A Sound and Complete Axiomatization of Operational Equivalence between Programs with Memory [†]

Ian Mason
University of Edinburgh
IAM@SAIL.STANFORD.EDU

Carolyn Talcott
Stanford University
CLT@SAIL.STANFORD.EDU

1. Introduction

In this paper we present a formal system for deriving assertions about programs with memory. The assertions we consider are of the following three forms: (i) e diverges (i.e. fails to reduce to a value), written $\uparrow e$; (ii) e_0 and e_1 reduce to the same value and have exactly the same effect on memory, written $e_0 \sim e_1$; and (iii) e_0 and e_1 reduce to the same value and have the same effect on memory up to production of garbage (are strongly isomorphic), written $e_0 \simeq e_1$. The e, e_j are expressions of a first-order Scheme- or Lisp-like language with the data operations *atom*, *eq*, *car*, *cdr*, *cons*, *setcar*, *setcdr*, the control primitives `let` and `if`, and recursive definition of function symbols.

The formal system we present defines a single-conclusion consequence relation $\Sigma \vdash \Phi$ where Σ is a finite set of constraints and Φ is an assertion. A constraint set is a finite subset of atomic and negated atomic formulas in the first-order language consisting of equality, the unary function symbols *car* and *cdr*, the unary relation *atom*, and constants from the set of atoms, $\mathbb{A}$. Constraints have the natural first-order interpretation. The semantics of the formal system is given by a semantic consequence relation $\Sigma \models \Phi$ which is defined in terms of a class of memory models for assertions and constraints. The main results of this paper are

Theorem (Soundness): The deduction system is sound: $\Sigma \vdash \Phi \Rightarrow \Sigma \models \Phi$.

Theorem (Completeness): The deduction system is complete for Φ not containing recursively defined function symbols: $\Sigma \models \Phi \Rightarrow \Sigma \vdash \Phi$.

Operational equivalence [Morris 1969, Plotkin 1975] abstracts the operational semantics of programs and is the basis for soundness results for program calculi and program transformation theories. Two expressions are operationally equivalent if they are indistinguishable in all program contexts. The importance of the strong isomorphism relation is that strong isomorphism relative to the empty set of constraints is the same as operational equivalence. Thus the formal system can be used for proving operational equivalence, and is complete for expressions which do not contain recursively defined function symbols.

From the rules of the formal system and the proof of completeness we obtain a decision procedure for the semantic consequence relation. This is an important step towards developing computer-aided deduction tools for reasoning about programs with memory.

[Oppen 1978] gives a decision procedure for the first-order theory of pure Lisp, i.e. the theory of *atom*, *car*, *cdr*, *cons* over acyclic list structures. [Nelsen and Oppen 1977]

[†] This research was partially supported by DARPA contract N00039-84-C-0211

treats the quantifier-free case over possibly cyclic list structures. Neither treats updating operations. [Boehm 1985] defines a first-order theory for reasoning about programs in the language Russell which includes facilities for allocating and modifying memory. Program constructs are defined by two classes of axioms: (1) axioms about the value returned and (2) axioms giving the effect on memory. Some relative completeness results are given, but no decidable fragments are considered. Russell is strongly typed and hence prohibits many Lisp programs. The semantics of the full first-order Lisp-like language was studied in [Mason 1986, 1986a]. Here the model-theoretic equivalence strong isomorphism ($\simeq$) was introduced and used as the basis for studying program equivalence. Many examples of proving program equivalence can be found in [Mason and Talcott 1985, Mason 1986, 1988]. [Felleisen 1987] develops a calculus for reasoning about programs with memory, function abstractions and control abstractions. [Mason and Talcott 1989a] gives an alternative approach to treating programs with memory and function abstractions and develops the theory of operational equivalence for this case. More complete surveys of reasoning about programs with memory can be found in [Mason 1986, 1986a, 1988] and [Felleisen 1987, 1988].

The remainder of this paper is organized as follows. We first define our language and its operational semantics. We then present the axioms and rules of the formal system. Following that we define memory models and semantic consequence and prove the soundness theorem. Finally we prove the completeness theorem. To do this we develop a syntactic representation of the operational semantics which is also useful for reasoning about programs in general.

We conclude this section with a summary of notational conventions. We use the usual notation for set membership and function application. Let Y, Y_0, Y_1 be sets. 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. $[y_1, \ldots, y_n]$ is the sequence of length n with ith element y_i. $\mathbf{P}_\omega(Y)$ is the set of finite subsets of Y. $[Y_0 \to Y_1]$ is the set of functions f with domain Y_0 and range contained in Y_1. We write $\mathrm{Dom}(f)$ for the domain of a function and $\mathrm{Rng}(f)$ for its range. For any function f, $f\{y := y'\}$ is the function f' such that $\mathrm{Dom}(f') = \mathrm{Dom}(f) \cup \{y\}$, $f'(y) = y'$, and $f'(z) = f(z)$ for $z \neq y, z \in \mathrm{Dom}(f)$. $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the natural numbers and $i, j, n, n_0, \ \ldots$ range over $\mathbb{N}$.

