

A First Order Logic of Effects

Ian A. Mason*

May 30, 1996

Abstract

In this paper we describe some of our progress towards an operational implementation of a modern programming logic. The logic is inspired by the variable type systems of Feferman, and is designed for reasoning about imperative functional programs. The logic goes well beyond traditional programming logics, such as Hoare's logic and Dynamic logic in its expressibility, yet is less problematic to encode into higher order logics. The main focus of the paper is to present an axiomatization of the base first order theory.

1 Introduction

VTL_{oE} [34, 23, 35, 37, 24] is a logic for reasoning about imperative functional programs, inspired by the variable type systems of Feferman. These systems are two sorted theories of operations and classes initially developed for the formalization of constructive mathematics [12, 13] and later applied to the study of purely functional languages [14, 15]. VTL_{oE} builds upon recent advances in the semantics of languages with effects [16, 19, 28, 32, 33] and goes well beyond traditional programming logics, such as Hoare's logic [7] and Dynamic logic [22] by treating a richer language and expressing more properties. It is close in spirit to Specification Logic [49] and to Evaluation Logic [44].

The underlying programming language of VTL_{oE}, λ_{mk} , is based on the call-by-value lambda calculus extended by the reference primitives `mk`, `set`, `get`. Atoms, references and lambda abstractions are all first class values – they can be bound to lambda variables, stored, and returned from procedures. It can thus be thought of as a fragment of untyped ML or a variant of Scheme. The logic combines the features and benefits of equational calculi as well as program and specification logics. There are three layers. The foundation is the syntax and semantics of λ_{mk} , the underlying term/programming language. The second layer is a first-order theory built on assertions of program equivalence and program modalities called *contextual assertions*. The third layer extends the logic to include class terms, class membership, and quantification over class variables. In this paper we concentrate on the first two layers. The main topic of this paper is the presentation of a Hilbert style formal system for VTL_{oE} and the proof of its completeness.

Contextual assertions were first introduced in [31] as a means for expressing *constraint propagation*. It was quickly realized that they are an essential feature of any language for reasoning about the effects of programs. In our earlier work on axiomatizing imperative features [30, 29, 33] we presented a simple sequent system for proving equations. The introduction rules for the allocation and updating primitives were complicated by ugly side conditions. Conditions so ugly as to make the implementation of the system unfeasible. The main result of this paper is to generalize these previous results to a full first order language with contextual assertions. The crucial point is that contextual assertions eliminate the need for *any* side conditions.

1.1 Overview and Notation

The remainder of this paper is organized as follows. In §2 we introduce the syntax and semantics of the terms of VTL_{oE}. In §3 we introduce the syntax and semantics of the formulas of VTL_{oE}. In §4 we present the proof theory of VTL_{oE}. In §5 we relate the semantics with the proof theory.

*Department of Mathematics & Computer Science, University of New England, Armadale, N.S.W, Australia 2350

We conclude the introduction with a summary of notation. Let X, Y, Y_0, Y_1 be sets. We specify meta-variable conventions in the form: let x range over X , which should be read as: the meta-variable x and decorated variants such as x', x_0, \dots , range over the set X . We use the usual notation for set membership and function application. Y^n is the set of sequences of elements of Y of length n . Y^* is the set of finite sequences of elements of Y . $\bar{y} = [y_1, \dots, y_n]$ is the sequence of length n with i th element y_i . $\mathbf{P}_\omega(Y)$ is the set of finite subsets of Y . $Y_0 \xrightarrow{f} Y_1$ is the set of finite maps from Y_0 to Y_1 . $[Y_0 \rightarrow Y_1]$ is the set of total functions f with domain Y_0 and range contained in Y_1 . We write $\text{Dom}(f)$ for the domain of a function and $\text{Rng}(f)$ for its range. For any function f , $f\{y := y'\}$ is the function f' such that $\text{Dom}(f') = \text{Dom}(f) \cup \{y\}$, $f'(y) = y'$, and $f'(z) = f(z)$ for $z \neq y, z \in \text{Dom}(f)$. $\mathbb{N} = \{0, 1, 2, \dots\}$ is the natural numbers and i, j, n, n_0, \dots range over \mathbb{N} .

2 The Syntax and Semantics of Terms

2.1 Syntax

The syntax of the terms of λ_{mk} is a simple extension of the lambda calculus to include basic atomic data \mathbb{A} , (such as the Lisp booleans `t` and `nil`, and in practice the natural numbers \mathbb{N}), together with a collection of primitive operations, $\mathbb{F} = \bigcup_{n \in \mathbb{N}} \mathbb{F}_n$, where \mathbb{F}_n is the (possibly empty) set of n -ary operations.

Booleans	$\{\mathbf{t}, \mathbf{nil}\} \subseteq \mathbb{A}$
Recognisers	$\mathbb{T} = \{\mathbf{atom?}, \mathbf{cell?}, \mathbf{lambda?}\}$
Unary Operations	$\mathbb{F}_1 = \{\mathbf{mk}, \mathbf{get}\} \cup \mathbb{T}$
Binary Operations	$\mathbb{F}_2 = \{\mathbf{app}, \mathbf{eq}, \mathbf{set}\}$
Ternary Operations	$\mathbb{F}_3 = \{\mathbf{br}\}$

The primitive operations include: the memory operations (`mk, get, set`) for allocating, dereferencing, and updating unary cells; the usual operations for strict branching (`br`) and arithmetic; the recognizing operations (`atom?, cell?, lambda?`) (or characteristic functions using the booleans `t` and `nil`) of their respective domains. We also treat application, `app`, as a binary operation for the sake of uniformity.

Together with the atoms, \mathbb{A} , we assume an infinite set of variables, \mathbb{X} and use these to define, by mutual induction, the set of λ -abstractions, \mathbb{L} , the set of value expressions, \mathbb{V} , the set of value substitutions, \mathbb{S} , the set of expressions, \mathbb{E} , and the set of contexts, \mathbb{C} , as the least sets satisfying the following equations:

Atoms	\mathbb{A}	a ranges over \mathbb{A}
Variables	\mathbb{X}	x, y, z ranges over \mathbb{X}
Lambda Expressions	$\mathbb{L} = \lambda \mathbb{X}. \mathbb{E}$	$\lambda x. e$ ranges over \mathbb{L}
Value Expressions	$\mathbb{V} = \mathbb{X} + \mathbb{A} + \mathbb{L}$	v ranges over \mathbb{V}
Value Substitutions	$\mathbb{S} = \mathbb{X} \xrightarrow{f} \mathbb{V}$	σ ranges over \mathbb{S}
Expressions	$\mathbb{E} = \mathbb{V} + \mathbb{F}_n(\mathbb{E}^n)$	e ranges over \mathbb{E}
Contexts	$\mathbb{C} = \{\bullet\} + \mathbb{X} + \mathbb{A} + \lambda \mathbb{X}. \mathbb{C} + \mathbb{F}_n(\mathbb{C}^n)$	C ranges over \mathbb{C}

λ is a binding operator and free and bound variables of expressions are defined as usual. $\text{FV}(e)$ is the set of free variables of e . A *value substitution* is a finite map σ from variables to value expressions, we let σ range over value substitutions. e^σ is the result of simultaneous substitution of free occurrences of $x \in \text{Dom}(\sigma)$ in e by $\sigma(x)$. We represent the function which maps x to v by $\{x := v\}$. Thus $e^{\{x := v\}}$ is the result of replacing free occurrences of x in e by v (avoiding the capture of free variables in v). Contexts are expressions with holes. We use \bullet to denote a hole. $C[e]$ denotes the result of replacing any holes in C by e . Free variables of e may become bound in this process. $\text{Traps}(C)$ is the set of variables that can actually be trapped in the process of filling the holes in C .

2.2 Abbreviations

In order to make programs easier to read, we introduce some abbreviations.

$f(x)$	\triangleq	$\text{app}(f, x)$	
$\text{let}\{x := e_0\}e_1$	\triangleq	$\text{app}(\lambda x.e_1, e_0)$	
$\text{seq}(e)$	\triangleq	e	
$\text{seq}(e_0, \dots, e_n)$	\triangleq	$\text{let}\{z := e_0\}\text{seq}(e_1, \dots, e_n)$	$z \notin \text{FV}(e_i)$ for $i \leq n$
$\text{if}(e_0, e_1, e_2)$	\triangleq	$\text{app}(\text{br}(e_0, \lambda z.e_1, \lambda z.e_2), \text{nil})$	for z fresh
$\text{not}(e)$	\triangleq	$\text{if}(e, \text{nil}, \text{t})$	
$\text{and}(e_0, e_1)$	\triangleq	$\text{if}(e_0, e_1, \text{nil})$	
$\text{and}(e_0, e_1, \dots, e_n)$	\triangleq	$\text{and}(e_0, \text{and}(e_1, \dots, e_n))$	
$\text{or}(e_0, e_1)$	\triangleq	$\text{if}(e_0, \text{t}, e_1)$	
$\text{or}(e_0, e_1, \dots, e_n)$	\triangleq	$\text{or}(e_0, \text{or}(e_1, \dots, e_n))$	

2.3 Programming Examples

2.3.1 Equality on Cells

To simplify matters later we have omitted equality on cells as a primitive operation. It is however easily definable.

$$\begin{aligned} \text{eq?} &\triangleq \lambda x.\lambda y.\text{if}(\text{and}(\text{cell?}(x), \text{cell?}(y)), \\ &\quad \text{let}\{x_0 := \text{get}(x), y_0 := \text{get}(y)\} \\ &\quad \quad \text{seq}(\text{set}(x, \text{nil}), \\ &\quad \quad \quad \text{set}(y, \text{t}), \\ &\quad \quad \quad \text{let}\{z := \text{get}(x)\} \\ &\quad \quad \quad \quad \text{seq}(\text{set}(x, x_0), \\ &\quad \quad \quad \quad \quad \text{set}(y, y_0), \\ &\quad \quad \quad \quad \quad \quad z)), \\ &\quad \text{nil}) \end{aligned}$$

2.3.2 Landin's Recursion Operator

Since the λ_{mk} -calculus extends the call-by-value λ -calculus the usual call-by-value fixed point combinator is a term in the language. A somewhat different fixed point combinator, that makes use of the reference primitives, is possible:

$$\begin{aligned} \mathbf{Y} &\triangleq \lambda y.\text{let}\{z := \text{mk}(\text{nil})\} \\ &\quad \text{seq}(\text{set}(z, \lambda x.\text{app}(\text{app}(y, \text{get}(z)), x)), \\ &\quad \quad \text{get}(z)) \end{aligned}$$

This version of the fixed-point combinator is essentially identical to the one suggested by Landin [27]. When applied to a functional F of the form $\lambda f.\lambda x.e$, \mathbf{Y} creates a private local cell, z , with contents $G = \lambda x.\text{app}(\text{app}(F, \text{get}(z)), x)$, and returns G . By privacy of z , G is equivalent to $F(G)$ (cf. [32]). Note that this example is typable in the simply typed lambda calculus (for provably non-empty types (cf. [24])). Thus adding operations for manipulating references to the simply typed lambda calculus causes the failure of strong normalization as well as many other of its nice mathematical properties.

2.3.3 Integer Streams

From an abstract point of view, a stream is simply a (possibly infinite) sequence of data [2]. In the λ_{mk} -calculus we can represent streams simply as functional objects. The sequence corresponding to a λ_{mk} -stream is the values returned by repeated application of the object to a fixed (and hopefully irrelevant argument). The simplest example of a non-trivial λ_{mk} -stream is the stream of natural numbers.

$$\begin{aligned} \text{makeStream} &\triangleq \lambda m. \text{let}\{z := \text{mk}(m)\} \\ &\quad \lambda x. \text{let}\{n := \text{get}(z)\} \\ &\quad \quad \text{seq}(\text{set}(z, n + 1), \\ &\quad \quad \quad n) \end{aligned}$$

Here `makeStream` applied to an integer m creates a stream of integers beginning with that integer. The so-created stream when queried (applied to any value) returns the next integer in the stream.

2.3.4 The Sieve of Eratosthenes

A somewhat more interesting example of a stream is the sieve of Eratosthenes [2]. We begin with the functional `filter` which expects an integer, n , and a stream, s and then creates a new stream. This new stream when queried repeatedly calls the stream argument, s , until an integer not divisible by the number argument, n , is returned. This number is then returned as the answer to the query.

$$\begin{aligned} \text{filter} &\triangleq \lambda n. \lambda s. \\ &\quad \lambda x. \text{let}\{m := s(\text{nil})\} \\ &\quad \quad \text{if}(\text{divides?}(n, m) \\ &\quad \quad \quad \text{filter}(n, s)(\text{nil}) \\ &\quad \quad \quad m) \end{aligned}$$

`sieve` is an expression which when evaluated creates a new the sieve of Eratosthenes. This new stream is a stream of the prime numbers. Each time the stream is queried it returns the current prime and updates its local stream to filter with this prime.

$$\begin{aligned} \text{sieve} &\triangleq \text{let}\{sc := \text{mk}(\text{makeStream}(2))\} \\ &\quad \lambda x. \text{let}\{s := \text{get}(sc)\} \text{let}\{p := s(\text{nil})\} \\ &\quad \quad \text{seq}(\text{set}(sc, \text{filter}(p, s)), \\ &\quad \quad \quad p) \end{aligned}$$

2.4 Semantics of Terms

The operational semantics of expressions is given by a reduction relation \mapsto^* on a syntactic representation of the state of an abstract machine, referred to as *computation descriptions*. A state has three components: the current instruction, the current continuation, and the current state of memory. Their syntactic counterparts are *redexes*, *reduction contexts*, and *memory contexts* respectively. Redexes describe the primitive computation steps. A primitive step is either a β_v -reduction or the application of a primitive operation to a sequence of value expressions. The set of redexes, \mathbb{E}_r , is defined as

$$\mathbb{E}_r = \mathbb{F}_n(\mathbb{V}^n)$$

Reduction contexts identify the subexpression of an expression that is to be evaluated next, they correspond to the standard reduction strategy (left-first, call-by-value) of [46] and were first introduced in [18]. The set of reduction contexts, \mathbb{R} , is the subset of \mathbb{C} defined by

$$\mathbb{R} = \{\bullet\} + \mathbb{F}_{m+n+1}(\mathbb{V}^m, \mathbb{R}, \mathbb{E}^n)$$

We let R ranges over \mathbb{R} . An expression is either a value expression or decomposes uniquely into a redex placed in a reduction context. An easy structural induction establishes that if $e \in \mathbb{E}$, then either $e \in \mathbb{V}$ or e can be written uniquely as $R[e']$ where $R \in \mathbb{R}$ and $e' \in \mathbb{E}_r$. The set of memory contexts, \mathbb{M} , is the set of contexts Γ of the form

$$\text{let}\{z_1 := \text{mk}(\text{nil})\} \dots \text{let}\{z_n := \text{mk}(\text{nil})\} \text{seq}(\text{set}(z_1, v_1), \dots, \text{set}(z_n, v_n), \bullet)$$

where $z_i \neq z_j$ when $i \neq j$. We include the possibility that $n = 0$, in which case $\Gamma = \bullet$. We let Γ ranges over \mathbb{M}

We have divided the memory context into allocation, followed by assignment to allow for the construction of cycles. Thus, any state of memory is constructible by such an expression. We can view memory contexts as finite maps from variables to value expressions. Hence we define the domain of Γ (as above) to be $\text{Dom}(\Gamma) = \{z_1, \dots, z_n\}$, and $\Gamma(z_i) = v_i$ for $1 \leq i \leq n$. Two memory contexts are considered the same if they are the same when viewed as functions. Viewing memory contexts and finite maps, we define the modification of memory contexts, $\Gamma\{z := \text{mk}(v)\}$, and the union of two memory contexts, $(\Gamma_0 \cup \Gamma_1)$, in the obvious way. Γ^σ is the result of applying σ to each value in the range of Γ .

The set of computations descriptions (briefly descriptions), \mathbb{D} , is defined to be the set $\mathbb{M} \times \mathbb{E}$. Thus a description is a pair with first component a memory context and second component an arbitrary expression. We do not require that the free variables of the expression be contained in the domain of the memory context. This allows us to define reductions uniformly in parameters that are not touched by the reduction step, and hence to provide a form of symbolic evaluation. We let $\Gamma; e$ ranges over \mathbb{D} . A *closed* description is a description of the form $\Gamma; e$ where $\text{FV}(e) \subseteq \text{Dom}(\Gamma)$. *Value descriptions* are descriptions whose expression component is a value expression, i.e. a description of the form $\Gamma; v$. $(\Gamma; e)^\sigma = \Gamma^\sigma; e^\sigma$.

