w' then the level of w must be 1 larger than the level of w' *(d)*. Finally, we sometimes wish to "unapproximate"—that is, given some world w', we would like to find a world w such that $w \rightsquigarrow w'$; an unapproximation to a given w' only exists if there is some world containing more information than w'.

Three of the most important consequences of axioms *(a)–(e)* are the following:

---

* The name "fashionable" is a play on words from when we used a time-based analogy for levels. A predicate P which holds fashionably is true on every world "now," but maybe not tomorrow.

** To avoid clutter in our presentation, when we write an interface axiom we omit universal quantifications for variables scoped over the entire equation; *e.g.*, axiom *(e)* is actually:

$$\forall w'.\; \left((\exists w.\, |w| = |w'| + 1) \;\to\; \exists w.\, w \rightsquigarrow w' \right)$$

Can't approximate: $|w| = 0 \;\rightarrow\; (\nexists w'.\; w \rightsquigarrow w')$ (1)

Can approximate: $(|w| > 0) \;\rightarrow\; \exists w'.\; w \rightsquigarrow w'$ (2)

Well founded: $\big(\forall w.\; (\forall w'.\; (w \rightsquigarrow w') \rightarrow w' \models P) \rightarrow w \models P\big) \rightarrow \forall w.\; w \models P$ (3)

That is, worlds of level 0 cannot be approximated further; but any world of level greater than 0 can be approximated. Moreover, the approximate relation is well-founded and thus allows proofs by induction over the action of approximation.

2.3 Models

A model is a triple $(\mathbb{W}, \rightsquigarrow, |\cdot|)$ of set of worlds, approximate operation, and level operation such that axioms *(a)–(e)* hold. We present a simple model to give intuition and then a series of *generators* that build complex models from simpler components. We conclude with a nontrivial model generated by *indirection theory*.

Naturals. A very simple model is the naturals, $(\mathbb{N}, \rightsquigarrow_{\mathbb{N}}, |\cdot|_{\mathbb{N}})$, *i.e.,* $\mathbb{W} \equiv \mathbb{N}$. It is simple to define the approximation operations in this setting as follows: $n \rightsquigarrow_{\mathbb{N}} n' \equiv n = n'+1$ and $|n|_{\mathbb{N}} \equiv n$. Axioms *(a)–(e)* follow directly from these definitions.

Generators. Showing that a particular model satisfies a collection of axioms is not always easy. A generator for a collection of axioms such as *(a)–(e)* is a method for constructing models for those axioms in a modular way by combining previous models in well-behaved ways. This is a particularly valuable technique in mechanized frameworks wherein small changes to the definitions can require significant amount of repair work. We use generators over a variety of axiom sets to allow rapid construction of models. From time to time we discover we are in some new setting and in that case our first task is to define a new generator so that if we encounter that setting again we can apply our new generator immediately. Our generators for the approximation axioms are:

- *Trivial.* Given any set of worlds $\mathbb{W}$, we can define the *trivial model* $(\mathbb{W}, \rightsquigarrow_0, |\cdot|_0)$ by setting $w \rightsquigarrow_0 w' \equiv \bot$ and $|w|_0 \equiv 0$. We stated axiom *(e)* delicately to enable the trivial model, since we want neither approximation nor unapproximation. Note that in the trivial model, all predicates are automatically hereditary.
- *Product.* Given a model $(\mathbb{W}, \rightsquigarrow, |\cdot|)$ and some other set S, we can define the *product model* $(\mathbb{W} \times S, \rightsquigarrow_{\mathbb{W}\times S}, |\cdot|_{\mathbb{W}\times S})$ by defining approximate and level as follows:

  $$(w, s) \rightsquigarrow_{\mathbb{W}\times S} (w', s') \;\equiv\; (s = s') \wedge (w \rightsquigarrow w') \quad \text{and} \quad |(w, s)|_{\mathbb{W}\times S} \;\equiv\; |w|.$$

  Axioms *(a)–(e)* follow directly from the fact that they hold on $\mathbb{W}$.
- *Bijection.* Given a model $(\mathbb{W}, \rightsquigarrow, |\cdot|)$, some other set S, and a bijection $f : \mathbb{W} \rightarrow S$, we can define the *bijection model* $(S, \rightsquigarrow_f, |\cdot|_f)$ by setting

  $$s \rightsquigarrow_f s' \;\equiv\; f^{-1}(s) \rightsquigarrow f^{-1}(s') \quad \text{and} \quad |s|_f \;\equiv\; |f^{-1}(s)|.$$

  Axioms *(a)–(e)* follow because f is a bijection and because the axioms hold on $\mathbb{W}$.

Although we only define a few generators here, we have found that they are sufficient for a large number of settings. One typically splits worlds into parts with trivial and nontrivial approximation behavior and combines the two using the product constructor, perhaps defining a bijection to a form more convenient for the remainder of one's proof. The trivial model is useful in most cases when the set of worlds does not have interesting approximation behavior; the exception is when one wishes to use the recursion operator μ defined in §4 since μ requires nontrivial approximation. In this case, the product model is useful in conjunction with the above model for the naturals $(\mathbb{N}, \rightsquigarrow_{\mathbb{N}}, |\cdot|_{\mathbb{N}})$ to add *non-trivial* approximation behavior to an arbitrary set of worlds $\mathbb{W}$.