2. The Operational Semantics

In existing applicative languages there are two approaches to introducing objects with memory. We shall call these the Lisp approach and the ML approach. In the Lisp approach the semantics of lambda abstraction is modified so that upon application lambda variables are bound to newly allocated memory cells. Reference to a variable returns the contents of the cell and there is an assignment operation (`setq` or `set!`) for updating the contents of the cell bound to a variable. With this modified semantics one can no longer use beta-conversion to reason about program equivalence. For example in the program $((\lambda x \ldots \mathtt{setq}(x, n + 1) \ldots)v)$ beta-conversion is not even meaningful, x cannot be substituted for by a value. Instead a cell must be allocated and x replaced by the cell name or labeled value. In the ML approach cells are added as a data type and operations are provided for creating cells and for accessing and modifying the contents. Reference to the contents of a cell must be made explicit. The semantics of lambda application is preserved and beta-value conversion remains a valid law for reasoning about programs. The Lisp approach provides a natural syntax since normally one wants to refer to the contents of a cell and not the cell itself. However the loss of the beta rule poses a serious problem for reasoning about programs. This approach also violates the principle of separating the mechanism for binding from that of allocation of memory [Mosses 1984]. Following the Scheme tradition, [Felleisen 1987] takes the Lisp approach to provide objects with memory. In order to obtain a reasonable calculus of programs, the programming language is extended to provide two sorts of lambda binding and an explicit dereferencing construct. There have been recent improvements in this calculus, but the problem of mixing binding and allocation is inherenent in the approach.

We take the ML approach to introducing objects with memory, adding primitive operations that create, access, and modify memory cells to the call-by-value lambda calculus. Since we are working in the first-order case, memories with cells that contain only a single atom or cell are not adequate for representing general list structures. Thus we treat memories with binary cells. In the higher-order case when cells can also contain function objects unary cells are sufficient. For brevity, we restrict our attention to expressions not containing recursively defined function symbols. The definitions and many of the intermediate results lift naturally to the full first-order language (see [Mason 1986]).

We fix a countably infinite set of atoms, $\mathbb{A}$, with two distinct elements playing the role of booleans, `T` for *true* and `Nil` for *false*. We also fix a countable set $\mathbb{X}$ of variables disjoint from $\mathbb{A}$.

Definition ($\mathbb{U}$,$\mathbb{E}$): The set of value expressions, $\mathbb{U}$, and the set of expressions, $\mathbb{E}$, are defined by

$$\mathbb{U} = \mathbb{X} \cup \mathbb{A}$$

$$\mathbb{E} = \mathbb{U} \cup \mathtt{let}\{\mathbb{X} := \mathbb{E}\}\mathbb{E} \cup \mathtt{if}(\mathbb{E}, \mathbb{E}, \mathbb{E}) \cup \mathbb{F}_1(\mathbb{E}) \cup \mathbb{F}_2(\mathbb{E}, \mathbb{E})$$

where the unary memory operations are $\mathbb{F}_1 = \{atom, car, cdr\}$ and the binary memory operations are $\mathbb{F}_2 = \{eq, cons, setcar, setcdr\}$. The equation for $\mathbb{E}$ is just compact notation for a standard inductive definition defining $\mathbb{E}$ to be the least set containing $\mathbb{U}$ and such that if $x \in \mathbb{X}$, $e_j \in \mathbb{E}$ for $j < 3$, $\delta_1 \in \mathbb{F}_1$, and $\delta_2 \in \mathbb{F}_2$ then $\mathtt{let}\{x := e_0\}e_1$, $\mathtt{if}(e_0, e_1, e_2)$, $\delta_1(e_1)$, and $\delta_2(e_1, e_2)$ are in $\mathbb{E}$. We let $a, a_0, \ldots$ range over $\mathbb{A}$, $x, x_0, y, z, \ldots$ range over $\mathbb{X}$, $u, u_0, \ldots$ range over $\mathbb{U}$, and $e, e_0, \ldots$ range over $\mathbb{E}$. The variable of a `let` is bound in the second expression, and the usual conventions concerning alpha conversion apply. We write $\mathrm{FV}(e)$ for the set of free variables of e. $\mathtt{seq}(e_0, \ldots, e_n)$ abbreviates $\mathtt{if}(e_0, \mathtt{seq}(e_1, \ldots, e_n), \mathtt{seq}(e_1, \ldots, e_n))$.

Expressions describe computations over S-expression memories — finite maps from (names of) cells to pairs of values, where a value is an atom or a cell. We call the value of a cell in a memory its contents. The memory operations are interpreted relative to a given memory as follows. *atom* is the characteristic function of the atoms, using the booleans `T` and `Nil`; *eq* tests whether two values are identical. *cons* takes two arguments, creates a new cell (extending the memory domain) whose contents is the pair of arguments, and returns the newly created cell. *car* and *cdr* return the first and second components of a cell. *setcar* and *setcdr* destructively alter an already existing cell. Given two arguments, the first of which must be a cell, *setcar* updates the given memory so that the first component of the contents of its first argument becomes its second argument. *setcdr* similarly alters the second component. Thus memories can be constructed in which one or both components of a cell can refer to the cell itself.