Definition (\mapsto^*): The reduction relation \mapsto^* is the reflexive transitive closure of \mapsto . The clauses are:

$$\begin{aligned} (\beta_v) \quad & \Gamma; R[\text{app}(\lambda x.e, v)] \mapsto \Gamma; R[e^{x:=v}] \\ (\text{atom}) \quad & \Gamma; R[\text{atom?}(v)] \mapsto \begin{cases} \Gamma; R[\mathbf{t}] & \text{if } v \in \mathbb{A} \\ \Gamma; R[\mathbf{nil}] & \text{if } v \in \mathbb{L} \cup \text{Dom}(\Gamma) \end{cases} \\ (\text{cell}) \quad & \Gamma; R[\text{cell?}(v)] \mapsto \begin{cases} \Gamma; R[\mathbf{t}] & \text{if } v \in \text{Dom}(\Gamma) \\ \Gamma; R[\mathbf{nil}] & \text{if } v \in \mathbb{L} \cup \mathbb{A} \end{cases} \\ (\text{eq}) \quad & \Gamma; R[\text{eq}(v_0, v_1)] \mapsto \begin{cases} \Gamma; R[\mathbf{t}] & \text{if } v_0 = v_1, v_0, v_1 \in \mathbb{A} \\ \Gamma; R[\mathbf{nil}] & \text{else if } v_0, v_1 \in \text{Dom}(\Gamma) \cup \mathbb{L} \cup \mathbb{A} \end{cases} \\ (\text{br}) \quad & \Gamma; R[\text{br}(v_0, v_1, v_2)] \mapsto \begin{cases} \Gamma; R[v_1] & \text{if } v_0 \in (\mathbb{A} - \{\mathbf{nil}\}) \cup \mathbb{L} \cup \mathbb{P} \cup \text{Dom}(\Gamma) \\ \Gamma; R[v_2] & \text{if } v_0 = \mathbf{nil} \end{cases} \\ (\text{mk}) \quad & \Gamma; R[\text{mk}(v)] \mapsto \Gamma\{z := \text{mk}(v)\}; R[z] \quad \text{if } z \notin \text{Dom}(\Gamma) \cup \text{FV}(R[v]) \\ (\text{get}) \quad & \Gamma; R[\text{get}(z)] \mapsto \Gamma; R[v] \quad \text{if } z \in \text{Dom}(\Gamma) \text{ and } \Gamma(z) = v \\ (\text{set}) \quad & \Gamma; R[\text{set}(z, v)] \mapsto \Gamma\{z := \text{mk}(v)\}; R[\mathbf{nil}] \quad \text{if } z \in \text{Dom}(\Gamma) \end{aligned}$$

Note that in the `atom?` and `cell?` rules if one of the arguments is a variable not in the domain of the memory context then the primitive reduction step is not determined. This is also the case in the `eq`, `br`, `get`, and `set` rules.

Definition ($\downarrow \uparrow \downarrow \sim$): A closed description, $\Gamma; e$ is *defined* (written $\downarrow \Gamma; e$) if it evaluates to a value description. A description is *undefined* (written $\uparrow \Gamma; e$) if it is not defined. Two descriptions, $\Gamma; e_0$ and $\Gamma; e_1$ are *equivalued* (written $\Gamma; e_0 \sim \Gamma; e_1$) if they are both undefined or have a common reduct (i.e. they both reduce to a particular description)

$$\begin{aligned} \downarrow(\Gamma; e) & \Leftrightarrow (\exists \Gamma'; v')(\Gamma; e \mapsto^* \Gamma'; v') \\ \uparrow(\Gamma; e) & \Leftrightarrow \neg \downarrow(\Gamma; e) \\ \Gamma; e_0 \sim \Gamma; e_1 & \Leftrightarrow ((\uparrow(\Gamma; e_0) \wedge \uparrow(\Gamma; e_1)) \vee (\exists \Gamma'; e)(\Gamma; e_0 \mapsto^* \Gamma'; e \wedge \Gamma; e_1 \mapsto^* \Gamma'; e)) \end{aligned}$$

For closed expressions e , we write $\downarrow e$ to mean $\downarrow \emptyset; e$, $e_0 \uparrow e_1$ to mean that $\downarrow e_0$ iff $\downarrow e_1$, and $e_0 \sim e_1$ to mean $\emptyset; e_0 \sim \emptyset; e_1$.

Some simple consequences of the computation rules are that reduction is functional modulo alpha conversion, memory contexts may be pulled out of reduction contexts, and computation is uniform in free variables, unreferenced memory and reduction contexts.

Lemma (cr):

- (i) $\Gamma_0[e_0] = \Gamma_1[e_1]$ if $\Gamma; e \mapsto \Gamma_i; e_i$ for $i < 2$
- (ii) $R[\Gamma[e]] \mapsto^* \Gamma; R[e]$ if $FV(R) \cap \text{Dom}(\Gamma) = \emptyset$.
- (iii) $\Gamma; e \mapsto \Gamma'; e' \Rightarrow (\Gamma; e)^\sigma \mapsto (\Gamma'; e')^\sigma$
if $\text{Dom}(\Gamma') \cap \text{Dom}(\sigma) = \emptyset = FV(\text{Rng}(\sigma)) \cap (\text{Dom}(\Gamma') - \text{Dom}(\Gamma))$.
- (iv) $\Gamma; e \mapsto \Gamma'; e' \Rightarrow (\Gamma_0 \cup \Gamma); e \mapsto (\Gamma_0 \cup \Gamma'); e'$ if $\text{Dom}(\Gamma') \cap \text{Dom}(\Gamma_0) = \emptyset$.
- (v) $\Gamma; R[e] \mapsto \Gamma'; R[e'] \Rightarrow \Gamma; R'[e] \mapsto \Gamma'; R'[e']$ if $(\text{Dom}(\Gamma') \cap FV(R')) \subseteq \text{Dom}(\Gamma)$

In **(cr.i)** \cong is the usual notion of alpha equivalence. It makes explicit the fact that arbitrary choice in cell allocation is the same phenomenon as arbitrary choice of names of bound variables.

2.5 Operational Equivalence of Terms

In this section we define the operational equivalence relation and study its general properties. Operational equivalence formalizes the notion of equivalence as black-boxes. Treating programs as black boxes requires only observing what effects and values they produce, and not how they produce them. Our definition extends the extensional equivalence relations defined by [40] and [46] to computation over memory structures. As shown by [3, 4, 9, 11, 25, 28, 32, 26, 38, 42, 45, 50, 51] operational equivalence and approximation can be characterized in various ways.

Definition (\cong): Two expressions are operationally equivalent, written $e_0 \cong e_1$, if for any closing context C , $C[e_0]$ is defined iff $C[e_1]$ is defined.

$$e_0 \cong e_1 \Leftrightarrow (\forall C \in \mathbb{C} \mid FV(C[e_0]) = FV(C[e_1]) = \emptyset)(C[e_0] \uparrow C[e_1])$$

The operational equivalence is not trivial since the inclusion of branching implies that `t` and `nil` are not equivalent. By definition operational equivalence is a congruence relation on expressions:

$$\text{Congruence : } e_0 \cong e_1 \Rightarrow (\forall C \in \mathbb{C})(C[e_0] \cong C[e_1])$$

However it is not necessarily the case that substitution instances of equivalent expressions are equivalent even if the instantiating expressions always returns a value. As a counter-example we have $\text{if}(\text{cell}?(x), \text{eq}(x, x), \text{t}) \cong \text{t}$ but $\text{if}(\text{cell}?(mk(\text{t})), \text{eq}(mk(\text{t}), mk(\text{t})), \text{t}) \cong \text{nil}$. The reason underlying this is that in the case of programs with effects, returning a value is not an appropriate characterization of definedness. In particular returning a value is not the same as being operationally equivalent to a value. This is in contrast to the purely functional case and is due to the presence of *effects*. For example, each of the following expressions always returns a value

$$mk(x) \quad \text{if}(\text{cell}?(x), \text{set}(x, y), x) \quad \text{if}(\text{cell}?(x), \text{get}(x), x)$$

but none is equivalent to a value, i.e. for no value expression v do we have $e \cong v$ for any of the above three expressions. The first has an *allocation effect*. The second has a *write effect*. The third has a *read effect*.

In general it is very difficult to establish the operational equivalence of expressions. Thus it is desirable to have a simpler characterization of \cong , one that limits the class of contexts (or observations) that must be considered. The main context lemma in this case is the following

Theorem (ciu): $e_0 \cong e_1 \Leftrightarrow (\forall \Gamma, \sigma, R)(FV(\Gamma[e_i^\sigma]) = \emptyset \Rightarrow (\Gamma[R[e_0^\sigma]] \uparrow \Gamma[R[e_1^\sigma]]))$

A proof of **(ciu)** appears in [32], and in [24].

3 The Syntax and Semantics of Formulas

3.1 Syntax

The first order fragment of our logic is a minor generalization of classical first order logic. The atomic formulas assert the equivaluedness and operational equivalence of expressions. In addition to the usual first-order formula constructions, we add a **let**-assertion: if Φ is a formula, x a variable, and e an expression then $\mathbf{let}\{x := e\}[\Phi]$ is a formula.

Definition (\mathbb{W}):

$$\mathbb{W} = (\mathbb{E} \sim \mathbb{E}) + (\mathbb{E} \cong \mathbb{E}) + (\mathbb{W} \Rightarrow \mathbb{W}) + (\mathbf{let}\{\mathbb{X} := \mathbb{E}\}[\mathbb{W}]) + (\forall \mathbb{X})(\mathbb{W})$$

For typographical convenience we will let L range over the class of **let** contexts. Thus L denotes a generic member of $\mathbf{let}\{\mathbb{X} := \mathbb{E}\} \bullet$.

3.2 Semantics

The meaning of formulas is given by a Tarskian satisfaction relation $\Gamma \models \Phi[\sigma]$.

Definition ($\Gamma \models \Phi[\sigma]$): Assume $\Gamma, \sigma, \Phi, e_j$ satisfy $\text{FV}(\Phi^\sigma) \cup \text{FV}(e_j^\sigma) \subseteq \text{Dom}(\Gamma)$ for $j < 2$, and $\text{FV}(\Gamma) = \emptyset$. Then we define the satisfaction relation $\Gamma \models \Phi[\sigma]$ by induction on the structure of Φ :

$$\begin{aligned} \Gamma \models (e_0 \sim e_1)[\sigma] & \quad \text{iff } \Gamma; e_0^\sigma \sim \Gamma; e_1^\sigma \\ \Gamma \models (e_0 \cong e_1)[\sigma] & \quad \text{iff } (\forall R \in \mathbb{R} \mid \text{FV}(R) \subseteq \text{Dom}(\Gamma))(\Gamma[R[e_0^\sigma]] \Downarrow \Gamma[R[e_1^\sigma]]) \\ \Gamma \models (\Phi_0 \Rightarrow \Phi_1)[\sigma] & \quad \text{iff } (\Gamma \models \Phi_0[\sigma]) \text{ implies } (\Gamma \models \Phi_1[\sigma]) \\ \Gamma \models \mathbf{let}\{x := e\}[\Phi][\sigma] & \quad \text{iff } (\Gamma; e^\sigma \mapsto^* \Gamma'; v) \text{ implies } \Gamma' \models \Phi[\sigma\{x := v\}] \\ \Gamma \models (\forall x)\Phi[\sigma] & \quad \text{iff } (\forall v \in \mathbb{V} \mid \text{FV}(v) \subseteq \text{Dom}(\Gamma))(\Gamma \models \Phi[\sigma\{x := v\}]) \end{aligned}$$

As is usual in logic we define the subsidiary notions of validity and logical consequence as follows:

$$\begin{aligned} \models \Phi & \quad \text{iff } (\forall \Gamma, \sigma \mid \text{FV}(\Phi^\sigma) \subseteq \text{Dom}(\Gamma))(\Gamma \models \Phi[\sigma]) \\ \Phi_0 \models \Phi_1 & \quad \text{iff } \models \Phi_0 \Rightarrow \Phi_1 \end{aligned}$$

3.3 Examples, Counterexamples and Caveats

Negation is definable, $\neg\Phi$ is just $\Phi \Rightarrow \mathbf{False}$, where **False** is any unsatisfiable assertion, such as $\mathbf{t} \cong \mathbf{nil}$. Similarly conjunction, \wedge , and disjunction, \vee and the biconditional, \Leftrightarrow , are all definable in the usual manner. Note that termination and non-termination are simple abbreviations. Note that the **let**-assertion is a binding operator akin to \forall . A simple example is the axiom which expresses the effects of **mk**:

$$(\forall z)(\forall y)(\mathbf{let}\{x := \mathbf{mk}(z)\}[\neg(x \cong y) \wedge \mathbf{cell}?(x) \cong \mathbf{t} \wedge \mathbf{get}(x) \cong v])$$

Note that there are at least two notions of definedness that we can express. We let $\downarrow e$ abbreviate $\neg(\mathbf{let}\{x := e\}[\mathbf{False}])$ and $\uparrow e$ abbreviate its negation $\mathbf{let}\{x := e\}[\mathbf{False}]$. A stronger notion of definedness is that of being equivalent (either via \sim or via \cong) to a value. These two notions will be important later.

We use the symbol \simeq to denote either of the binary relations in our logic, \cong and \sim . It is important to note that, unlike equality in first order logic, neither of these binary relations (\cong nor \sim) is a congruence in the sense that $e_0 \simeq e_1 \Rightarrow C[e_0] \simeq C[e_1]$ is falsifiable (even when no trapping occurs). For example

$$\mathbf{get}(x) \simeq \mathbf{t} \Rightarrow \mathbf{app}(\mathbf{seq}(\mathbf{set}(x, \mathbf{nil}), \lambda z.z), \mathbf{get}(x)) \simeq \mathbf{app}(\mathbf{seq}(\mathbf{set}(x, \mathbf{nil}), \lambda z.z), \mathbf{t})$$

is obviously not valid. Similarly false is the related principle that $\Phi \Rightarrow \mathbf{let}\{x := e\}[\Phi]$. For example

$$\mathbf{get}(z) \simeq \mathbf{t} \Rightarrow \mathbf{let}\{x := \mathbf{set}(z, \mathbf{nil})\}[\mathbf{get}(z) \simeq \mathbf{t}]$$

is clearly not valid. Also along these lines is the observation that while

$$\mathbf{let}\{x := e\}[\![e_0 \simeq e_1]\!] \Rightarrow \mathbf{let}\{x := e\}[\![e_0]\!] \simeq \mathbf{let}\{x := e\}[\![e_1]\!]$$

is valid, its converse is false. Since

$$\mathbf{let}\{x := \mathbf{mk}(0)\}\mathbf{let}\{y := \mathbf{mk}(0)\}[x] \simeq \mathbf{let}\{x := \mathbf{mk}(0)\}\mathbf{let}\{y := \mathbf{mk}(0)\}[y]$$

is valid, but $\mathbf{let}\{x := \mathbf{mk}(0)\}[\![\mathbf{let}\{y := \mathbf{mk}(0)\}[x \simeq y]\!]]$ is not.

3.3.1 Violation of Privacy

Rather than give the impression that everything is rosy, we point out the following problem raised in [37]. One seemingly desirable logical principle for contextual reasoning is to be able to replace the e by any operationally equivalent expression without changing the semantics of the contextual assertion $\mathbf{let}\{x := e\}[\![\Phi]\!]$. In other words the following principle seems desirable:

$$e_0 \cong e_1 \Rightarrow (\mathbf{let}\{x := e_0\}[\![\Phi]\!] \Leftrightarrow \mathbf{let}\{x := e_1\}[\![\Phi]\!])$$

However, there are several ways in which this can fail in this logic. For example e_0 may produce some garbage that e_1 does not, and this garbage may be detectable via Φ . For example letting

$$e_0 = \mathbf{seq}(\mathbf{mk}(0), \mathbf{mk}(0))$$

$$e_1 = \mathbf{mk}(0)$$

$$\Phi = (\exists y_0)(\exists y_1)(\mathbf{cell}?(y_0) \cong \mathbf{t} \wedge \mathbf{cell}?(y_1) \cong \mathbf{t} \wedge \mathbf{eq}(y_0, y_1) \cong \mathbf{nil})$$

provides a counterexample.

Another more troublesome counterexample relies on the fact that e_0 and e_1 may be equivalent due to the privacy of certain cells, however their privacy is not respected by the contextual assertion. A simple example of this is:

$$e_0 = \lambda x_0. x_0$$

$$e_1 = \mathbf{let}\{z := \mathbf{mk}(\lambda x_0. x_0)\}\lambda w. \mathbf{app}(\mathbf{get}(z), w)$$

$$\Phi = x \cong \lambda y. y$$

A simple induction on the length of computations (similar to those found in [32]) establishes that e_0 and e_1 are operationally equivalent, and hence $e_0 \cong e_1$ is valid. The essential observation is that the cell z is local to the value/object returned by e_1 and thus invisible and its contents unalterable outside this scope. However it is not the case that

$$\models \mathbf{let}\{x := e_0\}[\![\Phi]\!] \Leftrightarrow \mathbf{let}\{x := e_1\}[\![\Phi]\!]$$

However all is not lost, we do have that the weaker principle

$$e_0 \sim e_1 \Rightarrow (\mathbf{let}\{x := e_0\}[\![\Phi]\!] \Leftrightarrow \mathbf{let}\{x := e_1\}[\![\Phi]\!])$$

is valid.

3.4 Extending the Syntax of Contextual Assertions

For simplicity we have minimized the syntax of *contextual assertions* to simple \mathbf{let} statements. In earlier treatments [24] we dealt with a much wider class of contexts, called *univalent contexts*, (U-contexts). They are the largest natural class of contexts whose symbolic evaluation is unproblematic. The key restriction is that we forbid the hole to appear in the scope of a (non- \mathbf{let}) lambda, thus preventing the proliferation of holes. The class of U-contexts, \mathbb{U} , is defined as follows.

Definition (U):

$$\mathbb{U} = \{ \bullet \} + \mathbf{let}\{\mathbb{X} := \mathbb{E}\}\mathbb{U} + \mathbf{if}(\mathbb{E}, \mathbb{U}, \mathbb{U}) + \mathbb{F}_{m+n+1}(\mathbb{E}^m, \mathbb{U}, \mathbb{E}^n)$$

The semantics is a simple generalization of the one presented here, and the curious are referred to [24] for details. The main reason for restricting our attention to **let** contexts, apart from the simplicity in presentation, is that those left out may be considered abbreviations:

(0) $R[\Phi]$ abbreviates Φ .

(1) $\mathbf{seq}(e_1, e_2, \dots, e_n, [\Phi])$ abbreviates

$$\mathbf{let}\{x_1 := e_1\}[\dots[\mathbf{let}\{x_n := e_n\}[\Phi]]\dots]$$

provided x_i are fresh.