Indirection theory. The flagship non-trivial model for our approximation axioms is given by indirection theory [HDA10]. Indirection theory produces approximate solutions to a class of recursive domain equations defined by the pseudoequation:

$$K \quad \approx \quad F((K \times O) \to \mathbb{T})$$

Here F is a covariant functor, O is some "other" noncircular data, and K is the object one wishes to model. A cardinality argument shows that this pseudoequation has no solutions in set theory. Indirection theory approximates a solution by constructing a type K (called the *knot*) and a model $(K, \rightsquigarrow_K, |\cdot|_K)$ that satisfies axioms *(a)–(e)*. Our current construction of K is similar to the one given in [HDA10, §8] but we have enhanced it so that all predicates contained in a knot are hereditary [ADH10, `knot_hered.v`]. We use the product constructor to build the related model $(K \times O, \rightsquigarrow_{K\times O}, |\cdot|_{K\times O})$ and define $\mathbb{P}$ as the set of hereditary functions over $\rightsquigarrow_{K\times O}$ as in definition *(ii)*.

Indirection theory also constructs two functions, $\mathsf{squash} : \mathbb{N} \times F(\mathbb{P}) \to K$ and $\mathsf{unsquash} : K \to \mathbb{N} \times F(\mathbb{P})$ whose behavior is given by the following set of equivalences:

$$\begin{array}{lcl} \mathsf{squash}(\mathsf{unsquash}(k)) & = & k \\ \mathsf{unsquash}(\mathsf{squash}(n, F)) & = & (n, \mathsf{fmap}\ \mathsf{approx}_n\ F) \end{array}$$

That is, $\mathsf{squash} \circ \mathsf{unsquash}$ is the identity function, and $\mathsf{unsquash} \circ \mathsf{squash}$ is a kind of approximation function. The fmap function transforms $F : F(\mathbb{P})$ by locating all of the predicates P inside F and replacing them with $\mathsf{approx}_n(P)$, defined as:

$$\mathsf{approx}_n(P) \ \in \ \mathbb{P} \qquad \equiv \qquad \lambda w. \begin{cases} P(w) & |w|_{K\times O} \ < \ n \\ \bot & |w|_{K\times O} \ \geq \ n \end{cases}$$

The relationship between squash-unsquash and $(K, \rightsquigarrow_K, |\cdot|_K)$ is given by:

$$\begin{array}{lcl} |k| & = & (\mathsf{unsquash}(k)).1 \\ k \rightsquigarrow k' & \leftrightarrow & \mathsf{let}\ (n, F) = \mathsf{unsquash}(k)\ \ \mathsf{in}\ \ (n > 1) \ \wedge \ k' = \mathsf{squash}(n-1, F) \end{array}$$

The level of k is equal to the first projection of k's unsquashing and approximation is equivalent to unsquashing and then resquashing to the next lower level. Axioms *(a)–(d)* follow directly; for *(e)*, unsquash and then resquash to the next *higher* level.

We have used indirection theory to reason about first-class locks in a concurrent program [Hob08]; mutable references in the polymorphic λ-calculus; and program termination in a setting with function pointers and semantic `assert` statements [DH10].

2.4 Hereditary Base Logic

Truth constant:	$w \models \top$	$\equiv$	$\top$	(*v*)
Falsehood constant:	$w \models \bot$	$\equiv$	$\bot$	(*vi*)
Conjunction:	$w \models P \wedge Q$	$\equiv$	$(w \models P) \wedge (w \models Q)$	(*vii*)
Disjunction:	$w \models P \vee Q$	$\equiv$	$(w \models P) \vee (w \models Q)$	(*viii*)
Impredicative universal:	$w \models \forall x : \tau.\ P(x)$	$\equiv$	$\forall x : \tau.\ w \models P(x)$	(*ix*)
Impredicative existential:	$w \models \exists x : \tau.\ P(x)$	$\equiv$	$\exists x : \tau.\ w \models P(x)$	(*x*)
Implication:	$w \models P \Rightarrow Q$	$\equiv$	$\boxed{\begin{array}{l}\forall w'.\ (w \rightsquigarrow^* w') \rightarrow \\ \quad (w' \models P) \rightarrow (w' \models Q)\end{array}}$	(*xi*)
Negation:	$\neg P$	$\equiv$	$P \Rightarrow \bot$	(*xii*)

Given a model of approximation, we can now give semantic definitions for the operators of our base intuitionistic logic, which includes the usual propositional connectives as well as powerful higher-order quantification. Except for implication, each definition consists simply of a direct lifting of the underlying metalogic operator. These can be proved hereditary easily from the assumption that the subformulae are hereditary. In contrast, implication requires that the hereditary assumption be baked in; in the vast majority of cases (including all cases in which the implication was already hereditary) this does not change the meaning of the resulting formulae. The resulting model is exactly a Kripke model of intuitionistic logic. The standard intuitionistic proof theory (e.g., introduction and elimination rules) can all be proved as lemmas from these definitions.