To define the operational semantics we fix a countable set of (names of) cells, $\mathbb{C}$, disjoint from $\mathbb{A}$ and $\mathbb{X}$. $\mathbb{V} = \mathbb{A} \cup \mathbb{C}$ is the collection of storable memory values. The set of memories, $\mathbb{M}$, consists of finite maps from cells to pairs of values. Cells which appear in the range of a memory are assumed to lie in its domain. For each $n \in \mathbb{N}$ we also define a collection of n-ary memory objects, $\mathbb{O}^{(n)} \subseteq \mathbb{V}^n \times \mathbb{M}$, (elements of $\mathbb{O}^{(1)}$ are called objects, and we omit the superscript). The cells in the n-tuple component of a memory object must lie in the domain of its memory component. The set of environments or bindings, $\mathbb{B}$, is the collection of finite functions from $\mathbb{X}$ to $\mathbb{V}$. The set of descriptions of computations, $\mathbb{D}$, is a subset of $\mathbb{E} \times \mathbb{B} \times \mathbb{M}$. In a description the free variables of the expression must be in the domain of the environment, and cells in the range of the environment must be in the domain of the memory. This is all summed up in the following definition.

Definition (Semantic Domains):

Memories: $\mathbb{M} = \{\mu \in [Z \to (Z \cup \mathbb{A})^2] \mid Z \in \mathbf{P}_\omega(\mathbb{C})\}$

Objects: $\mathbb{O}^{(n)} = \{[v_0, \ldots, v_{n-1}]; \mu \mid \mu \in \mathbb{M}, v_i \in \mathrm{Dom}(\mu) \cup \mathbb{A}, i \in n\}$

Bindings: $\mathbb{B} = \{\beta \in [X \to \mathbb{V}] \mid X \in \mathbf{P}_\omega(\mathbb{X})\}$

Descriptions: $\mathbb{D} = \{e; \beta; \mu \mid \mathrm{FV}(e) \subseteq \mathrm{Dom}(\beta), \mathrm{Rng}(\beta) \subseteq \mathrm{Dom}(\mu) \cup \mathbb{A}, \mu \in \mathbb{M}\}$

We let $c, c_0, \ldots$ range over $\mathbb{C}$, $v, v_0, \ldots$ range over $\mathbb{V}$, $\mu, \mu_0, \ldots$ range over $\mathbb{M}$, $u; \mu, u_0; \mu_0, \ldots$ range over $\mathbb{O}$, $\beta, \beta_0, \ldots$ range over $\mathbb{B}$, and $e; \beta; \mu, e_0; \beta_0; \mu_0, \ldots$ range over $\mathbb{D}$. We use ";" in some notations, for example objects and descriptions, since some components of the these tuples are also collections (sets or tuples) and we wish to emphasize the outer level tuple structure. We extend environments to value expressions by adopting the convention that $\beta(a) = a$ when $a \in \mathbb{A}$.

The operational semantics of expressions is given by a reduction relation $\overset{*}{\mapsto}$ on descriptions. It is generated in the following manner. $\overset{*}{\mapsto}$ is the reflexive transitive closure of the single-step relation $\mapsto$ which is defined in terms of reductions of *primitive expressions* and *reduction contexts*. The single-step reduction relation, $\mapsto$, is a subset of $(\mathbb{D} \times \mathbb{D})$, as is $\overset{*}{\mapsto}$. The action of the memory operations is given by the primitive reduction relation, $\rightharpoondown$, which is a subset of $(\mathbb{F}_1(\mathbb{O}) \times \mathbb{O}) \cup (\mathbb{F}_2(\mathbb{O}^{(2)}) \times \mathbb{O})$. Finally, the evaluation relation, $\hookrightarrow$, is a subset of $(\mathbb{D} \times \mathbb{O})$. Evaluation is reduction composed with the operation converting value descriptions $(u; \beta; \mu)$ into memory objects.

Computation is a process of applying reductions to descriptions. The reduction to apply is determined by the unique decomposition of a non-value expression into a *reduction context* filled by a *primitive expression*.

Definition ($\mathbb{E}_{\mathrm{prim}}$): The set of primitive expressions, $\mathbb{E}_{\mathrm{prim}}$, is defined as

$$\mathbb{E}_{\mathrm{prim}} = \mathtt{if}(\mathbb{U}, \mathbb{E}, \mathbb{E}) \cup \mathtt{let}\{\mathbb{X} := \mathbb{U}\}\mathbb{E} \cup \mathbb{F}_1(\mathbb{U}) \cup \mathbb{F}_2(\mathbb{U}, \mathbb{U})$$

Definition (${}^{\varepsilon}\mathbb{E}$): The set of contexts, ${}^{\varepsilon}\mathbb{E}$, is defined inductively using the special symbol ε for holes:

$${}^{\varepsilon}\mathbb{E} = \{\varepsilon\} \cup \mathbb{X} \cup \mathbb{A} \cup \mathtt{let}\{\mathbb{X} := {}^{\varepsilon}\mathbb{E}\}{}^{\varepsilon}\mathbb{E} \cup \mathtt{if}({}^{\varepsilon}\mathbb{E}, {}^{\varepsilon}\mathbb{E}, {}^{\varepsilon}\mathbb{E}) \cup \mathbb{F}_1({}^{\varepsilon}\mathbb{E}) \cup \mathbb{F}_2({}^{\varepsilon}\mathbb{E}, {}^{\varepsilon}\mathbb{E})$$