(2) $\mathbf{let}\{x_1 := e_1, \dots, x_n := e_n\}[\Phi]$ abbreviates

$$\mathbf{let}\{x_1 := e_1\}[\dots[\mathbf{let}\{x_n := e_n\}[\Phi]]\dots]$$

provided $x_i \notin \text{FV}(e_j)$ for $1 \leq i < j \leq n$.

(3) $\mathbf{if}(e_0, [\Phi_0], [\Phi_1])$ abbreviates

$$\mathbf{let}\{z := e_0\}[(z \cong \mathbf{nil} \Rightarrow \Phi_1) \wedge (\neg(z \cong \mathbf{nil}) \Rightarrow \Phi_0)] \quad z \text{ fresh.}$$

(4) $\vartheta(e_0, \dots, e_n, U[\Phi], e_{n+1}, \dots)$ abbreviates $\mathbf{seq}(e_0, \dots, e_n, U[\Phi])$

That these abbreviations are in fact sound derive from theorem **(ca.iii)** in [24] which states that (in this generalized semantics):

Theorem (ca):

$$\text{(iii)} \quad U_0[U_1[\Phi]] \Leftrightarrow (U_0[U_1])[\Phi]$$

4 Proof Theory

Since contextual assertions are akin to modalities, we give a Hilbert style presentation. In the long run a natural deduction style system in the style of Prawitz [47] may be more desirable.

Definition ($\vdash \Phi$): The consequence relation, \vdash , is the smallest relation on \mathbb{W} that is closed under the rules given below.

The rules are partitioned into several groups. Each group of rules is given a label, for future reference, and members of the group are numbered. For example **(E.i)** refers to the first rule in the group of equivalence and evaluation rules (the second group below). A rule has a (possibly empty) set of premisses and a conclusion. In the case that the set of premisses is non-empty the rule is displayed with a horizontal bar separating the premisses from the conclusion.

Variable Convention: We adopt Barendregt's convention [8] that in any particular mathematical situation the bound and free variables in *expressions* are distinct. However we do (and must) allow free variables of expressions to coincide with bound (trappable) variables in *contexts*.

So for example we assume in: **(E.vi)** that x not free in R ; **(E.vii)** that $x \notin \text{FV}(e)$; **(C.v)** that $x \notin \text{FV}(\Phi)$; **(Q.ii)** that $x \notin \text{FV}(\Phi_0)$; **(mk.ii)** and **(mk.iii)** that $x \notin \text{FV}(e_0)$; and in **(S.i)** that $x \notin \bar{z}$. On the other hand in **(mk.i)** we must explicitly state that the variable x is distinct from the variables y and z . This convention makes the statement of **(Q.i)** somewhat cumbersome.

Most axioms hold true for both equivaluedness, \sim , and operational equivalence, \cong . If this is the case, then rather than write out the principle twice, we use the symbol \simeq to range over these two equivalence relations. One important reason for introducing \sim is that important principles fails for \cong . In particular **(C.iii)** below fails as indicated in [37] and in §3.3.1.

4.1 Basic Equivalence Axioms and Rules

The first, most basic axiom concerning operational equivalence and equivaluedness is that the booleans `t` and `nil` are not equivalent.

Non-Triviality (T).

$$(i) \quad \vdash \neg(t \simeq \text{nil})$$

The second set of rules concerning equivaluedness hold true also of operational equivalence. They are equivalence relations, **(E.i, E.ii, E.iii)**. They satisfy a certain restricted form of substitutivity, **(E.iv)**. And are preseved under simple forms of evaluation, **(E.v, E.vi, E.vii)**, these last three principles are (equivalent to) the let-rules of the lambda-calculus [39].

Equivalence and Evaluation rules (E).

$$(i) \quad \vdash e_0 \simeq e_0$$

$$(ii) \quad \vdash (e_0 \simeq e_1 \wedge e_1 \simeq e_2) \Rightarrow e_0 \simeq e_2$$

$$(iii) \quad \vdash e_0 \simeq e_1 \Rightarrow e_1 \simeq e_0$$

$$(iv) \quad \vdash e_0 \simeq e_1 \Rightarrow \mathbf{let}\{x := e_0\}e \simeq \mathbf{let}\{x := e_1\}e$$

$$(v) \quad \vdash \mathbf{app}(\lambda x.e, v) \simeq e^{\{x:=v\}} \simeq \mathbf{let}\{x := v\}e$$

$$(vi) \quad \vdash R[e] \simeq \mathbf{let}\{x := e\}R[x]$$

$$(vii) \quad \vdash \mathbf{let}\{x := e_0\}\mathbf{let}\{y := e_1\}e \simeq \mathbf{let}\{y := \mathbf{let}\{x := e_0\}e_1\}e$$

The remaining axioms rules concerning operational equivalence (other than that it is an equivalence relation) are: **(\cong .i)**, equivaluedness implies operational equivalence; **(\cong .ii)**, operational equivalence is preserved under the collection of garbage; **(\cong .iii)**, operational equivalence is non-trivial on abstractions; and they agree with one another on atoms and cells, **(\cong .iv)**.

Operational Equivalence rules (\cong).

$$(i) \quad \vdash e_0 \sim e_1 \Rightarrow e_0 \cong e_1$$

$$(ii) \quad \vdash e \cong \Gamma[e] \quad \text{provided } FV(e) \cap \text{Dom}(\Gamma) = \emptyset$$

$$(iii) \quad \frac{\vdash e_0 \cong e_1}{\vdash \lambda x.e_0 \cong \lambda x.e_1} \quad \text{(The } \xi \text{ Rule)}$$

$$(iv) \quad \vdash \tau(x) \sim \mathbf{t} \wedge \tau(y) \sim \mathbf{t} \Rightarrow (x \cong y \Leftrightarrow x \sim y) \quad \tau \in \{\text{atom?}, \text{cell?}\}$$

4.2 Contextual Axioms and Rules

Contextual assertions are a modality and as such possess a rule akin to necessitation, **(C.i)**. Note that this is a rule of proof and not an implication. A simple counterexample to the implication can be found in §3.3. The remaining axioms concerning contextual assertions are: **(C.ii)**, contextual assertions distribute across the equivalences, again a counterexample to the converse can be found in §3.3; **(C.iii)**, a form of contextual assertion introduction involving equivaluedness (the corresponding principle for operational equivalence is false, §3.3.1); **(C.iv)**, a principle akin to β conversion; and **(C.v)**, a principle allowing for the manipulation of contexts.

Contextual rules (C).

$$(i) \quad \frac{\vdash \Phi}{\vdash L[\Phi]} \quad \text{(Context Introduction)}$$

- (ii) $\vdash L[e_0 \simeq e_1] \Rightarrow L[e_0] \simeq L[e_1]$
- (iii) $\vdash e_0 \sim e_1 \Rightarrow (\mathbf{let}\{x := e_0\}[\Phi] \Leftrightarrow \mathbf{let}\{x := e_1\}[\Phi])$
- (iv) $\vdash \mathbf{let}\{x := v\}[\Phi] \Leftrightarrow \Phi^{x:=v}$
- (v) $\vdash \mathbf{let}\{x := e_0\}\mathbf{let}\{y := e_1\}[\Phi] \Leftrightarrow \mathbf{let}\{y := \mathbf{let}\{x := e_0\}e_1\}[\Phi]$

4.3 Logical Axioms and Rules

The propositional rules are, in addition to the usual Hilbert style presentation of modus ponens, **(P.iii)**, and a generating set of tautologies, **(P.i)** a modal axiom corresponding to **K** and its converse, **(P.ii)**.

Propositional rules (P).

- (i) $\vdash \Phi$ provided Φ is an instance of a tautology
- (ii) $\vdash L[\Phi_0 \Rightarrow \Phi_1] \Leftrightarrow (L[\Phi_0] \Rightarrow L[\Phi_1])$
- (iii)
$$\frac{\vdash \Phi_0 \quad \vdash \Phi_0 \Rightarrow \Phi_1}{\vdash \Phi_1} \quad \text{(Modus Ponens)}$$

Similarly the quantifier axioms are all standard [10] except for **(Q.iv)** and **(Q.v)** which assert that operations other than **mk** and **app** have no allocation effect, and that **mk** only allocates the value it returns.

Quantifier rules (Q).

- (i)
$$\frac{\vdash \Phi}{\vdash (\forall x)\Phi^{x:=x}} \quad \text{(Generalization } \forall I)$$
- (ii) $\vdash (\forall x)(\Phi_0 \Rightarrow \Phi_1) \Rightarrow (\Phi_0 \Rightarrow (\forall x)\Phi_1)$
- (iii) $\vdash (\forall x)\Phi \Rightarrow \Phi^{x:=v}$
- (iv) $\vdash (\forall x)\mathbf{let}\{z := \vartheta(\bar{y})\}[\Phi] \Rightarrow \mathbf{let}\{z := \vartheta(\bar{y})\}[(\forall x)\Phi] \quad \vartheta \in \mathbb{F} - \{\mathbf{mk}, \mathbf{app}\}$
- (v) $\vdash ((\forall x)\mathbf{let}\{z := \mathbf{mk}(y)\}[\Phi(x)] \wedge \mathbf{let}\{z := \mathbf{mk}(y)\}[\Phi(z)]) \Rightarrow \mathbf{let}\{z := \vartheta(\bar{y})\}[(\forall x)\Phi]$

Note that the converses of these last two axioms are easily derivable.

4.4 Undefinedness Principles

The most basic principle concerning undefinedness is that two undefined terms are both equivalued and operationally indistinguishable, **(U.i)**. The rest of the principles concern the partiality of the underlying operation. Note that in the case of the memory operations **mk** and **set**, being defined is not the same as being equivalent to a value. In the other cases this is true, although we need only express the weaker form. The stronger forms are derivable.

Undefinedness rules (U).

- (i) $\vdash \uparrow e_0 \Rightarrow (\uparrow e_1 \Leftrightarrow e_0 \simeq e_1)$
- (ii) $\vdash \downarrow \mathbf{mk}(z)$
- (iii) $\vdash \downarrow \mathbf{set}(z, x) \Leftrightarrow \mathbf{cell}?(z) \simeq \mathbf{t}$
- (iv) $\vdash \downarrow \mathbf{get}(x) \Leftrightarrow \mathbf{cell}?(x) \simeq \mathbf{t} \Leftrightarrow (\exists y)(\mathbf{get}(x) \simeq y)$
- (v) $\vdash \downarrow \vartheta(\bar{x}) \quad \text{for } \vartheta \in \mathbb{T} \cup \{\mathbf{eq}, \mathbf{br}\}$

4.5 Data Operation Axioms and Rules

We treat each operation in turn. We should point out, however, that we have grouped together a collection of principles that concern when an assertion propagates into or out of a context. They may be found after this collection, (S).

The principles concerning `mk`, other than the definedness principle above, (U.ii), are quite straightforward. (mk.i) describes the allocation effect of a call to `mk`. While (mk.ii) and (mk.iii) assert that the time of allocation has no discernable effect on the resulting call. In a world with control effects e_0 must be free of them for this principle to be valid [17].

mk rules (mk).

- (i) $\vdash \text{let}\{x := \text{mk}(z)\} \llbracket \neg(x \simeq y) \wedge \text{cell}?(x) \simeq \mathbf{t} \wedge \text{get}(x) \simeq z \rrbracket \quad x \text{ fresh}$
- (ii) $\vdash \text{let}\{y := e_0\} \text{let}\{x := \text{mk}(z)\} e_1 \simeq \text{let}\{x := \text{mk}(z)\} \text{let}\{y := e_0\} e_1$
- (iii) $\vdash \text{let}\{y := e_0\} \text{let}\{x := \text{mk}(z)\} \Phi \Leftrightarrow \text{let}\{x := \text{mk}(z)\} \text{let}\{y := e_0\} \Phi$

The first two contextual assertions regarding `set` are analogous to those of (mk.i). They describe what is returned and what is altered, what is not altered. The remaining four principles involve the commuting, cancellation, absorption, and idempotence of calls to `set`. For example the `set` absorption principle, (set.v), expresses that under certain simple conditions allocation followed by assignment may be replaced by a suitably altered allocation.

set rules (set).

- (i) $\vdash \text{let}\{x := \text{set}(z, y)\} \llbracket \text{get}(z) \simeq y \wedge x \simeq \text{nil} \rrbracket$
- (ii) $\vdash (y \simeq \text{get}(z) \wedge \neg(w \simeq z)) \Rightarrow \text{let}\{x := \text{set}(w, w_0)\} \llbracket y \simeq \text{get}(z) \rrbracket$
- (iii) $\vdash \neg(x_0 \simeq x_2) \Rightarrow \text{seq}(\text{set}(x_0, x_1), \text{set}(x_2, x_3)) \simeq \text{seq}(\text{set}(x_2, x_3), \text{set}(x_0, x_1))$
- (iv) $\vdash \text{seq}(\text{set}(x, y_0), \text{set}(x, y_1)) \simeq \text{set}(x, y_1)$
- (v) $\vdash \text{let}\{z := \text{mk}(x)\} \text{seq}(\text{set}(z, w), e) \simeq \text{let}\{z := \text{mk}(w)\} e$
- (vi) $\vdash \text{get}(x) \simeq y \Rightarrow \text{set}(x, y) \simeq \text{nil}$

The rules concerning `eq` are unproblematic. It is either true or false, (eq.i). Note that this dichotomy will imply that a call to `eq` is always equivalent to a value. It is true only when its arguments are both atoms and are equivalent, (eq.ii).

eq rules (eq).

- (i) $\vdash \text{eq}(x, y) \simeq \mathbf{t} \vee \text{eq}(x, y) \simeq \text{nil}$
- (ii) $\vdash \text{eq}(x, y) \simeq \mathbf{t} \Leftrightarrow (x \simeq y \wedge \text{atom}?(x) \simeq \mathbf{t} \wedge \text{atom}?(y) \simeq \mathbf{t})$

The recognisers are similarly simple. They are either true or false, (τ .i), and hence always equivalent to values. They are the characteristic functions of disjoint, and exhaustive sets, (τ .ii). And they correspond to the appropriate sets in question, (τ .iii).

τ rules, ($\tau \in \mathbb{T}$) (τ).

- (i) $\vdash \tau(x) \simeq \mathbf{t} \vee \tau(x) \simeq \text{nil}$
- (ii) $\vdash \tau(x) \simeq \mathbf{t} \Leftrightarrow \bigwedge_{\tau' \in \mathbb{T} - \{\tau\}} \tau'(x) \simeq \text{nil}$
- (iii) $\vdash \text{atom}?(v) \simeq \mathbf{t} \quad \text{provided } v \in \mathbb{A}$
- (iv) $\vdash \text{lambda}?(v) \simeq \mathbf{t} \quad \text{provided } v \in \mathbb{L}$

The branching primitive is as simple as `eq`. If its first argument is false then it returns its third argument, (**br.i**). If its first argument is not false then it returns its second argument, (**br.ii**). These together imply that a call to `br` is always equivalent to a value.

br rules (br).

- (i) $\vdash x \simeq \text{nil} \Rightarrow \text{br}(x, y, z) \simeq z$
- (ii) $\vdash \neg(x \simeq \text{nil}) \Rightarrow \text{br}(x, y, z) \simeq y$

4.6 Constraint Propagation Principles

An important class of axioms are those which allow assertions to propagated into and out of assertions.

Static \simeq rules (S).

- (i) $\vdash \text{let}\{x := \vartheta(\bar{z})\} \llbracket x \simeq \vartheta(\bar{z}) \rrbracket$ for $\vartheta \in \mathbb{F} - \{\text{mk}, \text{app}\}$
- (ii) $\vdash \pm(z_0 \simeq z_1) \Rightarrow \text{let}\{x := \vartheta(\bar{y})\} \llbracket \pm(z_0 \simeq z_1) \rrbracket$ for $\vartheta \in \mathbb{F}$
- (iii) $\vdash \downarrow \vartheta(\bar{y}) \Rightarrow (\text{let}\{x := \vartheta(\bar{y})\} \llbracket \pm(z_0 \simeq z_1) \rrbracket \Rightarrow \pm(z_0 \simeq z_1))$ for $\vartheta \in \mathbb{F}$
- (iv) $\vdash \pm(z \simeq \vartheta_0(\bar{y})) \Rightarrow \text{let}\{x := \vartheta_1(\bar{w})\} \llbracket \pm(z \simeq \vartheta_0(\bar{y})) \rrbracket$ $\vartheta_0, \vartheta_1 \in \mathbb{F}$
provided that if $\vartheta_1 \in \{\text{set}, \text{app}\}$, then $\vartheta_0 \in \mathbb{F} - \{\text{get}, \text{set}, \text{app}\}$.
- (v) $\vdash \downarrow \vartheta_1(\bar{w}) \Rightarrow (\text{let}\{x := \vartheta_1(\bar{w})\} \llbracket \pm(z \simeq \vartheta_0(\bar{y})) \rrbracket \Rightarrow \pm(z \simeq \vartheta_0(\bar{y})))$ $\vartheta_0, \vartheta_1 \in \mathbb{F}$
provided that if $\vartheta_1 \in \{\text{set}, \text{app}\}$, then $\vartheta_0 \in \mathbb{F} - \{\text{get}, \text{set}, \text{app}\}$.

As an aside we point out that (**S.i**) is provable when $\vartheta = \text{set}$. In (**S.iv**) and (**S.v**) if $\vartheta_1 \in \{\text{set}, \text{app}\}$ and $\vartheta_0 \in \{\text{get}, \text{set}, \text{app}\}$, then the principles have simple counterexamples.

4.7 Notes and Observations

1. The only axioms and rules concerning `get` are those in (**S**), (**U**), (**mk**) and (**set**).
2. Some axioms above are new in the sense that they have replaced principles that appeared in the earlier treatments [34, 23, 37, 24]. These were pointed out to me by Jacob Frost [20].