It is worth noting that the τ occuring above in the definition of quantification is allowed to range over all the types of the metalogic, including the type predicate itself; this makes the quantifiers *impredicative*. In contrast, a predicative quantifier would only be allowed to quantify over objects that are smaller according to some stratification, which turns out to be a significant technical restriction. Modeling certain programming language features, such as function closures, requires the stronger impredicative style of quantification, which we provide.

3 The Very Model of a Modern Multimodal Logic

Appel *et al.* [AMRV07] showed how to reason about the action of approximation using modal logic; we go further using the *multimodal* approach outlined in [DAH08]. A *modality* $M \in \mathbb{M}$ is a binary relation that commutes with the approximation relation $\rightsquigarrow$:

$$\mathbb{M} \equiv \left\{ M \in \Sigma \rightarrow \Sigma \rightarrow \mathbb{T} \;\middle|\; \left(\exists \sigma'.(\sigma \rightsquigarrow \sigma') \wedge (\sigma' M \sigma'')\right) \leftrightarrow \left(\exists \sigma'.(\sigma M \sigma') \wedge (\sigma' \rightsquigarrow \sigma'')\right) \right\} \quad (xiii)$$

Most "reasonable" relations one would like to define are modalities. We have seen four approximation relations: approximate $\rightsquigarrow$ and its reflexive $\rightsquigarrow^*$ and irreflexive $\rightsquigarrow^+$ transitive closures, and the same-level relation fashionably $\sim$; all four are modalities:

$$\{\rightsquigarrow, \rightsquigarrow^*, \rightsquigarrow^+, \sim\} \quad \subset \quad \mathbb{M} \tag{4}$$

The point of characterizing modalities is that we can then define modal operators parameterized by various modalities.

Necessarily:	$\sigma \models \Box_M\ P$	$\equiv$	$\forall \sigma'.\ (\sigma M \sigma') \to (\sigma' \models P)$	(*xiv*)
Hypothetically:	$\sigma' \models \Diamond_M\ P$	$\equiv$	$\exists \sigma.\ (\sigma M \sigma') \wedge (\sigma \models P)$	(*xv*)

Note we use the standard definition of the universal modality $\Box_M$, but our definition of the existential modality $\Diamond_M$ is backwards from what one might expect; indeed, we use the "proof-theoretic" dual discussed by Restall [Res00] as opposed to the more familiar boolean dual. We choose to work with the "backward" existential modality because the commutativity restrictions from definition *(xiii)* prove that this modality is hereditary, and in this sense, the modalities belong together. To get the usual boolean dual, one requires that the inverse relation commutes with approximation.

One of the major advantages of identifying and using modal operators is that there are a variety of useful rules and equations that apply to all modal operators. A few of these are listed below.

$$\Diamond_M\ P \vdash Q \quad = \quad P \vdash \Box_M Q \tag{5}$$

$$\Box_M\ (P \Rightarrow Q) \quad \vdash \quad \Box_M\ P \Rightarrow \Box_M Q \tag{6}$$

$$\Box_M\ (P \wedge Q) \quad = \quad \Box_M\ P \wedge \Box_M\ Q \tag{7}$$

$$\Diamond_M\ (P \vee Q) \quad = \quad \Diamond_M\ P \vee \Diamond_M\ Q \tag{8}$$

$$\Box_M \big(\forall x : \tau.\ P(x)\big) \quad = \quad \forall x : \tau.\ \Box_M\ P(x) \tag{9}$$

$$\Diamond_M \big(\exists x : \tau.\ P(x)\big) \quad = \quad \exists x : \tau.\ \Diamond_M\ P(x) \tag{10}$$

Lemma (5) gives the characteristic relationship between the $\Box$ modality and its associated dual $\Diamond$ modality. Readers familiar with modal logics will recognize (6) as axiom K, which is characteristic the "normal" modal logics. We prefer equalities (when they can be achieved) to implications because they allow us to use substitution tactics in mechanized proofs, (*e.g.*, `rewrite` in Coq) which is significantly more convenient than introducing a cut.

Given the data we have about worlds and approximation at this point, we can define two important modal operators which capture some of the important aspects of the approximation model.

Approximately:	$\triangleright P$	$\equiv$	$\Box_{\rightsquigarrow^+} P$	(*xvi*)
Fashionably:	$\bigcirc P$	$\equiv$	$\Box_{\sim}\ P$	(*xvii*)

The approximation modality $\triangleright$ is especially important because it mediates the action of approximation. It interacts in a significant way with both the key Gödel-Löb induction rule and with the recursion operator described in §4. The fashionability modality also interacts in a strong way with recursion. Because of the special relationship $\rightsquigarrow$ has with all the formulae of the logic, $\triangleright$ enjoys some additional properties.

$$\triangleright\,(\Box_M\ P) \quad = \quad \Box_M\ (\triangleright\ P) \tag{11}$$

$$\triangleright\,(P \Rightarrow Q) \quad = \quad \triangleright\,P \Rightarrow \triangleright\,Q \tag{12}$$

$$\triangleright\,(P \vee Q) \quad = \quad \triangleright\,P \vee \triangleright\,Q \tag{13}$$

$$Q \wedge \triangleright\,P \vdash P \quad \to \quad Q \vdash P \tag{14}$$

Lemma (11) shows that $\triangleright$ commutes with every $\Box$ modality; this is a consequence of the validity condition for modal operators. Lemma (12) shows that $\triangleright$ enjoys a stronger form of (6). Lemma (14), called the Gödel-Löb rule, is especially notable because it embodies a kind of induction principle. It says that we can prove that Q entails P if we can show the (apparently) weaker statement that $Q \wedge \triangleright P$ entails P; here $\triangleright P$ is the induction hypothesis.