Definition ($\mathbb{R}$): The set of reduction contexts, $\mathbb{R}$, is the subset of ${}^{\varepsilon}\mathbb{E}$ defined by

$$\mathbb{R} = \{\varepsilon\} \cup \mathtt{let}\{\mathbb{X} := \mathbb{R}\}\mathbb{E} \cup \mathtt{if}(\mathbb{R}, \mathbb{E}, \mathbb{E}) \cup \mathbb{F}_1(\mathbb{R}) \cup \mathbb{F}_2(\mathbb{U}, \mathbb{R}) \cup \mathbb{F}_2(\mathbb{R}, \mathbb{E})$$

We let E, E' range over ${}^{\varepsilon}\mathbb{E}$ and R range over $\mathbb{R}$. $E[\![e]\!]$ denotes the result of replacing any holes in E by e. Free variables of e may become bound in this process. We often adopt the usual convention that $[\![\;]\!]$ denotes a hole. To avoid proliferation of brackets when dealing with composition of contexts we write $E; E'[\![e]\!]$ for $E[\![E'[\![e]\!]]\!]$ and similarly for longer composition chains.

Lemma (Decomposition): If $e \in \mathbb{E}$ then either $e \in \mathbb{U}$ or e can be written uniquely as $R[\![e']\!]$ where R is a reduction context and $e' \in \mathbb{E}_{\mathrm{prim}}$.

Definition ($\mapsto$): The single-step reduction relation $\mapsto$ on $\mathbb{D}$ is defined by

(beta) $\quad R[\![\mathtt{let}\{x := u\}e]\!]; \beta; \mu \mapsto R[\![e]\!]; \beta\{x := \beta(u)\}; \mu$

(if) $\quad R[\![\mathtt{if}(u, e_1, e_2)]\!]; \beta; \mu \mapsto \begin{cases} R[\![e_1]\!]; \beta; \mu & \text{if } \beta(u) \neq \mathtt{Nil} \\ R[\![e_2]\!]; \beta; \mu & \text{if } \beta(u) = \mathtt{Nil} \end{cases}$

(delta) $\quad R[\![\delta(u_1, \ldots, u_n)]\!]; \beta; \mu \mapsto R[\![x]\!]; \beta\{x := v'\}; \mu'$

where in the (**beta**) clause $x \notin \mathrm{Dom}(\beta)$ and in the (**delta**) clause $x \notin \mathrm{Dom}(\beta)$, $\delta \in \mathbb{F}_n$, $\delta([v_1, \ldots, v_n]; \mu) \rightharpoondown v'; \mu'$, and $v_i = \beta(u_i)$ for $1 \leq i \leq n$. The primitive reduction relation, $\rightharpoondown$, in (**delta**) is defined by cases according to the nature of $\delta \in \mathbb{F}_n$.

Definition ($\rightharpoondown$): The primitive reduction relation $\delta([v_0, \ldots, v_{n-1}]; \mu) \rightharpoondown v'; \mu'$ is the least relation satisfying the following conditions.

$$atom(v; \mu) \rightharpoondown \begin{cases} \mathtt{T}; \mu & \text{if } v \in \mathbb{A} \\ \mathtt{Nil}; \mu & \text{otherwise} \end{cases}$$

$$car(c; \mu) \rightharpoondown v_0; \mu$$

$$cdr(c; \mu) \rightharpoondown v_1; \mu$$

$$eq([v_0, v_1]; \mu) \rightharpoondown \begin{cases} \mathtt{T}; \mu & \text{if } v_0 = v_1 \\ \mathtt{Nil}; \mu & \text{otherwise} \end{cases}$$

$$cons([v_0, v_1]; \mu) \rightharpoondown c; \mu\{c := [v_0, v_1]\} \qquad \text{for any } c \text{ such that } c \notin \mathrm{Dom}(\mu)$$

$$setcar([c, v]; \mu) \rightharpoondown c; \mu\{c := [v, v_1]\}$$

$$setcdr([c, v]; \mu) \rightharpoondown c; \mu\{c := [v_0, v]\}$$

where in the cases for *car*, *cdr*, *setcar* and *setcdr* we assume that $c \in \mathrm{Dom}(\mu)$ and $\mu(c) = [v_0, v_1]$.

Although formally *cons* is multi-valued, the values differ only by renaming of cells and generally we will not distinguish them. Defining *cons* as a relation rather than a function which makes an arbitrary choice is the semantic analog of alpha conversion and greatly simplifies many definitions and proofs. If μ is a memory and $\vartheta \in \{car, cdr\}$, then the function $\vartheta_\mu \in [\mathrm{Dom}(\mu) \to \mathbb{V}]$, is defined by

$$car_\mu(c) = v \Leftrightarrow (\exists v')(\mu(c) = [v, v']) \quad \text{and} \quad cdr_\mu(c) = v \Leftrightarrow (\exists v')(\mu(c) = [v', v])$$

Definition ($\hookrightarrow$,$\downarrow$,$\uparrow$): A description $e;\beta;\mu \in \mathbb{D}$ evaluates to the object $v;\mu' \in \mathbb{O}$, if it reduces to a value description denoting that object.