- (C.r.i) $\vdash \text{let}\{x := R[e]\}\Phi \Leftrightarrow \text{let}\{z := e\}\text{let}\{x := R[z]\}\Phi$
- (C.r.ii) $\vdash R[\text{let}\{x := e_0\}e_1] \simeq \text{let}\{x := e_0\}R[e_1]$ x not free in R

Proof (C.r.i):

- (1) $\vdash R[e] \sim \text{let}\{z := e\}R[z]$ by (**E.vi**)
- (2) $\vdash \text{let}\{x := R[e]\}\Phi \Leftrightarrow \text{let}\{x := \text{let}\{z := e\}R[z]\}\Phi$ by (**1, C.iii**) and (**P**)
- (3) $\vdash \text{let}\{x := R[e]\}\Phi \Leftrightarrow \text{let}\{z := e\}\text{let}\{x := R[z]\}\Phi$ by (**3, C.v**) and (**P**).

□_{C.r.i}

Proof (C.r.ii):

$$(1) \quad \vdash R[\mathbf{let}\{x := e_0\}e_1] \simeq \mathbf{let}\{y := \mathbf{let}\{x := e_0\}e_1\}R[y]$$

by (E.vi).

$$(2) \quad \vdash R[\mathbf{let}\{x := e_0\}e_1] \simeq \mathbf{let}\{x := e_0\}\mathbf{let}\{y := e_1\}R[y]$$

from (1) using (E.vii, E.ii).

$$(3) \quad \vdash R[\mathbf{let}\{x := e_0\}e_1] \simeq \mathbf{let}\{x := e_0\}R[e_1]$$

from (2) using (E.vi, C.i, C.ii) and (E.ii). $\square_{C.r.ii}$

3. Similarly an previous quantifier principle

$$(Q.p) \quad \vdash L[(\forall x)\Phi] \Rightarrow (\forall x)L[\Phi] \quad \text{where } x \notin FV(L) \cup \text{Traps}(L)$$

is now derivable, again pointed out to me by Jacob Frost [20].

Proof (Q.p):

- | | |
|--|--|
| (1) $\vdash (\forall x)\Phi \Rightarrow \Phi$ | by (Q.iii) |
| (2) $\vdash L[(\forall x)\Phi \Rightarrow \Phi]$ | from (1) using (C.i) |
| (3) $\vdash L[(\forall x)\Phi] \Rightarrow L[\Phi]$ | from (2) using (P.ii) and Modus Ponens |
| (4) $\vdash (\forall x)(L[(\forall x)\Phi] \Rightarrow L[\Phi])$ | from (3) using (Q.i) |
| (5) $\vdash L[(\forall x)\Phi] \Rightarrow (\forall y)(L[\Phi])$ | from (4) using (Q.ii) and the assumption that $x \notin FV(L)$ |

$\square_{Q.p}$

4.8 Derived Rules

Because $L[\]$ is a modality akin to \square we do not have a deduction theorem. However one can easily establish a weak form of the deduction theorem which is useful.

Theorem (Weak Deduction): ¹ Assume that from $\vdash \Phi_0$ one can establish $\vdash \Phi_1$, without using context introduction, (C.i); the ξ rule, (\cong .iii); or generalization on any variable free in Φ_0 . Then $\vdash \Phi_0 \Rightarrow \Phi_1$.

Proof (Weak Deduction): This is a very simple induction on the length of proof. \square

A simple corollary of this is a version of reduction ad absurdum:

$$(\mathbf{Reductio\ Ad\ Absurdum}) \quad \frac{\frac{(\vdash \Phi)}{\vdash \mathbf{False}}}{\vdash \neg \Phi}$$

is derivable if the derivation

$$\frac{(\vdash \Phi)}{\vdash \mathbf{False}}$$

does not use context introduction, (C.i); the ξ rule, (\cong .iii); or generalization on any variable free in Φ . Since if this is the case then in fact using (**weak deduction**) we have

$$\vdash \Phi \Rightarrow \mathbf{False}.$$

¹We could strengthen the theorem by weakening the condition *without using* to *without using on any formula depending on the assumption* $\vdash \Phi_0$

And consequently by definition $\vdash \neg\Phi$. A similar observation reduces the strong form of $(\exists\mathbf{I})$

$$(\exists\mathbf{E}) \quad \frac{\frac{(\vdash \Phi_0)}{\vdash \Phi_1} \quad \vdash (\exists y)\Phi_0 \quad y \notin \text{FV}(\Phi_1)}{\vdash \Phi_1}$$

to the derivable form:

$$(\exists\mathbf{E}) \quad \frac{\vdash \Phi_0 \Rightarrow \Phi_1}{\vdash (\exists y)\Phi_0 \Rightarrow \Phi_1} \quad y \notin \text{FV}(\Phi_1)$$

4.9 Simple Counterexamples

The following variations on the **(S)** principles are *not* valid.

- (1) $\mathbf{let}\{x := \vartheta(\bar{z})\}[\![x \simeq \vartheta(\bar{z})]\!] \quad \vartheta \in \{\mathbf{mk}, \mathbf{app}\}$
- (2) $\pm(z \simeq \vartheta_0(\bar{y})) \Rightarrow \mathbf{let}\{x := \vartheta_1(\bar{w})\}[\![\pm(z \simeq \vartheta_0(\bar{y}))]\!]$
 $\vartheta_1 \in \{\mathbf{app}, \mathbf{set}\}$ and $\vartheta_0 \in \{\mathbf{set}, \mathbf{get}, \mathbf{app}\}$
- (3) $\downarrow \vartheta_1(\bar{w}) \Rightarrow (\mathbf{let}\{x := \vartheta_1(\bar{w})\}[\![\pm(z \simeq \vartheta_0(\bar{y}))]\!] \Rightarrow \pm(z \simeq \vartheta_0(\bar{y})))$
 $\vartheta_1 \in \{\mathbf{app}, \mathbf{set}\}$ and $\vartheta_0 \in \{\mathbf{set}, \mathbf{get}, \mathbf{app}\}$

5 Completeness

We say that an expression is *first order*, $e \in \mathbb{E}^\circ$, iff it contains neither unapplied λ -expressions, nor non- λ applications. A formula is *first order*, $\Phi \in \mathbb{W}^\circ$, iff it is built up from first order expressions. The appropriate first order syntactic subclasses are defined formally by the following mutually recursive definitions:

Definition ($\mathbb{V}^\circ \mathbb{L}^\circ \mathbb{E}^\circ \mathbb{W}^\circ$):

$$\begin{aligned} \mathbb{V}^\circ &= \mathbb{X} + \mathbb{A} + \mathbb{P}^\circ \\ \mathbb{L}^\circ &= \lambda\mathbb{X}.\mathbb{E}^\circ \\ \mathbb{E}^\circ &= \mathbb{V}^\circ + \mathbf{app}(\mathbb{L}^\circ, \mathbb{E}^\circ) + (\mathbb{F}_n - \{\mathbf{app}\})(\mathbb{E}^{\circ n}) \\ \mathbb{W}^\circ &= (\mathbb{E}^\circ \simeq \mathbb{E}^\circ) + (\mathbb{W}^\circ \Rightarrow \mathbb{W}^\circ) + (\mathbf{let}\{\mathbb{X} := \mathbb{E}^\circ\}[\![\mathbb{W}^\circ]\!]) + (\forall\mathbb{X})(\mathbb{W}^\circ) \end{aligned}$$

Definition ($\mathbb{II} \tilde{\mathbb{II}}$): The set of *constraints*, \mathbb{II} , and the set of *complex constraints*, $\tilde{\mathbb{II}}$, are defined as follows:

$$\begin{aligned} \mathbb{II} &:= \pm(\mathbb{V}^\circ \sim \mathbb{V}^\circ) + (\mathbb{F}_1 - \{\mathbf{mk}\})(\mathbb{V}^\circ \sim \mathbb{V}^\circ) \\ \tilde{\mathbb{II}} &:= \pm(\mathbb{V}^\circ \simeq \mathbb{V}^\circ) + (\mathbb{F}_1 - \{\mathbf{mk}\})(\mathbb{V}^\circ \simeq \mathbb{V}^\circ) + (\tilde{\mathbb{II}} \Rightarrow \tilde{\mathbb{II}}) \end{aligned}$$

A *simple constraint set*, π , is defined to be a finite subset of \mathbb{II} , $\pi \in \mathbf{P}_\omega(\mathbb{II})$. A *complex constraint*, $\tilde{\pi}$, is an element of $\tilde{\mathbb{II}}$. We let π range over simple constraints sets, and $\tilde{\pi}$ range over complex constraints. A simple constraint is said to be *static* if it is a subset of $\mathbb{II} - \{\mathbf{get}\}(\mathbb{V}^\circ)$. A complex constraint is said to be static if it is a boolean combination of elements from $\mathbb{II} - \{\mathbf{get}\}(\mathbb{V}^\circ)$. Note that by **(S)**, static constraints propagate through any contextual assertion.

It would perhaps be more symmetric if we defined simple constraints to be conjunctions of constraints. The reason we define them to be sets constraints is to facilitate a single definition, in particular π_M below. Modulo this one definition,

the reader may reasonably assume that π is a finite conjunction of elements from Π . Thus any simple constraint set is equivalent to a single complex constraint. Note that a constraint set is a finite collection of formulas of the form $v_0 \sim v_1$ or $\neg(v_0 \sim v_1)$ or

$$\{\mathbf{get}, \mathbf{atom}?, \mathbf{cell}?, \mathbf{lambda}?\}(v_0) \sim v_1.$$

Negations of the latter are not needed since $\neg(\vartheta(v_0) \sim v_1)$ can be rewritten as $\{\vartheta(v_0) \sim z, \neg(z \sim v_1)\}$ for z fresh. We sometimes abuse notation and identify π with the conjunction of its members, hence treating a simple constraint set as a special type of complex constraint.

Definition (Cells(π)): Cells(π) is the subset of $\text{FV}(\pi)$ defined by

$$\text{Cells}(\pi) = \{x \in \text{FV}(\pi) \mid \pi \models \mathbf{cell}?(x) \sim \mathbf{t}\}.$$

If $x \in \text{Cells}(\pi)$, then x must be interpreted as a cell. To express the constraints implicit in a memory context Γ we define for any π the extension of π by Γ relative to a given set of variables X to be π_Γ^X :

Definition (π_Γ^X): If $X \in \mathbf{P}_\omega(\mathbb{X} - \text{Dom}(\Gamma))$ and $\text{FV}(\pi) \cap \text{Dom}(\Gamma) = \emptyset$, then we define π_Γ^X as follows

$$\begin{aligned} \pi_\Gamma^X &= \pi \cup \pi_{\text{cells}} \cup \pi_{\text{contents}} \cup \pi_{\text{distinct}} \\ \pi_{\text{cells}} &= \{\mathbf{cell}?(z) \sim \mathbf{t} \mid z \in \text{Dom}(\Gamma)\} \\ \pi_{\text{contents}} &= \{\mathbf{get}(z) \sim \Gamma(z) \mid z \in \text{Dom}(\Gamma)\} \\ \pi_{\text{distinct}} &= \{\neg(z \sim y) \mid y \in \text{FV}(\pi) \cup X \cup (\text{Dom}(\Gamma) - \{z\}), z \in \text{Dom}(\Gamma)\}. \end{aligned}$$

5.1 The First Completeness Theorem

We begin by proving a quantifier free version of the main theorem. The full version is then a simple generalization.

Theorem (Completeness – I): If $\Phi \in \mathbb{W}^0$ is first order and is quantifier free, then there is a complex constraint $\tilde{\pi}$ such that

$$\vdash \tilde{\pi} \Leftrightarrow \Phi$$

Note that by using propositional calculus together with **(P.ii)** (i.e. the principle

$$\vdash L[\Phi_0 \Rightarrow \Phi_1] \Leftrightarrow (L[\Phi_0] \Rightarrow L[\Phi_1])$$

it suffices to demonstrate the theorem when Φ is of the form

$$\underbrace{L_1[L_2[\dots L_n[e_0 \simeq e_1] \dots]]}_{n \geq 0}$$

For this reason we define

$$\mathbf{L}^* = \{\bullet\} + \mathbf{let}\{\mathbb{X} := \mathbb{E}^0\}\mathbf{L}^*$$

and let L^* range over \mathbf{L}^* .

The proof of the completeness theorem involves the symbolic evaluation of arbitrary formulas and expressions, with respect to a suitable set of constraints, to a canonical form. The symbolic evaluation of an expression, with respect to a set of constraints π , requires keeping track of three things: the newly allocated memory; the modifications to the original memory (described by π); and the remaining computation. The remainder of a computation is simply an expression. The newly allocated memory is simply a memory context. The modifications to the original memory are represented

by another special kind of context called a modification, M . We begin by defining relative to a fixed constraint set π a symbolic reduction relation $\xrightarrow{*}_{\pi}$. It is defined in such a way that:

Contexts: $(L^*_0[\Phi_0] \xrightarrow{*}_{\pi} L^*_1[\Phi_1])$ implies $\vdash \pi \Rightarrow (L^*_0[\Phi_0] \Leftrightarrow L^*_1[\Phi_1])$
and

Expressions: $(e_0 \xrightarrow{*}_{\pi} e_1)$ implies $\vdash \pi \Rightarrow e_0 \sim e_1$.

The definition requires the notion of a modification and the corresponding decomposition of contexts and expressions. The effects that the evaluation of an expression has on the original memory, described by constraints, are represented by contexts called modifications. They are simply sequences of assignments to variables that are not in the domain of the memory context, but are assumed to be cells.

Definition (Modifications): A *modification*, M , is a context of the form

$$\text{seq}(\text{set}(z_1, v_1), \dots, \text{set}(z_n, v_n), \bullet)$$

where $z_i = z_j$ implies $i = j$. We define $\text{Dom}(M) = \{z_1, \dots, z_n\}$ and (in analogy with $\Gamma(x)$) $M(z_i) = v_i$ for $i = 1, \dots, n$.

5.2 π -Reduction

In analogy to the semantic reduction relations we define \mapsto_{π} , and $\xrightarrow{*}_{\pi}$. In order to ensure that definitions are meaningful we introduce the notion of coherence. Roughly a constraint and a pair of memory and modification contexts are coherent (written $\text{Coh}(\pi, \Gamma; M)$) if $\text{Dom}(\Gamma) \cap \text{FV}(\pi) = \emptyset$, modifications in M are to elements of $\text{Cells}(\pi)$, π decides equality on $\text{Cells}(\pi)$, distinct elements of $\text{Dom}(M)$ are provably distinct in π , and π contains at most one **get** assertion for any z in $\text{Cells}(\pi)$. (The last condition is a technicality to make various definitions and proofs simpler.)

Definition (Coherence): If Γ is a memory context and M is a modification as above then we say $(\pi, \Gamma; M)$ is coherent, written $\text{Coh}(\pi, \Gamma; M)$, if the following five conditions hold:

- (1) $\text{Dom}(\Gamma) \cap \text{FV}(\pi) = \emptyset$
- (2) $\text{Dom}(M) \subseteq \text{Cells}(\pi)$
- (3) If $x_0, x_1 \in \text{Dom}(M)$ are distinct, then $\pi \models \neg(x_0 \sim x_1)$.
- (4) If $x_0, x_1 \in \text{Cells}(\pi)$, then $\pi \models (x_0 \sim x_1)$ or $\pi \models \neg(x_0 \sim x_1)$.
- (5) If $x \in \text{Cells}(\pi)$, then there is at most one formula $(\text{get}(z) \sim v) \in \pi$ with $\pi \models (z \sim x)$, and if $(\text{get}(z) \sim v) \in \pi$, then $x \in \text{Cells}(\pi)$.

We write $\text{Coh}(\pi, M)$ for $\text{Coh}(\pi, \Gamma; M)$ when $\text{Dom}(\Gamma)$ is empty, when $\text{Dom}(M)$ is empty we write $\text{Coh}(\pi, \Gamma)$ for $\text{Coh}(\pi, \Gamma; M)$, and when $\text{Dom}(\Gamma)$ and $\text{Dom}(M)$ are both empty we write $\text{Coh}(\pi)$ for $\text{Coh}(\pi, \Gamma; M)$.

One use of the notion of coherence is to ensure the simplicity of the following definition of π_M . If a modification, M , and a constraint set, π , are coherent, then the modification of π implicit in M is made explicit in π_M . To construct π_M from π we first remove the set of all assertions in π concerning contents of cells that are mutated by M . The set removed is referred to as π_{forget} and is well defined by virtue of coherence. Then we add to $\pi - \pi_{\text{forget}}$ the set of assertions, π_{assign} concerning the cells updated by M .