4 Recursion

In addition to its other benefits, the approximation structure baked into our logic gives us a powerful way to define recursive predicates. Suppose we have a predicate function F : $\mathsf{predicate} \to \mathsf{predicate}$; then we can construct the recursive predicate μF : $\mathsf{predicate}$ satisfying the usual fixpoint equation $\mu F = F(\mu F)$ provided that F is *contractive*. Before we can formally define contractiveness we need a few additional definitions.

Recall from above the "fashionably" modality $\bigcirc\ P \equiv \Box_{\sim}\ P$. The underlying relation $w \sim w'$ holds iff $|w| = |w'|$, so $\bigcirc\ P$ holds when P holds in all worlds of the same level. Using $\bigcirc$, we define a stronger form of implication called "subtyping."

$$P \subseteq Q \quad \equiv \quad \bigcirc\,(P \Rightarrow Q) \tag{xviii}$$

Subtyping is quite a bit stronger than regular implication because the only information it can "see" is the level of the current world. However, it is somewhat weaker than unconditional entailment. That is, if $w \models P \subseteq Q$ it might not be the case that $P \vdash Q$.

We say that P and Q are *equivalent* and write $P \cong Q$ iff $P \subseteq Q$ and $Q \subseteq P$. The intuition is that $w \models P \cong Q$ holds if P and Q are indistinguishable on worlds of level w and smaller. In other words, any world that separates P from Q must have a level greater than $|w|$.

We say that F is contractive iff:

$$\forall P, Q.\ \ \triangleright\,(P \cong Q) \vdash F(P) \cong F(Q) \tag{xix}$$

What does this mean? Roughly, it means that every time you iterate the predicate function F, it "consumes" one level of approximation before using its argument. Usually, this means that the definition of F contains a $\triangleright$ operator guarding the occurrence of its argument.

What all this means is that we can define μ as a finite number of iterations of F:

$$w \models \mu F \qquad \equiv \qquad w \models F^{|w|}(\bot) \tag{xx}$$

Here F^n means F iterated n times. The key point is that as long as F is contractive then we can prove the defining fixpoint theorem for μ:

$$\mu F \qquad = \qquad F(\mu F) \tag{15}$$

Note that in the end we get a strong fixpoint theorem such that μF is simply *equal* to its one-step unfolding, which makes this a form of *equirecursion*. In contrast, systems with *isorecursion* typically require some computational step to allow the folding and

unfolding of recursive definitions. Equirecursion is more convenient for our purposes because it allows us to use the rewriting facilities of the proof assistant, and also because it helps to decouple the semantics of the assertion logic from the (typically operational) semantics of the language.

5 Separation Algebras

Separation algebras are mathematical structures used to model separation logic. They provide the notion of disjoint merging that is central to the meaning of the operators of separation logic. We use a variant called a disjoint multi-unit separation algebra (hereafter just "DSA") [DHA09]. Briefly, a DSA is a set S and an associated three-place partial *join relation* $\oplus$, written $x \oplus y = z$, such that the join relation satisfies:

Functional:	$(x \oplus y = z_1) \;\rightarrow\; (x \oplus y = z_2) \;\rightarrow\; z_1 = z_2$	(f)
Commutative:	$x \oplus y \;=\; y \oplus x$	(g)
Associative:	$x \oplus (y \oplus z) \;=\; (x \oplus y) \oplus z$	(h)
Cancellative:	$(x_1 \oplus y = z) \;\rightarrow\; (x_2 \oplus y = z) \;\rightarrow\; x_1 = x_2$	(i)
Units:	$\forall x.\ \exists u_x.\ x \oplus u_x = x$	(j)
Disjointness:	$(x \oplus x = y) \;\rightarrow\; x = y$	(k)

These axioms define a structure that is like a commutative monoid in many ways, except that $\oplus$ is allowed to be a partial operation. The partiality is important, because it encodes disjointness. If $x \oplus y = z$, then x and y are disjoint, by definition.

Hidden in these axioms is the idea of an *identity*. We say x is an identity if whenever $x \oplus y = z$, then $y = z$. One fundamental property of identities is that x an identity if and only if $x \oplus x = x$. The units axiom *(j)* asserts the existence of (possibly many) identities. It is a consequence of the axioms that each element must have a *unique* identity associated with it.