$$e;\beta;\mu \hookrightarrow v;\mu' \;\Leftrightarrow\; (\exists u;\beta';\mu')(e;\beta;\mu \overset{*}{\mapsto} u;\beta';\mu' \,\wedge\, \beta'(u) = v)$$

As for primitive reductions, single-step reduction and evaluation are single-valued relations modulo renaming of cells. A description is defined, written $\downarrow e;\beta;\mu$, just if $(\exists v;\mu' \in \mathbb{O})(e;\beta;\mu \hookrightarrow v;\mu')$. A description is undefined, written $\uparrow e;\beta;\mu$, just if $\neg(\downarrow e;\beta;\mu)$. We identify a closed expression with the description consisting of it, the empty environment and the empty memory. Thus $\uparrow e$ abbreviates $\uparrow e;\emptyset;\emptyset$. In the absence of recursively defined functions reduction sequences are finite. In this case a description is undefined only if reduction terminates in attempting to access or update the contents of an atom. In the full first-order case undefinedness also includes divergence.

We define operational equivalence following [Plotkin 1975].

Definition ($\cong$): Two expressions are said to be operationally equivalent, written $e_0 \cong e_1$, if and only if for any closing context E, $E[\![e_0]\!]$ and $E[\![e_1]\!]$ are either both defined or both undefined.

$$(\forall E \in {}^{\varepsilon}\mathbb{E} \mid \mathrm{FV}(E[\![e_0]\!]) = \mathrm{FV}(E[\![e_1]\!]) = \emptyset)((\bigwedge_{i<2} \uparrow E[\![e_i]\!]) \vee (\bigwedge_{i<2} \downarrow E[\![e_i]\!]))$$

By definition operational equivalence is a congruence relation on expressions. However it is not necessarily the case that instantiations of equivalent expressions are equivalent even if the instantiation is defined. Note that `T` and `Nil` are not operationally equivalent. More generally, define two closed expressions to be trivially equivalent if both are undefined, both return the same atom or both return cells, then two expressions are operationally equivalent just if they are trivially equivalent in all closing contexts. This is the usual characterization of operational equivalence in the presence of basic data. Both definitions are equivalent in this setting since equality on basic data is computable. These observations are summarized in the following lemma.

Lemma (Congruence):

1. $e_0 \cong e_1 \;\Leftrightarrow\; (\forall E \in {}^{\varepsilon}\mathbb{E})(E[\![e_0]\!] \cong E[\![e_1]\!])$
2. It is not the case that $\downarrow e \wedge e_0 \cong e_1$ implies $e_0\{x := e\} \cong e_1\{x := e\}$ for arbitrary variable x and expressions e, e_0, e_1.
3. $\neg(\mathtt{T} \cong \mathtt{Nil})$
4. $e_0 \cong e_1 \Leftrightarrow (\forall E \in {}^{\varepsilon}\mathbb{E} \mid \mathrm{FV}(E[\![e_0]\!]) = \mathrm{FV}(E[\![e_1]\!]) = \emptyset)(E[\![e_0]\!] \cong^0 E[\![e_1]\!])$ where for closed expressions e_0', e_1' $e_0' \cong^0 e_1'$ iff

$$(\bigwedge_{i<2} \uparrow e_i') \vee (\exists v_0;\mu_0, v_1;\mu_1)((\bigwedge_{i<2} e_i' \hookrightarrow v_i;\mu_i) \wedge ((v_0 = v_1 \wedge v_0, v_1 \in \mathbb{A}) \vee (v_0, v_1 \in \mathbb{C})))$$

Proof (congruence):

(1) For the if direction take $E = [\![\;]\!]$. For the other direction note that for any E if E' is any closing context for $E[\![e_j]\!]$ for $j \in 2$ then $E'[\![E]\!]$ is a closing context for e_j for $j \in 2$.

(2) As a counter-example we have $eq(x, x) \cong \mathtt{T}$ but $eq(cons(\mathtt{T}, \mathtt{T}), cons(\mathtt{T}, \mathtt{T})) \cong \mathtt{Nil}$.

(3) The context $\mathtt{if}(\varepsilon, car(\mathtt{T}), \mathtt{T})$ will distinguish $\mathtt{T}$ and $\mathtt{Nil}$.

(4) The if direction is trivial. For the other direction suppose $(\bigwedge_{i<2} E[\![e_j]\!] \hookrightarrow v_j; \mu_j)$. If $v_0, v_1 \in \mathbb{A}$ and $v_0 \neq v_1$ then the context $\mathtt{if}(eq(v_0, E), car(\mathtt{T}), \mathtt{T})$ will distinguish the expressions. Similarly, if $v_0 \in \mathbb{A}$ and $v_1 \in \mathbb{C}$ then the context $\mathtt{if}(atom(E), car(\mathtt{T}), \mathtt{T})$ will distinguish the expressions.