Definition (π_M): For $\text{Coh}(\pi, M)$ we define π_M as follows

$$\begin{aligned} \pi_M &= (\pi - \pi_{\text{forget}}) \cup \pi_{\text{assign}} \\ \pi_{\text{assign}} &= \{\text{get}(z) \sim v \mid M(z) = v, z \in \text{Dom}(M)\} \\ \pi_{\text{forget}} &= \{(\text{get}(x) \sim v) \in \pi \mid (\exists z \in \text{Dom}(M))(\pi \models x \sim z)\} \end{aligned}$$

Definition ($M\{z := \mathbf{mk}(v)\}_\pi$): Suppose that M is a modification, $\text{Coh}(\pi, M)$ and $z \in \text{Cells}(\pi)$. Then $M\{z := \mathbf{mk}(v)\}_\pi$ is defined to be the modification M' with $\text{Dom}(M') = \text{Dom}(M) \cup \{z\}$, and for $z' \in \text{Dom}(M')$

$$M'(z') = \begin{cases} M(z') & \text{if } \pi \models \neg(z \sim z') \\ v & \text{if } \pi \models (z \sim z') \end{cases}$$

Definition ($e_0 \xrightarrow{*}_\pi e_1$): Assume that $\text{Coh}(\pi, \Gamma; M)$ and that $\pi' = (\pi_\Gamma)_M$. Then the reduction relation $\xrightarrow{*}_\pi$ on expressions is the reflexive transitive closure of $\xrightarrow{\pi}$ given by:

$$\begin{aligned} (\beta_v) \quad & \Gamma[M[R[\text{app}(\lambda x.e, v)]]] \xrightarrow{\pi} \Gamma[M[R[e^{x:=v}]]] \\ (\tau) \quad & \Gamma[M[R[\tau(v)]]] \xrightarrow{\pi} \begin{cases} \Gamma[M[R[\mathbf{t}]]] & \text{if } \pi' \models \tau(v) \sim \mathbf{t} \\ \Gamma[M[R[\mathbf{nil}]]] & \text{if } \pi' \models \tau(v) \sim \mathbf{nil} \end{cases} \quad \text{provided } \tau \in \mathbb{T} \\ (\text{eq}) \quad & \Gamma[M[R[\text{eq}(v_0, v_1)]]] \xrightarrow{\pi} \begin{cases} \Gamma[M[R[\mathbf{t}]]] & \text{if } \pi' \models \Psi_{\mathbf{t}}(v_0, v_1), \\ \Gamma[M[R[\mathbf{nil}]]] & \text{if } \pi' \models \neg\Psi_{\mathbf{t}}(v_0, v_1) \end{cases} \\ & \Psi_{\mathbf{t}}(v_0, v_1) := v_0 \sim v_1 \wedge \text{atom}?(v_0) \sim \mathbf{t} \wedge \text{atom}?(v_1) \sim \mathbf{t} \\ (\text{br}) \quad & \Gamma[M[R[\text{br}(v_0, v_1, v_2)]]] \xrightarrow{\pi} \begin{cases} \Gamma[M[R[v_1]]] & \text{if } \pi' \models \neg(v_0 \sim \mathbf{nil}) \\ \Gamma[M[R[v_2]]] & \text{if } \pi' \models v_0 \sim \mathbf{nil} \end{cases} \\ (\text{mk}) \quad & \Gamma[M[R[\mathbf{mk}(v)]]] \xrightarrow{\pi} \Gamma\{z := \mathbf{mk}(v)\}[M[R[z]]] \quad z \text{ fresh} \\ (\text{get}) \quad & \Gamma[M[R[\text{get}(z)]]] \xrightarrow{\pi} \Gamma[M[R[v]]] \quad \text{if } \pi' \models \text{get}(z) \sim v \\ (\text{set}) \quad & \Gamma[M[R[\text{set}(z, v)]]] \xrightarrow{\pi} \begin{cases} \Gamma\{z := \mathbf{mk}(v)\}[M[R[\mathbf{nil}]]] & \text{if } z \in \text{Dom}(\Gamma) \\ \Gamma[M\{z := \mathbf{mk}(v)\}_\pi[R[\mathbf{nil}]]] & \text{if } \pi \models \text{cell}?(z) \sim \mathbf{t} \end{cases} \end{aligned}$$

Definition ($L^*_0[\Phi_0] \xrightarrow{*}_\pi L^*_1[\Phi_1]$): Assume that $\text{Coh}(\pi, \Gamma; M)$. Then the reduction relation $\xrightarrow{*}_\pi$ on formulas is the reflexive transitive closure of $\xrightarrow{\pi}$ given by:

$$\begin{aligned} (\text{val}) \quad & \Gamma[M[\text{let}\{x := v\}\bullet]][\Phi] \xrightarrow{\pi} \Gamma[M][\Phi^{x:=v}] \\ (\text{red}) \quad & \Gamma[M[\text{let}\{x := e\}L^*]][\Phi] \xrightarrow{\pi} \Gamma_1[M_1[\text{let}\{x := e_1\}L^*]][\Phi] \\ & \text{provided } \Gamma[M[e]] \xrightarrow{\pi} \Gamma_1[M_1[e_1]] \end{aligned}$$

Lemma (Coherence): Coherence is preserved by syntactic reduction.

The Context Modification Introduction lemma, **(CMI)**, generalizes the contextual assertions concerning \mathbf{mk} and \mathbf{set} to arbitrary memory–modification contexts pairs.

Lemma (CMI): If $\text{Coh}(\pi, \Gamma; M)$ and $X = \text{FV}(\Gamma[M[R]]) \cup \text{FV}(\pi)$ then

$$\vdash \pi \Rightarrow \Gamma[M[R]][(\pi_\Gamma^X)_M].$$

Proof (CMI): Let

$$\begin{aligned} \pi_{\text{cells}} &= \{\text{cell}?(z) \sim \mathbf{t} \mid z \in \text{Dom}(\Gamma)\} \\ \pi_{\text{contents}} &= \{\text{get}(z) \sim \Gamma(z) \mid z \in \text{Dom}(\Gamma)\} \\ \pi_{\text{distinct}} &= \{\neg(z \sim y) \mid y \in X \cup (\text{Dom}(\Gamma) - \{z\}), z \in \text{Dom}(\Gamma)\}. \end{aligned}$$

Suppose without loss of generality that:

$$\begin{aligned}\Gamma &= \text{let}\{z_1 := \text{mk}(\text{nil})\} \dots \text{let}\{z_n := \text{mk}(\text{nil})\} \\ &\quad \text{seq}(\text{set}(z_1, v_1), \dots, \text{set}(z_n, v_n), \bullet) \\ \mathbf{M} &= \text{seq}(\text{set}(x_1, v'_1), \dots, \text{set}(x_m, v'_m), \bullet)\end{aligned}$$

Then

$$\begin{aligned}\vdash \pi &\Rightarrow \text{let}\{z_1 := \text{mk}(\text{nil})\} \dots \text{let}\{z_n := \text{mk}(\text{nil})\} \llbracket \pi \rrbracket \\ &\quad \text{by (S) and propositional logic (P)} \\ \vdash \pi &\Rightarrow \text{let}\{z_1 := \text{mk}(\text{nil})\} \dots \text{let}\{z_n := \text{mk}(\text{nil})\} \llbracket \pi_{\text{cells}} \cup \pi_{\text{distinct}} \rrbracket \\ &\quad \text{by (S), (mk.i) and propositional logic (P)} \\ \vdash \pi \cup \pi_{\text{cells}} \cup \pi_{\text{distinct}} &\Rightarrow \text{seq}(\text{set}(z_1, v_1), \\ &\quad \vdots \\ &\quad \text{set}(z_n, v_n), \bullet) \llbracket \pi \cup \pi_{\text{cells}} \cup \pi_{\text{distinct}} \cup \pi_{\text{contents}} \rrbracket \\ &\quad \text{by (S), (set.ii), (set.i) and propositional logic (P)}\end{aligned}$$

Thus by the above and **(P.cut)**

$$\vdash \pi \Rightarrow \Gamma \llbracket \pi_{\Gamma}^X \rrbracket$$

Now by coherence we may split π into two disjoint sets π' and π_{forget} so that

- (a.) for any $(v \sim \text{get}(w)) \in \pi'$ we have that $\pi' \models \neg(x \sim w)$ for every $x \in \text{Dom}(\mathbf{M}) = \{x_1, \dots, x_m\}$.
- (b.) π_{forget} contains only those statements of the form $(v \sim \text{get}(w))$ such that there is an $x \in \text{Dom}(\mathbf{M}) = \{x_1, \dots, x_m\}$ such that $\pi' \models x \sim w$. Thus

$$\begin{aligned}\vdash \pi' \cup \pi_{\text{cells}} \cup \pi_{\text{distinct}} &\Rightarrow \text{seq}(\text{set}(x_1, v'_1), \dots, \text{set}(x_m, v'_m), \bullet) \llbracket \pi' \cup \pi_{\text{cells}} \cup \pi_{\text{distinct}} \rrbracket \\ &\quad \text{by (S), (set.ii), (set.i), (a.) and propositional logic (P)} \\ \vdash \pi' &\Rightarrow \text{seq}(\text{set}(x_1, v'_1), \dots, \text{set}(x_m, v'_m), \bullet) \llbracket \{\text{get}(x_i) \sim v_i \mid i = 1, \dots, m\} \rrbracket \\ &\quad \text{by (S), (set.ii), (set.i), and propositional logic (P)}\end{aligned}$$

Thus by the above, **(P.cut)** and **(C.rdx)**

$$\vdash \pi \Rightarrow \Gamma[\mathbf{M}[\mathbf{R}]] \llbracket (\pi_{\Gamma}^X)_{\mathbf{M}} \rrbracket$$

\square_{CMI}

A simple but useful corollary of **(CMI)** is the following:

Corollary (cmi):

$$\text{If } \text{Coh}(\pi, \Gamma; \mathbf{M}), \quad \text{and} \quad \vdash (\pi_{\Gamma})_{\mathbf{M}} \Rightarrow \uparrow e, \quad \text{then} \quad \vdash \pi \Rightarrow \uparrow \Gamma[\mathbf{M}[\mathbf{R}[e]]].$$

Proof (cmi):

- (1) $\vdash \pi \Rightarrow \Gamma[\mathbf{M}[\mathbf{R}]] \llbracket (\pi_{\Gamma}^X)_{\mathbf{M}} \rrbracket$ by **(CMI)**.
- (2) $\vdash (\pi_{\Gamma})_{\mathbf{M}} \Rightarrow \uparrow e$ by assumption.
- (3) $\vdash \pi \Rightarrow \Gamma[\mathbf{M}[\mathbf{R}]] \llbracket \uparrow e \rrbracket$ by the above two facts and **(P.cut)**.
- (4) $\vdash \pi \Rightarrow \Gamma[\mathbf{M}[\mathbf{R}]] \llbracket \text{seq}(e, \text{False}) \rrbracket$ by definition.
- (5) $\vdash \pi \Rightarrow \text{seq}(\Gamma[\mathbf{M}[\mathbf{R}[e]]], \llbracket \text{False} \rrbracket)$ by repeated application of **(C.r)**
- (6) $\vdash \pi \Rightarrow \uparrow \Gamma[\mathbf{M}[\mathbf{R}[e]]]$ by definition.

□_{cmi}

Before we state the key lemmas, we require one last set of definitions. Syntactic reduction is defined so that if π contains enough information concerning the nature of the free variables of e , then

$$e \xrightarrow{*}_{\pi} \Gamma[M[e']],$$

and either $e' = v$ or else e' corresponds to a stuck state, one that cannot reduce due to simple type mismatches.

Definition (π -stuck state): An expression e is said to be π -stuck state if e can be written as $\Gamma[M[R[e']]]$ for some Γ , M , and e' , such that $e' \in \{\mathbf{get}(v), \mathbf{set}(v, v')\}$, and $(\pi_{\Gamma})_M \models \mathbf{cell}?(v) \sim \mathbf{nil}$.

An expression e is said to *reduce* to a π -stuck state if $e \xrightarrow{*}_{\pi} e'$, and e' is a π -stuck state. Similarly a formula $L^*[\Phi]$ is said to reduce to a π -stuck state if

$$L^*[\Phi] \xrightarrow{*}_{\pi} \Gamma[M[\mathbf{let}\{x := R[e']\bullet\}]][\Phi']$$

$\text{Coh}(\pi, \Gamma; M)$, and $\Gamma[M[R[e']]]$ is a π -stuck state.

In order to formalize the notion of a constraint set π containing enough information, we make the following definitions. A accessor chain of length n is a reduction context of the form

$$\vartheta_1(\vartheta_2(\dots \vartheta_n(\bullet) \dots))$$

where $\vartheta_i \in \{\mathbf{get}\}$. Note that an accessor chain of length 0 is just \bullet . Finally we define the notion of n -completeness for constraint sets relative to a finite set of variables and atoms, $[\bar{x}, A]$. The idea is that such a constraint set contains sufficient information to completely determine the evaluation of any expression of size less than n built from the given variables and atoms.

Definition (n -Complete w.r.t. $[\bar{x}, A]$): π is n -complete w.r.t. $[\bar{x}, A]$ if for every Θ, Θ_0 , accessor chains of length $\leq n$, and $y, y_0 \in \bar{x}$, if $\pi \models \Theta[y] \sim v$ and $\pi \models \Theta_0[y_0] \sim v_0$, then

- (1) $\pi \models \tau(v) \sim \mathbf{t}$ or $\pi \models \tau(v) \sim \mathbf{nil}$ $\tau \in \mathbb{T}$
- (2) $\pi \models v \sim \alpha$ or $\pi \models \neg(v \sim \alpha)$ $\alpha \in A \cup \{\mathbf{t}, \mathbf{nil}, v_0\}$
- (3) $\pi \models \mathbf{cell}?(v) \sim \mathbf{t}$ implies $(\exists v_0 \in \mathbb{V})(\pi \models \mathbf{get}(v) \sim v_0)$
- (4) $\pi \models \mathbf{cell}?(v) \sim \mathbf{nil}$ implies $\neg(\exists v_0 \in \mathbb{V})(\pi \models \mathbf{get}(v) \sim v_0)$

Definition ($\text{Atoms}(Z)$): If $Z \subseteq \mathbb{E}$, then $\text{Atoms}(Z)$ is the set of atoms occurring in Z .

5.3 The Main Lemmas

The following five lemmas enable a straightforward proof of the completeness theorem. Lemmas 0., 1., 3., and 4. hold for the full language, while Lemma 2. holds only for those expressions which are first order.

Lemma (0): If $\tilde{\pi}_0$ and $\tilde{\pi}_1$ are complex constraints, then

$$\tilde{\pi}_0 \models \tilde{\pi}_1 \quad \text{iff} \quad \vdash \tilde{\pi}_0 \Rightarrow \tilde{\pi}_1.$$

Lemma (1):

- (i) If $e \xrightarrow{*}_{\pi} e'$, then $\vdash \pi \Rightarrow e \sim e'$.
- (ii) If $L^*[\Phi] \xrightarrow{*}_{\pi} L^*[\Phi']$, then $\vdash \pi \Rightarrow (L^*[\Phi] \Leftrightarrow L^*[\Phi'])$.

Lemma (2): Assume $e, L^*[\Phi]$ are first order, $\text{FV}(e, L^*) \subseteq X$, $\text{Atoms}(\pi, e, L^*) \subseteq A$ and that $m \in \mathbb{N}$.

- (i) If π is $(r(e) + m)$ -complete w.r.t. $[X, A]$ and $\text{Coh}(\pi)$, then either e reduces to a π -stuck state, or else there exists a memory context Γ , a modification M , and a v such that $e \xrightarrow{*}_{\pi} \Gamma[M[v]]$, $\text{Coh}(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is m -complete w.r.t. $[X \cup \text{Dom}(\Gamma), A \cup \text{Atoms}(v)]$.
- (ii) If π is $(r(L^*) + m)$ -complete w.r.t. $[X, A]$ and $\text{Coh}(\pi)$, then either $L^*[\Phi]$ reduces to a π -stuck state, or else there exists a memory context Γ , a modification M and a substitution σ such that $L^*[\Phi] \xrightarrow{*}_{\pi} \Gamma[M][[\Phi^{\sigma}]]$, $\text{Coh}(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is m -complete w.r.t. $[X \cup \text{Dom}(\Gamma), A \cup \text{Atoms}(\text{Rng}(\sigma))]$.

Lemma (3): For any consistent $\pi, \bar{x}, A \in \mathbf{P}_{\omega}(\mathbb{A})$, and $n \in \mathbb{N}$ there exists $N \in \mathbb{N}$ and a family of constraint sets $\{\pi_i\}_{i < N}$ such that

- (i) Each π_i is n -complete w.r.t. $[\bar{x}, \text{Atoms}(\pi_i, A)]$, and $\text{Coh}(\pi_i)$.
- (ii) $\vdash \pi \Leftrightarrow (\bigvee_{i < N} \pi_i)$

Lemma (4): Suppose π and $e_i = \Gamma_i[M_i[v_i]]$, $i < 2$, are such that $\text{Coh}(\pi, \Gamma_i; M_i)$ for $i < 2$, and π is 1-complete with respect to $[\text{FV}(e_0) \cup \text{FV}(e_1), \text{Atoms}(e_0, e_1)]$. Then there are static complex constraints $\tilde{\pi}_i$ and $\tilde{\pi}_{ii}$ such that

- (i) $\vdash \pi \Rightarrow (\tilde{\pi}_i \Leftrightarrow e_0 \sim e_1)$
- (ii) $\vdash \pi \Rightarrow (\tilde{\pi}_{ii} \Leftrightarrow e_0 \cong e_1)$

5.4 Proof of Completeness – I

Proof (Completeness – I): Fix a particular $L^*[[e_0 \simeq e_1]]$ and let $\text{FV}(L^*[[e_0 \simeq e_1]]) = X$ and $\text{Atoms}(L^*[[e_0 \simeq e_1]]) = A$. By propositional logic and lemma (3) we need only show that there is a complex constraint $\tilde{\pi}$ such that

$$\vdash \pi \Rightarrow (\tilde{\pi} \Leftrightarrow L^*[\Phi])$$

assuming that π is m -complete w.r.t. $[X, \text{Atoms}(\pi, A)]$, and $\text{Coh}(\pi)$ for suitably large m ($m \geq 1 + r(L^*) + \max(r(e_0), r(e_1))$). Now by lemma (2.(ii)) either $L^*[[e_0 \simeq e_1]]$ reduces to a π -stuck state, or else there exists a memory context Γ , a modification M and a substitution σ such that $L^*[[e_0 \simeq e_1]] \xrightarrow{*}_{\pi} \Gamma[M][[(e_0 \simeq e_1)^{\sigma}]]$, $\text{Coh}(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is $(m - r(L^*))$ -complete w.r.t. $[X \cup \text{Dom}(\Gamma), A \cup \text{Atoms}(\text{Rng}(\sigma))]$. We consider these two cases in turn:

(1.) Suppose $L^*[[e_0 \simeq e_1]]$ reduces to a π -stuck state:

$$L^*[\Phi] \xrightarrow{*}_{\pi} \Gamma[M[\mathbf{let}\{x := R[e']\bullet\}][[\Phi']]] \quad \text{Coh}(\pi, \Gamma; M) \text{ and } \Gamma[M[R[e']]] \text{ is a } \pi\text{-stuck state.}$$

Hence $e' \in \{\mathbf{get}(v), \mathbf{set}(v, v')\}$, and $(\pi_{\Gamma})_M \models \mathbf{cell}?(v) \sim \mathbf{nil}$. In this situation using lemma (0), the axioms for undefinedness, (U), and corollary (cmi) we have that

$$\vdash \pi \Rightarrow \uparrow \Gamma[M[R[e']]]$$

Thus by this (with $R = \mathbf{let}\{x := R[\bullet_1]\bullet_2\}$) and (P) we obtain

$$\begin{aligned} \vdash \pi &\Rightarrow \Gamma[M[\mathbf{let}\{x := R[e', L^*]\}][[\Phi']]] \\ \vdash \pi &\Rightarrow L^*[[e_0 \simeq e_1]] \quad \text{by lemma (1.(ii))} \end{aligned}$$

Thus any tautological complex constraint $\tilde{\pi}$ will suffice. \square_1 .