In the following section we shall see how to use a separation algebra to build a separation logic. For the remainder of this section, we will briefly touch on some example DSAs and constructions for building more complicated ones.

5.1 Models

A model of a separation algebra is a set of worlds $\mathbb{W}$ together with a join relation $\oplus$ satisfying axioms *(f)–(k)*. We give two trivial examples, followed by a series of simple generators, and conclude with some nontrivial generators and examples.

Examples and generators. The DSA axioms are well-behaved in the sense that they are easily propagated across a variety of useful constructions. In our work we have used the following, all of which are already implemented in Coq to enable rapid development:

– *Discrete.* Given a set S, define the *discrete DSA* $(S, \oplus_=)$ by defining

$$s_1 \;\oplus_=\; s_2 \;=\; s_3 \qquad\equiv\qquad s_1 \;=\; s_2 \;=\; s_3$$

Every element joins only with itself and is an identity. Axioms *(f)–(k)* follow.

– *Option.* Given a set S, define the *option DSA* $(S_?, \oplus_?)$ by setting $S_? \equiv \mathsf{None} + \mathsf{Some}(s)$ and the join relation $\oplus_?$ as the least relation satisfying (where $s_? \in S_?$):

$$\begin{array}{rcrcl} \mathsf{None} & \oplus_? & s_? & = & s_? \\ s_? & \oplus_? & \mathsf{None} & = & s_? \end{array}$$

The $\oplus_?$ relation includes $\mathsf{None} \oplus_? \mathsf{None} = \mathsf{None}$. Axioms *(f)–(k)* follow easily.

– *Products.* If we are given two DSAs $(A, \oplus_A)$ and $(B, \oplus_B)$, we can define the *product DSA* $(A \times B, \oplus_{A\times B})$ componentwise by setting:

$$(a_1, b_1) \ \oplus_{A\times B} \ (a_2, b_2) \ = \ (a_3, b_3) \quad \equiv \quad (a_1 \oplus_A a_2 = a_3) \wedge (b_1 \oplus_B b_2 = b_3)$$

Axioms *(f)–(k)* follow directly from the same axioms on A and B.

– *Functions.* Given a set A and a DSA $(B, \oplus_B)$, we can define the *function DSA* $(A \to B, \oplus_{A\to B})$ by lifting the DSA on B pointwise as follows:

$$f \ \oplus_{A\to B} \ g \ = \ h \qquad \equiv \qquad \forall a.\, \big(f(a) \ \oplus_B \ g(a) \ = \ h(a)\big)$$

Axioms *(f)–(k)* follow directly from the axioms on B.

– *Bijection.* Given a DSA $(A, \oplus_A)$, a set B, and a bijection $f : A \to B$, we can define the *bijection DSA* $(B, \oplus_f)$ by setting

$$b_1 \ \oplus_f \ b_2 \ = \ b_3 \qquad \equiv \qquad f^{-1}(b_1) \ \oplus_A \ f^{-1}(b_2) \ = \ f^{-1}(b_3)$$

Axioms *(f)–(k)* follow because f is a bijection and the axioms hold on A.

The previous generators are simple but very useful. For example, if A is a set of addresses and V a set of values, then the archetypical example of partial program heaps is given by the DSA $(A \to (V_?), \oplus_{A\to(V_?)})$, using the function and option generators. We have a large number of other generators in our toolkit: void, unit, discrete, disjoint sums, lists, subset, lift, Π-types, Σ-types, finite partial maps, and lattices; a number of these are described in some detail in [DHA09]. Here we explain another generator, similar in some ways to the bijection DSA covered above but more general:

– *Section–retraction.* The section–retraction generator is a bit tricky. Suppose we have a DSA $(B, \oplus_B)$. A function $h : B \to B$ is a *join homomorphism* when:

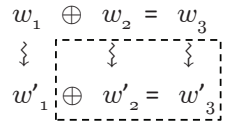
$$b_1 \ \oplus_B \ b_2 \ = \ b_3 \qquad \to \qquad h(b_1) \ \oplus \ h(b_2) = \ h(b_3) \qquad \textbf{(xxi)}$$

That is, joining is preserved by h. Now suppose we have a set A and a section–retraction pair: two functions $f : A \to B$ and $g : B \to A$ such that $g \circ f$ is the identity function on A; note that in any section–retraction pair f is automatically injective while g is automatically surjective. Suppose further that $f \circ g : B \to B$ is a join homomorphism. Define the *section–retraction DSA* $(A, \oplus_{\langle f,g\rangle})$ by setting:

$$a_1 \ \oplus_{\langle f,g\rangle} \ a_2 \ = \ a_3 \qquad \equiv \qquad f(a_1) \ \oplus_B \ f(a_2) \ = \ f(a_3)$$

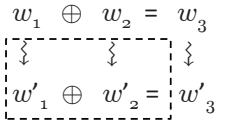
In other words, we take the separation structure induced on the preimage of f. Axioms *(f)*, *(i)*, and *(k)* follow directly from the injectivity of f and the underlying

$$(w_1 \oplus w_2 = w_3) \to (w_1 \rightsquigarrow w_1') \to$$
$$\exists w_2', w_3'.\ (w_1' \oplus w_2' = w_3') \wedge (w_2 \rightsquigarrow w_2') \wedge (w_3 \rightsquigarrow w_3')$$