$\square_{\mathbf{congruence}}$

3. The Formal System

In this section we present the language and rules of our formal system. The assertion language $\mathbb{L}$ and the constraint language $\mathcal{L}$ are defined as follows:

Definition ($\mathbb{L}$)**:**

$$\mathbb{L} = (\mathbb{E} \simeq \mathbb{E}) \cup (\mathbb{E} \sim \mathbb{E}) \cup (\uparrow \mathbb{E})$$

Definition ($\mathcal{L}$)**:**

$$\mathcal{L} = (car(\mathbb{U}) = \mathbb{U}) \cup (cdr(\mathbb{U}) = \mathbb{U}) \cup (\mathbb{U} = \mathbb{U}) \cup \neg(\mathbb{U} = \mathbb{U}) \cup (atom(\mathbb{U})) \cup \neg(atom(\mathbb{U}))$$

We let $\varphi, \ldots$ range over $\mathcal{L}$ $\Phi, \ldots$ range over $\mathbb{L}$, and $\Sigma, \Sigma_0, \Delta, \ldots$ range over $\mathbf{P}_\omega(\mathcal{L})$.

The set of constraints $\mathcal{L}$ is a subset of the atomic and negated atomic formulas in the first-order language consisting of equality, the unary function symbols *car* and *cdr*, the unary relation *atom*, and constants from $\mathbb{A}$. We will freely use standard notions such as first-order satisfaction, $\models$.

Definition ($\mathrm{Th}(\mathbb{A})$)**:** The theory $\mathrm{Th}(\mathbb{A})$ is defined by

$$\mathrm{Th}(\mathbb{A}) = \{atom(a), \neg(a = a') \;\big|\; a, a' \in \mathbb{A}, a \neq a'\}$$

To state the rules, as well as the side conditions on rules, we use the following notation. The result of pushing a context E through an assertion Φ is defined by

$$E[\![\Phi]\!] = \begin{cases} \uparrow E[\![e]\!] & \text{if } \Phi = \uparrow e \\ E[\![e_0]\!] \sim E[\![e_1]\!] & \text{if } \Phi = e_0 \sim e_1 \\ E[\![e_0]\!] \simeq E[\![e_1]\!] & \text{if } \Phi = e_0 \simeq e_1. \end{cases}$$

For $\vartheta \in \{car, cdr\}$, x is ϑ-less in Σ just if $\neg(\exists u \in \mathbb{U})(\Sigma \models \vartheta(x) = u)$ and $(\forall y \in \mathbb{X})((\vartheta(y) = u) \in \Sigma \Rightarrow \Sigma \models \neg(x = y))$. If x is ϑ-less in Σ then the only way to consistently add information concerning $\vartheta(x)$ is by adding an assertion of the form $\vartheta(x) = u$. Furthermore, if x is ϑ-less in Σ then we can add $\vartheta(x) = u$ to Σ without changing equality consequences of Σ. $\mathrm{Dom}(\Sigma)$ is the subset of $\mathrm{FV}(\Sigma)$ defined by $\mathrm{Dom}(\Sigma) = \{x \in \mathrm{FV}(\Sigma) \;\big|\; \Sigma \models \neg atom(x)\}$. If $x \in \mathrm{Dom}(\Sigma)$ then x must be interpreted as a cell. For each constraint $\varphi \in \mathcal{L}$ there is a corresponding assertion $T(\varphi) \in \mathbb{L}$ defined by

$$T(\varphi) = \begin{cases} \vartheta(x) \sim u & \text{if } \varphi \text{ is } \vartheta(x) = u \text{ and } \vartheta \in \{car, cdr\} \\ eq(u_0, u_1) \sim \mathtt{T} & \text{if } \varphi \text{ is } u_0 = u_1 \\ eq(u_0, u_1) \sim \mathtt{Nil} & \text{if } \varphi \text{ is } \neg(u_0 = u_1) \\ atom(x) \sim \mathtt{T} & \text{if } \varphi \text{ is } atom(x) \\ atom(x) \sim \mathtt{Nil} & \text{if } \varphi \text{ is } \neg atom(x). \end{cases}$$

3.1. The Rules

Definition ($\Sigma \vdash \Phi$): The consequence relation, $\vdash$, is the smallest relation that is closed under the rules given below.

The rules are partitioned into several groups of related rules. Each group of rules is given a label for future reference and members of the group are numbered. For example (**S.i**) refers to the first rule in the group of structural rules (the first 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. A pair of rules that differ by interchanging premiss and conclusion is presented as a single rule with a double bar.

Most of the rules are solely concerned with $\sim$. Of the rest those that concern all assertions are (**S.ii, L, R.i**). The structural rule (**S.ii**) is used to put constraint sets into a form necessary for application of another rule — for example (**set.vii**). It also incorporates trivial facts concerning equality and the nature of the atoms. The left elimination rules, (**L.i, L.ii**), enable one to reason by cases while (**L.iii**) allows one to eliminate vacuous constraints. (**R.i**) states the key property of reduction contexts.

The rules concerning divergence are (**D, cons.iii, set.vii**). The latter two concern both $\uparrow$ and $\sim$ and allow memory descriptions encoded in a constraint set to be represented syntactically. The rules concerning $\simeq$ are (**E.i,ii,iii,G**). The first three simply assert that it is an equivalence relation weaker than $\sim$. The garbage collection rule allows for the elimination of garbage — *cons* cells no longer accessible.

Structural rules (S).