(2.) Suppose $L^*[[e_0 \simeq e_1]] \xrightarrow{*}_{\pi} \Gamma[M][[(e_0 \simeq e_1)^{\sigma}]] = \Gamma[M][[e_0^{\sigma} \simeq e_1^{\sigma}]]$, $\text{Coh}(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is $(m - r(L^*))$ -complete w.r.t. $[X \cup \text{Dom}(\Gamma), A \cup \text{Atoms}(\text{Rng}(\sigma))]$. Now

$$\begin{aligned} \vdash \pi &\Rightarrow L^*[[e_0 \simeq e_1]] \Leftrightarrow \Gamma[M][[(e_0 \simeq e_1)^{\sigma}]] \quad \text{by lemma (1a).} \\ \vdash \pi &\Rightarrow \Gamma[M][[(\pi_{\Gamma})_M]] \quad \text{by lemma (CMI).} \end{aligned}$$

Now let $\pi' = (\pi_\Gamma)_M$ then by lemma **(2.i)**, for $i < 2$, either e_i^σ reduces to a π' -stuck state, or else there exists a memory context Γ_i , a modification M_i , and a v_i such that $e_i^\sigma \xrightarrow{*}_{\pi'} \Gamma_i[M_i[v_i]]$ and $\text{Coh}(\pi', \Gamma_i; M_i)$. Thus apriori there are three cases to consider: when e_0^σ and e_1^σ both reduce to π' -stuck states; when exactly one of e_0^σ and e_1^σ reduce to a π' -stuck state; and when neither e_0^σ nor e_1^σ reduces to a π' -stuck state.

(2a.) Suppose e_0^σ and e_1^σ both reduce to π' -stuck states. Thus $e_i^\sigma \xrightarrow{*}_{\pi'} \Gamma_i[M_i[R[e'_i]]]$, and $e'_i \in \{\mathbf{get}(v), \mathbf{set}(v, v')\}$, with $(\pi'_\Gamma)_{M_i} \models \mathbf{cell}?(v) \sim \mathbf{nil}$. In this situation we can use lemma **(0)**, the axioms for undefinedness, **(U)**, and corollary **(cmi)** we have that

$$\vdash \pi' \Rightarrow \uparrow \Gamma_i[M_i[R[e'_i]]] \quad \text{for } i < 2$$

And so by lemma **(1.i)** and the rules concerning undefinedness, **(U)**,

$$\vdash \pi' \Rightarrow e_0^\sigma \simeq e_1^\sigma \quad \text{Thus} \quad \vdash \pi \Rightarrow L^*[e_0 \simeq e_1]$$

and so again any tautological complex constraint suffices. \square_{2a} .

(2b.) Suppose, without loss of generality that only e_0^σ reduces to a stuck state. Thus $e_0^\sigma \xrightarrow{*}_{\pi'} \Gamma_0[M_0[R[e'_0]]]$, and $e'_0 \in \{\mathbf{get}(v), \mathbf{set}(v, v')\}$, with $(\pi'_\Gamma)_M \models \mathbf{cell}?(v) \sim \mathbf{nil}$. Again we use lemma **(0)**, the axioms for undefinedness, **(U)**, and **(cmi)** we have that

$$\vdash \pi' \Rightarrow \uparrow \Gamma_0[M_0[R[e'_0]]]$$

$$\vdash \pi' \Rightarrow \uparrow e_0^\sigma$$

Now on the other hand there exists a memory context Γ_1 , a modification M_1 , and a v_1 such that $e_1^\sigma \xrightarrow{*}_{\pi'} \Gamma_1[M_1[v_1]]$. Thus

$$\begin{aligned} \vdash \pi' \Rightarrow \downarrow \Gamma_1[M_1[v_1]] & \quad \text{using } \text{Coh}(\pi', \Gamma_1; M_1) \text{ and } \mathbf{(U)} \\ \vdash \pi' \Rightarrow \downarrow e_1^\sigma & \quad \text{by lemma } \mathbf{(1.i)} \\ \vdash \pi' \Rightarrow \neg(e_0^\sigma \simeq e_1^\sigma) & \quad \text{by the above and } \mathbf{(U.i)} \\ \vdash \pi \Rightarrow L^*[\neg(e_0 \simeq e_1)] & \quad \text{by the above and } \mathbf{(P.cut)} \\ \vdash \pi \Rightarrow \neg L^*[e_0 \simeq e_1] & \quad \text{by } \mathbf{(P.not)} \text{ and since } \vdash \pi \Rightarrow \neg L^*[\mathbf{False}] \end{aligned}$$

Thus any tautologically false complex constraint $\tilde{\pi}$ will suffice. \square_{2b} .

(2c.) Suppose neither e_0^σ nor e_1^σ reduces to a π' -stuck state. So by assumption for $i < 2$ there exists a memory context Γ_i , a modification M_i , and a v_i such that $e_i^\sigma \xrightarrow{*}_{\pi'} \Gamma_i[M_i[v_i]]$ and $\text{Coh}(\pi', \Gamma_i; M_i)$. Thus

$$\begin{aligned} \vdash \pi' \Rightarrow (e_i^\sigma \simeq \Gamma_i[M_i[v_i]]) & \quad \text{by lemma } \mathbf{(1.i)}. \\ \vdash \pi' \Rightarrow (e_0^\sigma \simeq e_1^\sigma \Leftrightarrow (\Gamma_0[M_0[v_0]] \simeq \Gamma_1[M_1[v_1]])) & \end{aligned}$$

Thus

$$\vdash \pi \Rightarrow (L^*[e_0 \simeq e_1] \Leftrightarrow (\Gamma[M][\Gamma_0[M_0[v_0]] \simeq \Gamma_1[M_1[v_1]]]))$$

Consequently by lemma **(4)** we obtain the desired $\tilde{\pi}$. Note that by **(S)**, and coherence we have

$$\vdash \pi \Rightarrow \Gamma[M][\tilde{\pi}] \Leftrightarrow \tilde{\pi}$$

since the $\tilde{\pi}$ provided by lemma **(4)** is static. \square_{2c} \square_{C-1}

5.5 Proofs of the Lemmas

Lemma (0): If $\tilde{\pi}_0$ and $\tilde{\pi}_1$ are complex constraints, then

$$\tilde{\pi}_0 \models \tilde{\pi}_1 \quad \text{iff} \quad \vdash \tilde{\pi}_0 \Rightarrow \tilde{\pi}_1.$$

Proof (0): Nelson and Oppen [41]. \square_0

Lemma (1):

- (i) If $e \mapsto_{\pi}^* e'$, then $\vdash \pi \Rightarrow e \sim e'$.
- (ii) If $L^*[\Phi] \mapsto_{\pi}^* L'^*[\Phi']$, then $\vdash \pi \Rightarrow (L^*[\Phi] \Leftrightarrow L'^*[\Phi'])$.

Proof (1): It suffices to show that if $\text{Coh}(\pi, \Gamma; M)$, then

- (i) $\Gamma[M[e]] \mapsto_{\pi}^* \Gamma'[M'[e']]$ implies $\vdash \pi \Rightarrow (\Gamma[M[e]] \sim \Gamma'[M'[e']])$
- (ii) $\Gamma[M][\Phi] \mapsto_{\pi}^* \Gamma'[M'][\Phi']$ implies $\vdash \pi \Rightarrow (\Gamma[M][\Phi] \Leftrightarrow \Gamma'[M'][\Phi'])$

Let $\pi' = (\pi_{\Gamma})_M$ and observe that the proof naturally divides into cases corresponding to the definitions of \mapsto_{π}^* . We begin by proving **(1.i)**.

Proof (1.i):

(β_v) Assume that $e = R[\text{app}(\lambda x.e_0, v)]$. In this case:

$$\begin{aligned} \vdash \text{app}(\lambda x.e_0, v) &\sim e_0^{\{x:=v\}} && \text{by axiom (E.v).} \\ \vdash \pi' \Rightarrow \text{app}(\lambda x.e_0, v) &\sim e_0^{\{x:=v\}} && \text{by axioms (P).} \\ \vdash \pi \Rightarrow \Gamma[M[R]][\text{app}(\lambda x.e_0, v) &\sim e_0^{\{x:=v\}}] && \text{by lemmas (CMI), and (P.cut).} \\ \vdash \pi \Rightarrow \Gamma[M[R[\text{app}(\lambda x.e_0, v)]]] &\sim \Gamma[M[R[e_0^{\{x:=v\}}]]] && \text{by axioms (P), and (C.ii).} \end{aligned}$$

(τ) Assume that $e = R[\tau(v)]$ for $\tau \in \mathbb{T}$. In this case:

$$\begin{aligned} \pi' \models \tau(v) &\sim \mathbf{b} && \text{by assumption, for } \mathbf{b} \in \{\mathbf{t}, \mathbf{nil}\}. \\ \vdash \pi' \Rightarrow \tau(v) &\sim \mathbf{b} && \text{by lemma (0).} \\ \vdash \pi \Rightarrow \Gamma[M[R]][\tau(v) &\sim \mathbf{b}] && \text{by lemmas (CMI) and (P.cut).} \\ \vdash \pi \Rightarrow \Gamma[M[R[\tau(v)]]] &\sim \Gamma[M[R[\mathbf{b}]]] && \text{by axioms (P) and (C.ii).} \end{aligned}$$

(eq.t) Assume that $e = R[\text{eq}(v_0, v_1)]$ and $e' = R[\mathbf{t}]$. In this case:

$$\begin{aligned} \pi' \models \Psi_{\mathbf{t}}(v_0, v_1) &&& \text{by assumption.} \\ \pi' \models v_0 \sim v_1 \wedge \text{atom}?(v_0) &\sim \mathbf{t} \wedge \text{atom}?(v_1) \sim \mathbf{t} && \text{by definition of } \Psi_{\mathbf{t}}. \\ \vdash \pi' \Rightarrow (v_0 \sim v_1 \wedge \text{atom}?(v_0) &\sim \mathbf{t} \wedge \text{atom}?(v_1) \sim \mathbf{t}) && \text{by lemma (0).} \\ \vdash \pi' \Rightarrow \text{eq}(v_0, v_1) &\sim \mathbf{t} && \text{by axiom (eq.ii).} \\ \vdash \pi \Rightarrow \Gamma[M[R]][\text{eq}(v_0, v_1) &\sim \mathbf{t}] && \text{by lemmas (CMI) and (P.cut).} \\ \vdash \pi \Rightarrow \Gamma[M[R[\text{eq}(v_0, v_1)]]] &\sim \Gamma[M[R[\mathbf{t}]]] && \text{by axioms (P) and (C.ii).} \end{aligned}$$

(eq.nil) Assume that $e = R[\mathbf{eq}(v_0, v_1)]$ and $e' = R[\mathbf{nil}]$. In this case:

$\pi' \models \neg \Psi_{\mathbf{t}}(v_0, v_1)$	by assumption.
$\pi' \models \neg(v_0 \sim v_1 \wedge \mathbf{atom}?(v_0) \sim \mathbf{t} \wedge \mathbf{atom}?(v_1) \sim \mathbf{t})$	by definition of $\Psi_{\mathbf{t}}$.
$\vdash \pi' \Rightarrow \neg(v_0 \sim v_1 \wedge \mathbf{atom}?(v_0) \sim \mathbf{t} \wedge \mathbf{atom}?(v_1) \sim \mathbf{t})$	by lemma (0) .
$\vdash \pi' \Rightarrow \neg(\mathbf{eq}(v_0, v_1) \sim \mathbf{t})$	by axiom (eq.ii) .
$\vdash \pi' \Rightarrow \mathbf{eq}(v_0, v_1) \sim \mathbf{nil}$	by axioms (eq.i) and (P) .
$\vdash \pi \Rightarrow \Gamma[M[R]][\mathbf{eq}(v_0, v_1) \sim \mathbf{nil}]$	by lemmas (CMI) and (P.cut) .
$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{eq}(v_0, v_1)]]] \sim \Gamma[M[R[\mathbf{nil}]]]$	by axioms (P) and (C.ii) .

(brt) Assume that $e = R[\mathbf{br}(v_0, v_1, v_2)]$ and that $\pi' \models \neg(v_0 \sim \mathbf{nil})$. In this case:

$\pi' \models \neg(v_0 \sim \mathbf{nil})$	by assumption.
$\vdash \pi' \Rightarrow \neg(v_0 \sim \mathbf{nil})$	by lemma (0) .
$\vdash \pi' \Rightarrow \mathbf{br}(v_0, v_1, v_2) \sim v_1$	by axioms (br.ii) and (P) .
$\vdash \pi \Rightarrow \Gamma[M[R]][\mathbf{br}(v_0, v_1, v_2) \sim v_1]$	by lemmas (CMI) and (P.cut) .
$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{br}(v_0, v_1, v_2)]]] \sim \Gamma[M[R[v_1]]]$	by axioms (P) and (C.ii) .

(brnil) Assume that $e = R[\mathbf{br}(v_0, v_1, v_2)]$ and that $\pi' \models v_0 \sim \mathbf{nil}$. In this case:

$\pi' \models v_0 \sim \mathbf{nil}$	by assumption.
$\vdash \pi' \Rightarrow v_0 \sim \mathbf{nil}$	by lemma (0) .
$\vdash \pi' \Rightarrow \mathbf{br}(v_0, v_1, v_2) \sim v_2$	by axioms (br.i) and (P) .
$\vdash \pi \Rightarrow \Gamma[M[R]][\mathbf{br}(v_0, v_1, v_2) \sim v_2]$	by lemmas (CMI) and (P.cut) .
$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{br}(v_0, v_1, v_2)]]] \sim \Gamma[M[R[v_2]]]$	by axioms (P) and (C.ii) .

(mk) Assume that $e = R[\mathbf{mk}(v)]$. In this case:

$\vdash R[\mathbf{mk}(v)] \sim \mathbf{let}\{z := \mathbf{mk}(v)\}R[z]$	
for z fresh, by axiom (E.vi) .	
$\vdash \Gamma[M][R[\mathbf{mk}(v)] \sim \mathbf{let}\{z := \mathbf{mk}(v)\}R[z]]$	
by rule (C.i) .	
$\vdash \Gamma[M[R[\mathbf{mk}(v)]]] \sim \Gamma[M[\mathbf{let}\{z := \mathbf{mk}(v)\}R[z]]]$	
by axiom (C.ii) .	
$\vdash \Gamma[M[R[\mathbf{mk}(v)]]] \sim \Gamma[\mathbf{let}\{z := \mathbf{mk}(v)\}[M[R[z]]]$	
by axioms (E) , (C.i) , and (mk.ii) .	
$\vdash \Gamma[M[R[\mathbf{mk}(v)]]] \sim \Gamma[\mathbf{let}\{z := \mathbf{mk}(\mathbf{nil})\}\mathbf{seq}(\mathbf{set}(z, v), [M[R[z]])]$	
by axioms (set.v) , (C.i) and (E) .	

$$\vdash \Gamma[M[R[\mathbf{mk}(v)]]] \sim \Gamma\{z := \mathbf{mk}(v)\}[M[R[z]]]$$

by axioms **(mk.ii)**, **(C.i)** and **(E)**.

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{mk}(v)]]] \sim \Gamma\{z := \mathbf{mk}(v)\}[M[R[z]]]$$

by axioms **(P)**.

(get) Assume that $e = R[\mathbf{get}(z)]$ and $e' = R[v]$. In this case:

$$\pi' \models \mathbf{get}(z) \sim v$$

by assumption.

$$\vdash \pi' \Rightarrow \mathbf{get}(z) \sim v$$

by lemma **(0)**.

$$\vdash \pi \Rightarrow \Gamma[M[R]][\mathbf{get}(z) \sim v]$$

by lemmas **(CMI)** and **(P.cut)**.

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{get}(z)]]] \sim \Gamma[M[R[v]]]$$

by axioms **(C.ii)** and **(P)**.

(set) Assume that $e = \mathbf{set}(z, v)$.

$$\vdash \mathbf{set}(z, v) \sim \mathbf{seq}(\mathbf{set}(z, v), \mathbf{nil})$$

by axioms **(E.v)**, **(C.ii)**, **(set.i)**, and **(P)**.

$$\vdash \pi \Rightarrow \Gamma[M[R]][\mathbf{set}(z, v) \sim \mathbf{seq}(\mathbf{set}(z, v), \mathbf{nil})]$$

by lemmas **(CMI)**, **(P.cut)**, and **(P)**.