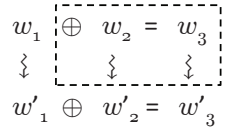
(l)

$$(w_1 \oplus w_2 = w_3) \to (w_3 \rightsquigarrow w_3') \to$$
$$\exists w_1', w_2'.\ (w_1' \oplus w_2' = w_3') \wedge (w_1 \rightsquigarrow w_1') \wedge (w_2 \rightsquigarrow w_2')$$



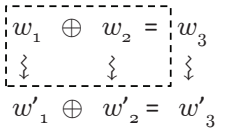
(m)

$$(w_1' \oplus w_2' = w_3') \to (w_1 \rightsquigarrow w_1') \to$$
$$\exists w_1, w_2.\ (w_1 \oplus w_2 = w_3) \wedge (w_2 \rightsquigarrow w_2') \wedge (w_3 \rightsquigarrow w_3')$$



(n)

$$(w_1' \oplus w_2' = w_3') \to (w_3 \rightsquigarrow w_3') \to$$
$$\exists w_1, w_2.\ (w_1 \oplus w_2 = w_3) \wedge (w_1 \rightsquigarrow w_1') \wedge (w_2 \rightsquigarrow w_2')$$



(o)

Fig. 1. Axioms for Mixing Separation and Approximation

axioms on $\oplus_B$. Axiom *(g)* is even simpler and is direct from the commutativity of $\oplus_B$. The associativity *(h)* and units *(j)* axioms are tougher; both require that $g \circ f$ is the identity, $f \circ g$ is a join homomorphism, and the underlying axioms on $\oplus_B$.

The significance of the section–retraction generator is that it will be just what is needed to handle the unsquash–squash pair constructed by indirection theory.

6 Mixing Separation and Approximation

Once we have defined the separation structure on a set of worlds, we are nearly ready to define the operators of separation logic. However, to interface with the approximation features of the logic, we need some additional axioms which ensure that separation and approximation can play well together in the same sandbox (see figure 1). These four axioms have the flavor of commuting diagrams; we require that the approximation relation and separation and "slide around" each other cleanly. (There are a total of six possible cases, but two are subsumed by commutativity). These axioms let us prove the heredity of the operators of separation logic and to show certain useful results about the commutativity of approximation operators with separation operators.

Now we can give the definitions of the standard operators of separation logic.

Empty: $\sigma \models \mathsf{emp} \equiv \mathsf{identity}\ \sigma$ (*xxii*)

Separation: $\sigma \models P * Q \equiv \exists \sigma_1, \sigma_2.\ (\sigma_1 \oplus \sigma_2 = \sigma) \wedge (\sigma_1 \models P) \wedge (\sigma_2 \models Q)$ (*xxiii*)

Seplication: $\sigma_1 \models P \mathrel{-\!\!*} Q \equiv \forall \sigma_1', \sigma_2, \sigma.\ (\sigma_1 \rightsquigarrow^* \sigma_1') \to (\sigma_1' \oplus \sigma_2 = \sigma)$
$\to (\sigma_2 \models P) \to (\sigma \models Q)$ (*xxiv*)

The assertion emp and the separating conjunction $*$ can be shown hereditary by using axioms *(l)* and *(m)*. Notice that the definition of seplication explicitly quantifies over all more approximate worlds, just as does the definition of implication, making it immediately hereditary from the definition. Just as with implication, the semantics takes on an intuitionistic flavor, but in general works exactly as expected.

With these definitions stated, we can easily prove the standard inference rules of separation logic and various equalities among formulae. Note equations (20) and (21); these elegant equations are the result of our insistence that approximation and separation interact smoothly. Their proofs make essential use of axioms *(n)* and *(o)*.

Commutativity:	$P * Q$	$=$	$Q * P$	(16)
Associativity:	$(P * Q) * R$	$=$	$P * (Q * R)$	(17)
Identity:	$\mathsf{emp} * P$	$=$	P	(18)
Seplication adjoint:	$(P * Q) \vdash R$	$=$	$P \vdash (Q \mathrel{-\!\!*} R)$	(19)
Approx sepconjunction:	$\triangleright(P * Q)$	$=$	$(\triangleright P * \triangleright Q)$	(20)
Approx seplication:	$\triangleright(P \mathrel{-\!\!*} Q)$	$=$	$(\triangleright P \mathrel{-\!\!*} \triangleright Q)$	(21)
Split sepconjunction:	$(P \vdash Q)$	$\to$	$(R \vdash S) \to (P * R) \vdash (Q * S)$	(22)
Cut seplication:	$(P \vdash Q \mathrel{-\!\!*} R)$	$\to$	$(S \vdash Q) \to (P * S) \vdash R$	(23)

In addition to the standard operators of separation logic, we can define three substructural modalities. First, we say that w_1 is a *substate* of w_2, written $w_1 \preceq w_2$, when

$$w_1 \preceq w_2 \quad \equiv \quad \exists w'.\ w_1 \oplus w' = w_2 \tag{xxv}$$