(i) $\Sigma \cup \{\varphi\} \vdash T(\varphi)$

(ii) $$\frac{\Sigma \cup \{\varphi\} \vdash \Phi}{\Sigma \vdash \Phi} \ \text{(double bar)}$$

where in (ii) $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models \varphi$.

Left elimination (L).

(i) $$\frac{\Sigma \cup \{atom(x)\} \vdash \Phi \qquad \Sigma \cup \{\neg atom(x)\} \vdash \Phi}{\Sigma \vdash \Phi}$$

(ii) $$\frac{\Sigma \cup \{\neg(u_0 = u_1)\} \vdash \Phi \qquad \Sigma \cup \{u_0 = u_1\} \vdash \Phi}{\Sigma \vdash \Phi}$$

(iii) $$\frac{\Sigma \cup \{\vartheta(x) = z\} \vdash \Phi}{\Sigma \vdash \Phi}$$

where in (iii) $\vartheta \in \{car, cdr\}$, $x \in \mathrm{Dom}(\Sigma)$, and $z \notin \mathrm{FV}(\Phi) \cup \mathrm{FV}(\Sigma)$.

Equivalence rules (E).

(i) $$\frac{\Sigma \vdash e_0 \sim e_1}{\Sigma \vdash e_0 \simeq e_1}$$ (ii) $$\frac{\Sigma \vdash e_0 \simeq e_1 \qquad \Sigma \vdash e_1 \simeq e_2}{\Sigma \vdash e_0 \simeq e_2}$$ (iii) $$\frac{\Sigma \vdash e_0 \simeq e_1}{\Sigma \vdash e_1 \simeq e_0}$$

(iv) $\Sigma \vdash e_0 \sim e_0$ (v) $$\frac{\Sigma \vdash e_0 \sim e_1 \qquad \Sigma \vdash e_1 \sim e_2}{\Sigma \vdash e_0 \sim e_2}$$ (vi) $$\frac{\Sigma \vdash e_0 \sim e_1}{\Sigma \vdash e_1 \sim e_0}$$

Rule concerning eq (eq).

$$\text{(i)}\quad \frac{\Sigma \vdash eq(x,y) \sim \mathtt{T}}{\Sigma \vdash x \sim y}$$

Divergence rules (D).

$$\text{(i)}\quad \frac{\Sigma \vdash \uparrow e_0 \qquad \Sigma \vdash \uparrow e_1}{\Sigma \vdash e_0 \sim e_1} \qquad \text{(ii)}\quad \frac{\Sigma \vdash \uparrow e_0 \qquad \Sigma \vdash e_0 \sim e_1}{\Sigma \vdash \uparrow e_1}$$

$$\text{(iii)}\quad \frac{\Sigma \vdash atom(x) \sim \mathtt{T}}{\Sigma \vdash \uparrow \vartheta(x)} \qquad \text{(iv)}\quad \frac{\Sigma \vdash atom(x) \sim \mathtt{T}}{\Sigma \vdash \uparrow set\vartheta(x,y)}$$

where in (iii,iv) $\vartheta \in \{car, cdr\}$

Reduction context rules (R).

$$\text{(i)}\quad \frac{\Sigma \vdash \Phi}{\Sigma \vdash R[\![\Phi]\!]}$$

$$\text{(ii)}\quad \Sigma \vdash R[\![\mathtt{if}(e_0, e_1, e_2)]\!] \sim \mathtt{if}(e_0, R[\![e_1]\!], R[\![e_2]\!])$$

$$\text{(iii)}\quad \Sigma \vdash R[\![\mathtt{let}\{x := e_0\}e_1]\!] \sim \mathtt{let}\{x := e_0\}R[\![e_1]\!]$$

where in (iii) $x \notin \mathrm{FV}(R)$.

Rules concerning `let` (let).

$$\text{(i)}\quad \Sigma \vdash e \sim \mathtt{let}\{x := e\}x \qquad \text{(ii)}\quad \Sigma \vdash e\{x := u\} \sim \mathtt{let}\{x := u\}e$$

Rules concerning `if` (if).

$$\text{(i)}\quad \Sigma \vdash \mathtt{seq}(e_0, e_1) \sim \mathtt{let}\{x := e_0\}e_1$$

$$\text{(ii)}\quad \Sigma \vdash \mathtt{if}(\mathtt{Nil}, e_0, e_1) \sim e_1 \qquad \text{(iii)}\quad \frac{\Sigma \vdash eq(u, \mathtt{Nil}) \sim \mathtt{Nil}}{\Sigma \vdash \mathtt{if}(u, e_0, e_1) \sim e_0}$$

where in (i) $x \notin \mathrm{FV}(e_1)$.