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{set}(z, v)]]] \sim \Gamma[M[R[\mathbf{seq}(\mathbf{set}(z, v), \mathbf{nil})]]]$$

by axioms **(C.ii)**, and **(P)**.

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{set}(z, v)]]] \sim \Gamma[M[\mathbf{seq}(\mathbf{set}(z, v), R[\mathbf{nil}])]]$$

by axioms **(E.vi)**, **(P)**, **(C.i)**, and **(E)**.

Now we consider two cases: $z \in \text{Dom}(\Gamma)$; and $z \notin \text{Dom}(\Gamma)$. In the former case we have:

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{set}(z, v)]]] \sim \Gamma[M[\mathbf{seq}(\mathbf{set}(z, v), R[\mathbf{nil}])]]$$

by the above.

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{set}(z, v)]]] \sim \Gamma[\mathbf{seq}(\mathbf{set}(z, v), M[R[\mathbf{nil}])]]$$

by axioms **(set.iii)**, **(P)**, **(C.i)**, and **(E)**.

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{set}(z, v)]]] \sim \Gamma\{z := \mathbf{mk}(v)\}[M[R[\mathbf{nil}]]]$$

by axioms **(set.iii)**, **(set.iv)**, **(C.i)**, **(P)**, and **(E)**.

In the latter case we have:

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{set}(z, v)]]] \sim \Gamma[M[\mathbf{seq}(\mathbf{set}(z, v), R[\mathbf{nil}])]]$$

by the above.

$$\vdash \pi \Rightarrow \Gamma[M[R[\mathbf{set}(z, v)]]] \sim \Gamma[\Gamma[M\{z := \mathbf{mk}(v)\}]_{\pi}[R[\mathbf{nil}]]]$$

by axioms **(set.iii)**, **(set.iv)**, **(C.i)**, **(P)**, and **(E)**.

□_{1a}

Note that in every case other than **(mk)** and **(set)** we actually prove the stronger result that

$$\Gamma[M[R[e]]] \stackrel{*}{\mapsto}_{\pi} \Gamma[M[R[e']]] \text{ implies } \vdash \pi \Rightarrow \Gamma[M[R]] \llbracket e \sim e' \rrbracket.$$

This is useful in the proof of the second part of the lemma.

Proof (1.ii): Assume that $\text{Coh}(\pi, \Gamma; M)$. Then we have two cases:

$$\begin{aligned} \text{(val)} \quad & \Gamma[M[\mathbf{let}\{x := v\}\bullet]] \llbracket \Phi \rrbracket \mapsto_{\pi} \Gamma[M][\Phi^{x:=v}] \\ \text{(red)} \quad & \Gamma[M[\mathbf{let}\{x := e\}\bullet]] \llbracket \Phi \rrbracket \mapsto_{\pi} \Gamma_1[M_1[\mathbf{let}\{x := e_1\}\bullet]] \llbracket \Phi \rrbracket \\ & \text{where } \Gamma[M[e]] \mapsto_{\pi} \Gamma_1[M_1[e_1]] \end{aligned}$$

(val) In this case:

$$\begin{aligned} \vdash \mathbf{let}\{x := v\} \llbracket \Phi \rrbracket & \Leftrightarrow \Phi^{x:=v} && \text{by axiom (C.iv).} \\ \vdash \pi' \Rightarrow \mathbf{let}\{x := v\} \llbracket \Phi \rrbracket & \Leftrightarrow \Phi^{x:=v} && \text{by axioms (P).} \\ \vdash \pi \Rightarrow \Gamma[M][\mathbf{let}\{x := v\} \llbracket \Phi \rrbracket] & \Leftrightarrow \Phi^{x:=v} && \text{by lemmas (CMI) and (P.cut).} \\ \vdash \pi \Rightarrow \Gamma[M][\mathbf{let}\{x := v\} \llbracket \Phi \rrbracket] & \Leftrightarrow \Gamma[M][\Phi^{x:=v}] && \text{by lemma (CMI).} \\ \vdash \pi \Rightarrow \Gamma[M[\mathbf{let}\{x := v\}\bullet]] \llbracket \Phi \rrbracket & \Leftrightarrow \Gamma[M][\Phi^{x:=v}] && \text{by definition.} \end{aligned}$$

(red) Here we consider three separate subcases depending on the nature of the reduction $\Gamma[M[e]] \mapsto_{\pi} \Gamma[M'[e']]$. If this does not involve **(mk)** or **(set)**, then by the stronger version of lemma **(1.i)** we have that

$$\vdash \pi \Rightarrow \Gamma[M] \llbracket e \sim e' \rrbracket.$$

Thus by axioms **(C.iii)**, **(P)**, and lemma **(P.cut)** we obtain the desired conclusion:

$$\vdash \pi \Rightarrow (\Gamma[M[\mathbf{let}\{x := e\}\bullet]] \llbracket \Phi \rrbracket \Leftrightarrow \Gamma[M[\mathbf{let}\{x := e'\}\bullet]] \llbracket \Phi \rrbracket).$$

Thus we are only left with the cases when the reductions involves **(mk)** or **(set)**.

(mk) Let $R[\bullet_1, \bullet_2]$ abbreviate $\mathbf{let}\{x := \bullet_1\}\bullet_2$. In this case we need to show that

$$\vdash \pi \Rightarrow (\Gamma[M[R[\mathbf{mk}(v), \bullet]]] \llbracket \Phi \rrbracket \Leftrightarrow \Gamma\{z := \mathbf{mk}(v)\}[M[R[z, \bullet]]] \llbracket \Phi \rrbracket) \quad \text{for } z \text{ fresh.}$$

To this end:

$$\begin{aligned} \vdash \Gamma[M[R[\mathbf{mk}(v), \bullet]]] \llbracket \Phi \rrbracket & \Leftrightarrow \Gamma[M[\mathbf{let}\{z := \mathbf{mk}(v)\}R[z, \bullet]]] \llbracket \Phi \rrbracket \\ & \text{by axioms (C.v), (P) and rule (C.i).} \\ \vdash \Gamma[M[R[\mathbf{mk}(v), \bullet]]] \llbracket \Phi \rrbracket & \Leftrightarrow \Gamma[\mathbf{let}\{z := \mathbf{mk}(v)\}[M[R[z, \bullet]]] \llbracket \Phi \rrbracket] \\ & \text{by axioms (mk.iii) and (P).} \\ \vdash \Gamma[M[R[\mathbf{mk}(v), \bullet]]] \llbracket \Phi \rrbracket & \Leftrightarrow \Gamma[\mathbf{let}\{z := \mathbf{mk}(\mathbf{nil})\}\mathbf{seq}(\mathbf{set}(z, v), \{M[R[z, \bullet]]\}) \llbracket \Phi \rrbracket] \\ & \text{by axioms (set.v), (C.iii) and (P) and rule (C.i).} \\ \vdash \Gamma[M[R[\mathbf{mk}(v), \bullet]]] \llbracket \Phi \rrbracket & \Leftrightarrow \Gamma\{z := \mathbf{mk}(v)\}[M[R[z, \bullet]]] \llbracket \Phi \rrbracket \\ & \text{by (mk.iii) and (P).} \end{aligned}$$

(set) Again let $R[\bullet_1, \bullet_2]$ abbreviate $\mathbf{let}\{x := \bullet_1\}\bullet_2$. In this case $e = \mathbf{set}(z, v)$ and we must consider two possibilities, either $z \in \text{Dom}(\Gamma)$ or not. In the former case we must show:

$$\vdash \pi \Rightarrow (\Gamma[M[R[\mathbf{set}(z, v), \bullet]]][\Phi] \Leftrightarrow \Gamma\{z := \mathbf{mk}(v)\}[M[R[\mathbf{nil}, \bullet]]][\Phi])$$

While in the latter case we must show

$$\vdash \pi \Rightarrow (\Gamma[M[R[\mathbf{set}(z, v), \bullet]]][\Phi] \Leftrightarrow \Gamma[M\{z := \mathbf{mk}(v)\}]_{\pi}[R[\mathbf{nil}, \bullet]][\Phi])$$

In either case we begin by observing that:

$$\vdash \mathbf{set}(z, v) \sim \mathbf{seq}(\mathbf{set}(z, v), \mathbf{nil}) \quad \text{by axioms (E.v), (C.ii), (set.i), and (P).}$$

$$\vdash R[\mathbf{set}(z, v), \bullet][\Phi] \Leftrightarrow R[\mathbf{seq}(\mathbf{set}(z, v), \mathbf{nil}), \bullet][\Phi] \quad \text{by axiom (C.iii).}$$

$$\vdash R[\mathbf{set}(z, v), \bullet][\Phi] \Leftrightarrow \mathbf{seq}(\mathbf{set}(z, v), R[\mathbf{nil}, \bullet])[\Phi] \quad \text{by axioms (C.v), and (E).}$$

$$\vdash \pi_{\Gamma} \Rightarrow M[R[\mathbf{set}(z, v), \bullet]][\Phi] \Leftrightarrow M[\mathbf{seq}(\mathbf{set}(z, v), R[\mathbf{nil}, \bullet])][\Phi]$$

by axioms (P), (C.i), and (C.ii).

$$\vdash \pi_{\Gamma} \Rightarrow M[R[\mathbf{set}(z, v), \bullet]][\Phi] \Leftrightarrow \mathbf{seq}(M[\mathbf{set}(z, v)], R[\mathbf{nil}, \bullet])[\Phi]$$

by axioms (P), and (C.v).

Now the latter case is simple since

$$\vdash M[\mathbf{set}(z, v)] \sim M\{z := \mathbf{mk}(v)\}_{\pi}[\mathbf{nil}]$$

by the proof of lemma (1.i).

$$\vdash \pi_{\Gamma} \Rightarrow (M[R[\mathbf{set}(z, v), \bullet]][\Phi] \Leftrightarrow \mathbf{seq}(M\{z := \mathbf{mk}(v)\}_{\pi}[\mathbf{nil}], R[\mathbf{nil}, \bullet])[\Phi])$$

by axioms (P), and (C.iii).

$$\vdash \pi_{\Gamma} \Rightarrow (M[R[\mathbf{set}(z, v), \bullet]][\Phi] \Leftrightarrow M\{z := \mathbf{mk}(v)\}_{\pi}[R[\mathbf{nil}, \bullet]][\Phi])$$

by axioms (C.iv), (P), and (C.iii).

$$\vdash \pi \Rightarrow (\Gamma[M[R[\mathbf{set}(z, v), \bullet]]][\Phi] \Leftrightarrow \Gamma[M\{z := \mathbf{mk}(v)\}]_{\pi}[R[\mathbf{nil}, \bullet]][\Phi])$$

by (P.cut) and (P).

Now in the former case we have:

$$\vdash \pi_{\Gamma} \Rightarrow M[\mathbf{set}(z, v)] \sim \mathbf{seq}(\mathbf{set}(z, v), M[\mathbf{nil}])$$

again by the proof of lemma (1.i).

$$\vdash \pi_{\Gamma} \Rightarrow (M[R[\mathbf{set}(z, v), \bullet]][\Phi] \Leftrightarrow \mathbf{seq}(\mathbf{set}(z, v), M[\mathbf{nil}], R[\mathbf{nil}, \bullet])[\Phi])$$

by axioms (P), (C.iii), (C.i), (C.v).

$$\vdash \pi_{\Gamma} \Rightarrow (M[R[\mathbf{set}(z, v), \bullet]][\Phi] \Leftrightarrow \mathbf{seq}(\mathbf{set}(z, v), M[R[\mathbf{nil}, \bullet]][\Phi]))$$

by axioms (C.iii), (C.iv), (C.i), (C.v) and (P).

$$\vdash \pi \Rightarrow (\Gamma[M[R[\mathbf{set}(z, v), \bullet]]][\Phi] \Leftrightarrow \Gamma[\mathbf{seq}(\mathbf{set}(z, v), M[R[\mathbf{nil}, \bullet]][\Phi])])$$

by lemma (P.cut) and axioms (C.ii) and (P).

$$\vdash \pi \Rightarrow (\Gamma[M[R[\mathbf{set}(z, v), \bullet]]][\Phi] \Leftrightarrow \Gamma\{x := \mathbf{mk}(v)\}M[R[\mathbf{nil}, \bullet]][\Phi])$$

by axioms (mk.i), (set.iii) (set.iv) (C.v) and (C.iii).

□₁.(ii) □₁

Lemma (2): Assume $e, L^*[[\Phi]]$ are first order, $FV(e, L^*) \subseteq X$, and $Atoms(\pi, e, L^*) \subseteq A$.

- (i) If π is $(r(e) + m)$ -complete w.r.t. $[X, A]$ and $Coh(\pi)$, then either e reduces to a π -stuck state, or else there exists a memory context Γ , a modification M , and a v such that $e \xrightarrow{*}_{\pi} \Gamma[M[v]]$, $Coh(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is m -complete w.r.t. $[X \cup Dom(\Gamma), A \cup Atoms(v)]$.
- (ii) If π is $(r(L^*) + m)$ -complete w.r.t. $[X, A]$ and $Coh(\pi)$, then either $L^*[[\Phi]]$ reduces to a π -stuck state, or else there exists a memory context Γ , a modification M and a substitution σ such that $L^*[[\Phi]] \xrightarrow{*}_{\pi} \Gamma[M][[\Phi^{\sigma}]]$, $Coh(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is m -complete w.r.t. $[X \cup Dom(\Gamma), A \cup Atoms(Rng(\sigma))]$.

Proof (2): These both follow from the simple observation that if $e \mapsto_{\pi} e'$ and π is $(r(e) + m)$ -complete w.r.t. $[X, A]$, then π is $(r(e') + m)$ -complete w.r.t. $[X, A]$. □₂

Lemma (3): For any consistent $\pi, \bar{x}, A \in \mathbf{P}_{\omega}(\mathbb{A})$, and $n \in \mathbb{N}$ there exists $N \in \mathbb{N}$ and a family of constraint sets $\{\pi_i\}_{i < N}$ such that

- (i) Each π_i is n -complete w.r.t. $[\bar{x}, Atoms(\pi_i, A)]$, and $Coh(\pi_i)$.
- (ii) $\vdash \pi \Leftrightarrow (\bigvee_{i < N} \pi_i)$

Proof (3): Suppose that π is consistent but not n -complete w.r.t. $[\bar{x}, Atoms(A)]$. Pick Θ, Θ_0 , accessor chains of length $\leq n$, and $y, y_0 \in \bar{x}$ such that $\pi \models \Theta[y] \sim v$ and $\pi \models \Theta_0[y_0] \sim v_0$ and one of the conditions (1), ..., (4) fails. We repair each possible failure in turn. If condition (1) fails then

$$\vdash \pi \Leftrightarrow ((\pi \cup \{\tau(v) \sim \mathbf{t}\}) \vee (\pi \cup \{\tau(v) \sim \mathbf{nil}\}))$$

by $(\tau.i)$, for $\tau \in \mathbb{T}$. If (2) fails then

$$\vdash \pi \Leftrightarrow ((\pi \cup \{v \sim \alpha\}) \vee (\pi \cup \{\neg(v \sim \alpha)\}))$$

by propositional logic (P), for $\alpha \in A \cup \{\mathbf{t}, \mathbf{nil}, v_0\}$. If (3) fails then

$$\vdash \pi \Leftrightarrow (\pi \cup \{\mathbf{get}(v) \sim z\})$$

for z fresh, by (U.iv) and existential elimination ($\exists E$). If (4) fails, then π is inconsistent by (U.iv) and ($\exists I$). This contradicts our initial assumptions. Thus in each (possible) case it is possible to enlarge π or branch and enlarge so as to rectify this particular failure. Generating the required family $\{\pi_i \mid i < N\}$ is now trivial. □₃

Lemma (4): Suppose π and $e_i = \Gamma_i[M_i[v_i]]$, $i < 2$, are such that $Coh(\pi, \Gamma_i; M_i)$ for $i < 2$, and π is 1-complete with respect to $[FV(e_0) \cup FV(e_1), Atoms(e_0, e_1)]$. Then there are static complex constraints $\tilde{\pi}_i$ and $\tilde{\pi}_{ii}$ such that

- (i) $\vdash \pi \Rightarrow (\tilde{\pi}_i \Leftrightarrow e_0 \sim e_1)$
- (ii) $\vdash \pi \Rightarrow (\tilde{\pi}_{ii} \Leftrightarrow e_0 \cong e_1)$

Proof (4): Define M_i^{π} as follows:

$$M_i^{\pi}(z) = \begin{cases} M_i(w) & \text{if } w \in Dom(M_i) \text{ and } \pi \models w \sim z \\ w & \text{if } \pi \models w \sim \mathbf{get}(z) \text{ and } (\forall y \in Dom(M_i))(\pi \models \neg(z \sim y)) \end{cases}$$

The supposition that $Coh(\pi, \Gamma_i; M_i)$ and π is 1-complete with respect to $[FV(e_i), Atoms(e_i)]$ suffice to ensure that M_i^{π} is defined on $Cells(\pi)$. Actually M_i^{π} as defined is a relation modulo π -equivaluedness, but this suffices for our purposes.