Informally, w_1 is a smaller state than w_2 because you can add w' to w_1 to get w_2; it corresponds to the *substate* relation with respect to the separation structure. Second, we say that w_1 and w_2 are *orthogonal*, written $w_1 \sharp w_2$, when

$$w_1 \sharp w_2 \quad \equiv \quad \exists w'.\ w_1 \oplus w_2 = w' \tag{xxvi}$$

Two states are orthogonal when they are compatible in the sense that they can join together. Finally, w_1 and w_2 are *substructurally comparable*, written $w_1 \overset{\oplus}{\sim} w_2$, when

$$w_1 \overset{\oplus}{\sim} w_2 \quad \equiv \quad \exists w.\ (w_1 \sharp w) \wedge (w_2 \sharp w) \tag{xxvii}$$

Two worlds are substructurally comparable when there exists some world (typically an identity) that is orthogonal to both of them. We can consider the elements of a DSA as being divided into equivalence classes where there is one class for each unit, and every element with the same unit is in the class. Then $\overset{\oplus}{\sim}$ ranges over all the elements in the same equivalence class.

All of these substructural relations are valid modalities according to the definition from §3. The validity proofs are direct consequence of axioms from Figure 1.

$$\{\preceq, \sharp, \overset{\oplus}{\sim}\} \ \subset \ \mathbb{M} \tag{24}$$

A further consequence is that our substructural modalities are all fashionable:

$$(w_1 \preceq w_2) \vee (w_1 \sharp w_2) \vee (w_1 \overset{\oplus}{\sim} w_2) \ \to \ w_1 \sim w_2 \tag{25}$$

We often find it convenient to express substructural ideas using modalities like these. For example, consider the diamond form of the substate relation; $\Diamond_{\preceq} P$ holds exactly when some substate of the current state satisfies P. In other words, adding $\Diamond_{\preceq}$ makes a predicate invariant under state expansion.[†] A little manipulation shows that:

$$\Diamond_{\preceq} P \;=\; P * \top. \tag{26}$$

7 Separation logics over knots

An important use case (indeed, our motivating use case) for combining approximation with separation are the "knots" of indirection theory. We can quite easily demonstrate that knots satisfy the approximation axioms using the interface provided by indirection theory. However, to define a separation structure on knots, we need to define an appropriate join relation and prove the DSA axioms. The knots provided to clients are *opaque*, which means the client cannot examine the details of the construction. However, the client has provided the critical functor F describing the internal structure of unsquashed knots. We require the client to define a separation structure over F which we then use to induce a separation structure over knots.

We proceed in stages. First we must make the set $\mathbb{N} \times F(\mathbb{P})$ into a DSA. We will require that the client of indirection theory demonstrate that F is a functor on DSAs, *i.e.*, whenever X is a DSA, then $F(X)$ is also a DSA. Furthermore, we require that whenever $f : X \to Y$ is a join homomorphism, then $\mathsf{fmap}\ f : F(X) \to F(Y)$ must also be a join homomorphism. Now we use our generators to construct the DSA $(\mathbb{N} \times F(\mathbb{P}), \oplus_{(\mathbb{N}_=) \times (F(\mathbb{P}_=))})$: that is, we pair up a discrete DSA on $\mathbb{N}$ with the DSA generated by applying F to the discrete DSA on $\mathbb{P}$.

We will use the section–retraction generator to induce a DSA for the set $A \equiv K$ from the above DSA for $B \equiv \mathbb{N} \times F(\mathbb{P})$. Indirection theory gives us the section–retraction pair $(\mathsf{unsquash}, \mathsf{squash})$. It turns out to be quite simple to show that $\mathsf{unsquash} \circ \mathsf{squash}$ is a join homomorphism on B, completing the construction of the DSA for K.

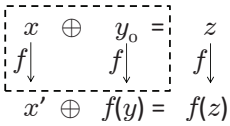
We have two of the ingredients needed for a logic over knots with both separation and approximation. We have the approximation structure and we have a DSA. However, in order to complete the picture we need to prove the distributive axioms from §6.

The two "forward" axioms *(l)* and *(m)* follow easily from the assumption that F is a functor on DSAs. The "backward" axioms *(n)* and *(o)*, however, are more involved. Proving these axioms appears to require additional technical restrictions on the functor F, having to do with "unmapping." The precise statement of these technical requirements is given in Figure 2 and is rather involved. However, proving that particular functors F have this property is usually easy.

Suppose one has a function $f : A \to B$ where A and B are DSAs. We say that f has *left unmappings* when it satisfies axiom *(p)* and *right unmappings* when it satisfies *(q)*. We say a functor F *preserves unmappings* if, whenever f is a join homomorphism with left (right) unmappings, then $\mathsf{fmap}\ f$ has left (right) unmappings.