(4.(i)) Without loss of generality we may assume that $\text{Dom}(\Gamma_0)$ and $\text{Dom}(\Gamma_1)$ are disjoint and of the same cardinality, $|\text{Dom}(\Gamma_0)| = |\text{Dom}(\Gamma_1)|$. Since otherwise any tautologically false static constraint would suffice. Now for each bijection

$$f : \text{Dom}(\Gamma_0) \rightarrow \text{Dom}(\Gamma_1)$$

define π_f to be the set

$$\{x_i \sim f(x_i), \Gamma_0(x_i) \sim \Gamma_1(f(x_i)), M_0^\pi(z) \sim M_1^\pi(z), v_0 \sim v_1 \mid x_i \in \text{Dom}(\Gamma_0), z \in \text{Cells}(\pi)\}$$

Observe that π_f is a set of static constraints. The desired static complex constraint is just

$$\tilde{\pi}_i = \bigvee \{ \pi_f \mid f : \text{Dom}(\Gamma_0) \rightarrow \text{Dom}(\Gamma_1) \quad \text{a bijection} \}$$

□_{4.(i)}

(4.(ii)) This case is slightly more complex than the previous one. We begin by defining two increasing sequences, $\{\delta_j^i \mid j \in \mathbb{N}\}$, of subsets of $\text{Dom}(\Gamma_i)$ for $i < 2$ by induction.

$$\delta_0^i = \text{Dom}(\Gamma_i) \cap \text{FV}(M_i[v_i])$$

and having defined δ_j^i we define δ_{j+1}^i by:

$$\delta_{j+1}^i = \delta_j^i \cup \{z \in \text{Dom}(\Gamma_i) \mid (\exists x \in \delta_j^i)(z \in \text{FV}(\Gamma_i(x)))\}$$

Put $\delta^i = \bigcup_{j \in \mathbb{N}} \delta_j^i$. We claim that $\text{Dom}(\Gamma_i) - \delta^i$ is precisely the garbage created by e_i . Consequently we may assume that the two sets δ^0 and δ^1 are disjoint and of the same cardinality, $|\delta^0| = |\delta^1|$, since otherwise any tautologically static constraint would suffice. We now proceed in the same fashion as in the previous case. for each bijection

$$f : \delta^0 \rightarrow \delta^1$$

define π_f to be the set

$$\{x_i \sim f(x_i), \Gamma_0(x_i) \cong \Gamma_1(f(x_i)), M_0^\pi(z) \cong M_1^\pi(z), v_0 \cong v_1 \mid x_i \in \text{Dom}(\Gamma_0), z \in \text{Cells}(\pi)\}$$

Observe that π_f is a set of static constraints. The desired static complex constraint is just

$$\tilde{\pi}_{ii} = \bigvee \{ \pi_f \mid f : \delta^0 \rightarrow \delta^1 \quad \text{a bijection} \}$$

□_{4.(ii)}

6 The Second Completeness Theorem

We now demonstrate that the quantifier free assumption can be eliminated from the completeness result of the previous section.

Definition (II): The set of *quantified constraints*, $\check{\Pi}$ is defined by:

$$\check{\Pi} := \pm(\mathbb{V}^\circ \simeq \mathbb{V}^\circ) + (\mathbb{F}_1 - \{\mathbf{mk}\})(\mathbb{V}^\circ \simeq \mathbb{V}^\circ) + (\check{\Pi} \Rightarrow \check{\Pi}) + (\forall \mathbb{X})\check{\Pi}$$

Theorem (Completeness – II): If $\Phi \in \mathbb{W}^\circ$ is first order, then there is a quantified constraint $\check{\pi}$ such that

$$\vdash \check{\pi} \Leftrightarrow \Phi$$

Proof (Completeness – II): We prove this by induction on the quantifier rank of our first order Φ . The only new case we need to consider in this more general situation is how to simplify a formula of the form $L^*[(\forall x)\Phi_0]$. Fix a particular $L^*[(\forall x)\Phi]$ and let $\text{FV}(L^*[(\forall x)\Phi]) = X$ and $\text{Atoms}(L^*[(\forall x)\Phi]) = A$. By propositional logic and lemma (3) we need only show that there is a complex constraint $\tilde{\pi}$ such that

$$\vdash \pi \Rightarrow (\tilde{\pi} \Leftrightarrow L^*[(\forall x)\Phi])$$

assuming that π is m -complete w.r.t. $[X, \text{Atoms}(\pi, A)]$, and $\text{Coh}(\pi)$ for suitably large m ($m \geq 1 + r(L^*) + \max(r((\forall x)\Phi)$). Now by lemma (2.ii) either $L^*[(\forall x)\Phi]$ reduces to a π -stuck state, or else there exists a memory context Γ , a modification M and a substitution σ such that $L^*[(\forall x)\Phi] \xrightarrow{*}_{\pi} \Gamma[M][[(\forall x)\Phi]^{\sigma}]$, $\text{Coh}(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is $(m - r(L^*))$ -complete w.r.t. $[X \cup \text{Dom}(\Gamma), A \cup \text{Atoms}(\text{Rng}(\sigma))]$. We should point out that we are not using a more general version of (2.ii) than the one stated, since there are no restrictions on the formula inside the contextual assertion.

The first case proceeds exactly as in the proof of the first completeness theorem. We are left to deal with the second case.

(2.) Suppose $L^*[(\forall x)\Phi] \xrightarrow{*}_{\pi} \Gamma[M][[(\forall x)\Phi]^{\sigma}] = \Gamma[M][[e_0^{\sigma} \simeq e_1^{\sigma}]]$, $\text{Coh}(\pi, \Gamma; M)$ and $(\pi_{\Gamma})_M$ is $(m - r(L^*))$ -complete w.r.t. $[X \cup \text{Dom}(\Gamma), A \cup \text{Atoms}(\text{Rng}(\sigma))]$. Now

$$\vdash \pi \Rightarrow (L^*[(\forall x)\Phi] \Leftrightarrow \Gamma[M][[(\forall x)\Phi]^{\sigma}]) \quad \text{by lemma (1a).}$$

Also note that:

$$\vdash \Gamma[M][[(\forall x)\Phi]^{\sigma}] \Leftrightarrow \Gamma[M][(\forall x)(\Phi^{\sigma})] \quad \text{given obvious hygiene assumptions.}$$

$$\vdash \Gamma[M][(\forall x)(\Phi^{\sigma})] \Leftrightarrow \Gamma[(\forall x)M[(\Phi^{\sigma})]] \quad \text{by repeated application of (Q.iv) and its converse}$$

$$\vdash \Gamma[(\forall x)M[(\Phi^{\sigma})]] \Leftrightarrow ((\forall x)\Gamma[M[(\Phi^{\sigma})]]) \wedge \Gamma[M[\bigwedge_{z \in \text{Dom}(\Gamma)} (\Phi^{\sigma})^{\{x:=z\}}]]$$

by repeated application of (Q.v) and its converse

We are now in a position to apply the induction hypothesis to obtain:

$$\vdash \tilde{\pi}_0 \Leftrightarrow \Gamma[M[(\Phi^{\sigma})]]$$

$$\vdash \tilde{\pi}_1 \Leftrightarrow \Gamma[M[\bigwedge_{z \in \text{Dom}(\Gamma)} (\Phi^{\sigma})^{\{x:=z\}}]]$$

Consequently

$$\vdash \pi \Rightarrow (L^*[(\forall x)\Phi] \Leftrightarrow ((\forall x)\tilde{\pi}_0 \wedge \tilde{\pi}_1))$$

and thus we reach the desired conclusion. \square

To obtain the desired correspondence between the proof theory and semantics we have to elaborate on the first lemma. We need an analysis of the theory of those first order structures in the language

`{get, atom?, cell?, lambda?}`

which satisfy the principles enumerated in §4. This theory is known to be decidable, indeed the *weak monadic second order* is shown to be decidable in Rabin's landmark S2S paper [48]. Consequently we may conclude that the provability of a first order $\Phi \in \mathbb{W}^{\circ}$ is decidable.

7 Conclusions, Directions and Future Work

In this paper we have continued our investigations into a *Variable Typed Logic of Effects* that began in [34, 23, 35, 37, 24]. In particular we presented an axiomatization of the base first order theory. In [21] we described an encoding of this

logic into the generic proof assistant *Isabelle* [43]. Encoding the syntax and proof theory of the logic was a relatively painless procedure. Especially when compared with the contortions required for logics of the Hoare and Dynamic ilk [36, 1].

Since the semantics of the underlying λ_{mk} -calculus is operational, and the semantics of the logic is defined strictly in terms of syntactic entities, it seems not unreasonable to expect an implementation to be capable of encoding it. This would allow for both proof theoretical and semantic reasoning to be carried out at the same time in the same context [56]. This would have two obvious attractions:

1. It will also allow the system to semantically verify its own proof system, an extremely attractive idea.
2. It would also allow for the dynamic enrichment of the proof theory by introducing new, semantically verified, principles. Thus the logic implemented would be truly dynamic.

The only obstacle to successfully encoding the semantics is the problem of encoding lambda calculus style contexts and hole filling (i.e the corresponding notion of substitution with variable binding capture). To achieve this it may be necessary to adopt the *binding structure* approach developed in [52, 54].

While we have presented a completeness result in this paper, substantial work still remains to be done. We mention here several open problems in this area that we expect to obtain results within the three year time-frame.

1. The axiom system presented here contains axioms and rules for quantifiers but not structured data (i.e. typically immutable pairs), however the question of whether these axioms and rules are complete remains open. We conjectured that the techniques presented in [55] can be modified and adapted to this framework to obtain an affirmative answer to this conjecture.
2. In an operational setting, the main tool for establishing principles such as structural induction, fixed-point induction, co-induction, and simulation induction is induction on the length of computation. By incorporating both semantical and proof theoretical principles into the proof environment we solve the problem of how computation induction can be formalized within this programming logic.
3. Thus far we have studied systems with control features [53] as well as systems with imperative features. The unification of these two theories is yet to be studied in detail. While most of the nice meta-theoretic properties such as completeness should carry over, certain principles must be modified if they are to remain valid [17].
4. In the long term it is hoped that our work on concurrent and distributed programming [5] and [6] will result in a unified approach to all three enrichments to the underlying functional language.

References

- [1] A. Avron, F. Honsell, I. A. Mason, and Robert Pollack. Using Typed Lambda Calculus to Implement Formal Systems on a Machine. *Journal of Automated Reasoning*, 9:309–354, 1992.
- [2] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. The MIT Press, McGraw-Hill Book Company, 1985.
- [3] S. Abramsky. The lazy lambda calculus. In D.A. Turner, editor, *Research Topics in Functional Programming*. Addison-Wesley, 1990.
- [4] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51(1):1–77, 1991.
- [5] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. Towards a theory of actor computation. In *The Third International Conference on Concurrency Theory (CONCUR '92)*, volume 630 of *Lecture Notes in Computer Science*, pages 565–579. Springer Verlag, August 1992.
- [6] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation, 1999. to appear.

- [7] K.R. Apt. Ten years of Hoare's logic: A survey—part I. *ACM Transactions on Programming Languages and Systems*, 4:431–483, 1981.
- [8] H. Barendregt. *The lambda calculus: its syntax and semantics*. North-Holland, 1981.
- [9] B. Bloom. Can LCF be topped? *Information and Computation*, 87:264–301, 1990.
- [10] C.C. Chang and H.J. Keisler. *Model Theory*. North-Holland, Amsterdam, 1973.
- [11] L. Egidi, F. Honsell, and S. Ronchi della Rocca. Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticae*, 16(2):149–170, 1992.
- [12] S. Feferman. A language and axioms for explicit mathematics. In *Algebra and Logic*, volume 450 of *Springer Lecture Notes in Mathematics*, pages 87–139. Springer Verlag, 1975.
- [13] S. Feferman. Constructive theories of functions and classes. In *Logic Colloquium '78*, pages 159–224. North-Holland, 1979.
- [14] S. Feferman. A theory of variable types. *Revista Colombiana de Matemáticas*, 19:95–105, 1985.
- [15] S. Feferman. Polymorphic typed lambda-calculi in a type-free axiomatic framework. In *Logic and Computation*, volume 106 of *Contemporary Mathematics*, pages 101–136. A.M.S., Providence R. I., 1990.
- [16] M. Felleisen. *The Calculi of Lambda- ν -cs Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages*. PhD thesis, Indiana University, 1987.
- [17] M. Felleisen, 1993. Personal communication.
- [18] M. Felleisen and D.P. Friedman. Control operators, the SECD-machine, and the λ -calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, pages 193–217. North-Holland, 1986.
- [19] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103:235–271, 1992.
- [20] J. Frost, 1995. Personal communication.
- [21] J. Frost and I.A. Mason. An Operational Logic of Effects. In *Proceedings of the Australasian Theory Symposium, CATS '96*, pages 147–156, 1996.
- [22] D. Harel. Dynamic logic. In D. Gabbay and G. Guenther, editors, *Handbook of Philosophical Logic, Vol. II*, pages 497–604. D. Reidel, 1984.
- [23] F. Honsell, I. A. Mason, S. F. Smith, and C. L. Talcott. A theory of classes for a functional language with effects. In *Proceedings of CSL92*, volume 702 of *Lecture Notes in Computer Science*, pages 309–326. Springer, Berlin, 1993.
- [24] F. Honsell, I. A. Mason, S. F. Smith, and C. L. Talcott. A Variable Typed Logic of Effects. *Information and Computation*, 119(1):55–90, May 1995.
- [25] D. Howe. Equality in the lazy lambda calculus. In *Fourth Annual Symposium on Logic in Computer Science*. IEEE, 1989.
- [26] T. Jim and A.R. Meyer. Full abstraction and the context lemma. In *Theoretical Aspects of Computer Science*, volume 526 of *Lecture Notes in Computer Science*, pages 131–151. Springer-Verlag, 1991.
- [27] P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6:308–320, 1964.
- [28] I. A. Mason. *The Semantics of Destructive Lisp*. PhD thesis, Stanford University, 1986. Also available as CSLI Lecture Notes No. 5, Center for the Study of Language and Information, Stanford University.

- [29] I. A. Mason and C. L. Talcott. Axiomatizing operational equivalence in the presence of side effects. In *Fourth Annual Symposium on Logic in Computer Science*. IEEE, 1989.
- [30] I. A. Mason and C. L. Talcott. A sound and complete axiomatization of operational equivalence between programs with memory. Technical Report STAN-CS-89-1250, Department of Computer Science, Stanford University, 1989.
- [31] I. A. Mason and C. L. Talcott. Reasoning about programs with effects. In *Programming Language Implementation and Logic Programming, PLILP'90*, volume 456 of *Lecture Notes in Computer Science*, pages 189–203. Springer-Verlag, 1990.
- [32] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.
- [33] I. A. Mason and C. L. Talcott. Inferring the equivalence of functional programs that mutate data. *Theoretical Computer Science*, 105(2):167–215, 1992.
- [34] I. A. Mason and C. L. Talcott. References, local variables and operational reasoning. In *Seventh Annual Symposium on Logic in Computer Science*, pages 186–197. IEEE, 1992.
- [35] I. A. Mason and C. L. Talcott. Program transformation via contextual assertions. In *Logic, Language and Computation. Festschrift in Honor of Satoru Takasu*, volume 792 of *Lecture Notes in Computer Science*, pages 225–254. Springer, Berlin, 1994.
- [36] I.A. Mason. Hoare's Logic in the LF. Technical Report ECS-LFCS-87-32, Laboratory for foundations of computer science, University of Edinburgh, 1987.
- [37] I.A. Mason and C.L. Talcott. Reasoning about Object Systems in VTLoE. *International Journal of Foundations on Computer Science*, 6(3):265–298, 1995.
- [38] R. Milner. Fully abstract models of typed λ -calculi. *Theoretical Computer Science*, 4:1–22, 1977.
- [39] E. Moggi. Computational lambda-calculus and monads. In *Fourth Annual Symposium on Logic in Computer Science*. IEEE, 1989.
- [40] J. H. Morris. *Lambda calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, 1968.
- [41] C. G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. Technical Report STAN-CS-77-647, Department of Computer Science, Stanford University, 1977.
- [42] C-H.L. Ong. *The Lazy Lambda Calculus: An investigation into the Foundations of Functional Programming*. PhD thesis, Imperial College, University of London, 1988.
- [43] Lawrence C. Paulson. *Isabelle, A Generic Theorem Prover*. Number 82 in *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.
- [44] A. M. Pitts. Evaluation logic. In *IVth Higher-Order Workshop, Banff*, volume 283 of *Workshops in Computing*. Springer-Verlag, 1990.
- [45] A. M. Pitts and I. Stark. On the observable properties of higher order functions that dynamically create local names. In *ACM Sigplan Workshop on State in Programming Languages*. YaleU/DCS/RR-968, 1993.
- [46] G. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.
- [47] D. Prawitz. *Natural Deduction: A Proof-theoretical Study*. Almqvist and Wiksell, 1965.

- [48] Michael O. Rabin. Decidability of second order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.
- [49] J.C. Reynolds. Idealized ALGOL and its specification logic. In D. Néel, editor, *Tools and Notions for Program Construction*, pages 121–161. Cambridge University Press, 1982.
- [50] S. F. Smith. From operational to denotational semantics. In *MFPS 1991*, volume 598 of *Lecture Notes in Computer Science*, pages 54–76. Springer-Verlag, 1992.
- [51] C. L. Talcott. *The essence of Rum: A theory of the intensional and extensional aspects of Lisp-type computation*. PhD thesis, Stanford University, 1985.
- [52] C. L. Talcott. Binding structures. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press, 1991.
- [53] C. L. Talcott. A theory for program and data type specification. *Theoretical Computer Science*, 104:129–159, 1992.
- [54] C. L. Talcott. A theory of binding structures and its applications to rewriting. *Theoretical Computer Science*, 112:99–143, 1993.
- [55] R. L. Tenney. *Decidable Pairing Functions*. PhD thesis, Cornell University, 1972. also appears as Computer Science TR 72-136.
- [56] R. W. Weyhrauch. Prolegomena to a theory of formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.