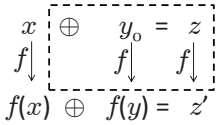
[†] Such predicates were called *intuitionistic* in Reynolds' work on separation logic. [Rey02]

$$x' \oplus f(y) = f(z) \;\; \rightarrow \\ \quad \exists x, y_0.\; x \oplus y_0 = z \;\wedge\; f(x) = x' \;\wedge\; f(y_0) = f(y)$$



(p)

$$f(x) \oplus f(y) = z' \;\; \rightarrow \\ \quad \exists y_0, z.\; x \oplus y_0 = z \;\wedge\; f(y_0) = f(y) \;\wedge\; f(z) = z'$$



(q)

Fig. 2. Left and right unmappings

The existence of unmappings means that f has a weak kind of invertability property, and the preservation of unmappings means that when such a weakly invertable function is applied with fmap, the resulting function is itself weakly invertable.

As with approximation and DSAs, we can show that many standard constructions (when considered as functors) have the property of preserving unmappings. For example, products, disjoint sums, functions and lists all preserve unmappings.

If F preserves unmappings, then we can prove the "unapproximation" axioms *(n)* and *(o)* for knots. The key is to note that the approx function has left and right unmappings, and then lift the unmappings through the functor F using *(p)* and *(q)*. The unmappings of fmap f then provide the required witnesses for axioms *(n)* and *(o)*.

We now have all the pieces necessary to build a separation logic with approximation over the knots of indirection theory. In the final accounting, the client must provide, in addition to the data necessary for indirection theory itself, a proof that F is a functor on DSAs, and an easy technical proof about the preservation of unmappings. From this basic data, a rich logic of separation and approximation is automatically built.

8 Conclusion

We have presented a method for constructing powerful assertion logics using a Kripke semantics over a set of *worlds*. We have given axiomatic interfaces that worlds must satisfy in order to support higher-order stores in the step-indexing style, and to support substrucural features in the style of separation logic. These two features interact in nontrivial ways, and we have further shown how to get an elegant and well-behaved logic by requiring the approximation and separation relations to commute with one another. Finally, we have shown throughout the paper how to construct models of these axiomatic interfaces that support a variety of interesting programming language domains. The proofs and constructions that appear in this paper have been mechanized in Coq and are freely available as part of the Mechanized Semantic Library [ADH10].

Acknowledgements. Aquinas Hobor is supported by a Lee Kuan Yew Postdoctoral Fellowship. Robert Dockins and Andrew W. Appel are supported in part by NSF grant CNS-0910448 and AFOSR grant FA9550-09-1-0138.

References

[ADH10] Andrew Appel, Robert Dockins, and Aquinas Hobor. Mechanized Semantic Library. Available at http://msl.cs.princeton.edu, 2009–2010.

[Ahm04] Amal J. Ahmed. *Semantics of Types for Mutable State*. PhD thesis, Princeton University, Princeton, NJ, November 2004. Tech Report TR-713-04.

[AMRV07] Andrew W. Appel, Paul-Andre Melliès, Christopher D. Richards, and Jerôme Vouillon. A very modal model of a modern, major, general type system. In *Proc. 34th Annual Symposium on Principles of Programming Languages (POPL'07)*, pages 109–122, January 2007.

[CDOY09] Cristiano Calcagno, Dino Distefano, Peter O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. In *Proc. of 36th Annual Symp. on Principles of Programming Languages (POPL)*, pages 289–300, 2009.

[COY07] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok 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, 2007.

[DAH08] Robert Dockins, Andrew W. Appel, and Aquinas Hobor. Multimodal separation logic for reasoning about operational semantics. In *24th Conference on the Mathematical Foundations of Programming Semantics (MFPS XXIV)*, pages 5–20. Springer Electronic Notes in Theoretical Computer Science (ENTCS), 2008.

[DH10] Robert Dockins and Aquinas Hobor. A theory of termination via indirection. Under submission, July 2010.

[DHA09] Robert Dockins, Aquinas Hobor, and Andrew W. Appel. A fresh look at separation algebras and share accounting. In *The 7th Asian Symposium on Programming Languages and Systems*. Springer ENTCS, 2009. To appear.

[DOY06] Dine Distefano, Peter W. O'Hearn, and Hongseok Yang. A local shape analysis based on separation logic. In *Proc. of 12th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 287–302. Springer, 2006.

[GBCS07] Alexey Gotsman, Josh Berdine, Byron Cook, and Mooly Sagiv. Thread-modular shape analysis. In *PLDI '07: 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2007.

[HDA10] Aquinas Hobor, Robert Dockins, and Andrew W. Appel. A theory of indirection via approximation. In *Proc. 37th Annual ACM Symposium on Principles of Programming Languages (POPL'10)*, pages 171–185, January 2010.

[Hob08] Aquinas Hobor. *Oracle Semanatics*. PhD thesis, Princeton University, Princeton, NJ, November 2008.

[Nip02] Tobias Nipkow. Hoare logics for recursive procedures and unbounded nondeterminism. In *Computer Science Logic*, volume 2471/2002 of *LNCS*, pages 155–182. Springer, 2002.

[Res00] Greg Restall. *An Introduction to Substructural Logics*. Routledge, London, England, 2000.

[Rey02] John Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS 2002: IEEE Symposium on Logic in Computer Science*, pages 55–74, July 2002.