Rules for $cons$ (cons).

$$\text{(i)}\quad \Sigma \vdash \mathtt{let}\{x_0 := cons(\mathtt{T},\mathtt{T})\}\mathtt{let}\{x_1 := cons(\mathtt{T},\mathtt{T})\}e$$
$$\sim \mathtt{let}\{x_1 := cons(\mathtt{T},\mathtt{T})\}\mathtt{let}\{x_0 := cons(\mathtt{T},\mathtt{T})\}e$$

$$\text{(ii)}\quad \Sigma \vdash \mathtt{seq}(e_0, \mathtt{let}\{x := cons(u_0, u_1)\}e_1) \sim \mathtt{let}\{x := cons(u_0, u_1)\}\mathtt{seq}(e_0, e_1)$$

$$\text{(iii)}\quad \frac{\Sigma \cup \Delta \vdash \Phi}{\Sigma \vdash \mathtt{let}\{x := cons(u_a, u_d)\}[\![\Phi]\!]}$$

where in (ii) $x \notin \mathrm{FV}(e_0)$, and in (iii) $\Phi \in (\mathbb{E} \sim \mathbb{E}) \cup (\uparrow \mathbb{E})$, $x \notin (\mathrm{FV}(\Sigma) \cup \{u_a, u_d\}) = Z$ and

$$\Delta = \{\neg atom(x), car(x) = u_a, cdr(x) = u_d, \neg(x = y) \;\big|\; y \in Z \cup \mathrm{FV}(\Phi) - \{x\}\}$$

Rules for *setcar* **and** *setcdr* **(set).**

(i) $$\frac{\Sigma \vdash eq(x_0, x_2) \sim \mathtt{Nil}}{\Sigma \vdash \mathtt{seq}(set\vartheta(x_0, x_1), set\vartheta(x_2, x_3), e) \sim \mathtt{seq}(set\vartheta(x_2, x_3), set\vartheta(x_0, x_1), e)}$$

(ii) $\Sigma \vdash \mathtt{seq}(set\vartheta(x, y_0), set\vartheta(x, y_1)) \sim set\vartheta(x, y_1)$

(iii) $\Sigma \vdash \mathtt{seq}(set\vartheta(x, y), x) \sim set\vartheta(x, y)$

(iv) $\Sigma \vdash \mathtt{seq}(setcdr(x_0, x_1), setcar(x_2, x_3), e) \sim \mathtt{seq}(setcar(x_2, x_3), setcdr(x_0, x_1), e)$

(v) $\Sigma \vdash setcar(cons(z, y), x) \sim cons(x, y)$

(vi) $\Sigma \vdash setcdr(cons(x, z), y) \sim cons(x, y)$

(vii) $$\frac{\Sigma \cup \{\vartheta(x) = u_0\} \vdash \Phi}{\overline{\Sigma \cup \{\vartheta(x) = u_1\} \vdash \mathtt{seq}(set\vartheta(x, u_0), [\![\Phi]\!])}}$$

where $\vartheta \in \{car, cdr\}$ and in (vii) $\Phi \in (\mathbb{E} \sim \mathbb{E}) \cup (\uparrow \mathbb{E})$, $x \in \mathrm{Dom}(\Sigma)$, and x is ϑ-less in Σ.

Garbage collection rule (G). If Γ is a context of the form

$$\begin{aligned}&\mathtt{let}\{z_1 := cons(\mathtt{T}, \mathtt{T})\} \ldots \mathtt{let}\{z_n := cons(\mathtt{T}, \mathtt{T}))\}\\ &\quad \mathtt{seq}(setcar(z_1, u_1^a), setcdr(z_1, u_1^d), \ldots, setcar(z_n, u_n^a), setcdr(z_n, u_n^d), \varepsilon).\end{aligned}$$

and $\{z_1, \ldots, z_n\} \cap \mathrm{FV}(e) = \emptyset$, then

$\Sigma \vdash \Gamma[\![e]\!] \simeq e.$

3.2. Consequences

The following are some consequences of the rules defining $\Sigma \vdash \Phi$.

Lemma (Equiv): If $\Sigma \cup \mathrm{Th}(\mathbb{A})$ and $\Sigma' \cup \mathrm{Th}(\mathbb{A})$ have the same first-order consequences then $\Sigma \vdash \Phi \Leftrightarrow \Sigma' \vdash \Phi$.

Proof (Equiv): By (**S.ii**). $\square_{\mathbf{Equiv}}$

Lemma (Mon): If $\Sigma \vdash \Phi$ then $\Sigma \cup \Sigma' \vdash \Phi$.

Proof (Mon): By induction on the length of proof and cases on the last rule applied. We consider the two most interesting cases.

(cons.iii) Assume that $x \notin \mathrm{FV}(\Sigma')$ and that we have derived

$$\Sigma \vdash \mathtt{let}\{x := cons(u_a, u_d)\}[\![\Phi]\!]$$

where the last rule applied is (**cons.iii**). Then by induction hypothesis $\Sigma \cup \Sigma' \cup \Delta' \vdash \Phi$ where $\Delta' = \Delta \cup \{\neg(x = y) \mid y \in \mathrm{FV}(\Sigma')\}$. Hence by (**cons.iii**) we are done. $\square_{\mathbf{cons.iii}}$

(set.vii) Assume we have derived $\Sigma \cup \{\vartheta(x) = u_1\} \vdash \mathtt{seq}(set\vartheta(x, u_0), [\![\Phi]\!])$ and the last rule applied is (**set.vii**) in the forward direction. Thus $x \in \mathrm{Dom}(\Sigma)$, x is ϑ-less in Σ, and

$\Sigma \cup \{\vartheta(x) = u_0\} \vdash \Phi$. By (**L**) we may assume that for $\vartheta(z) = u \in \Sigma'$ we have either $\Sigma \cup \Sigma' \models z = x$ or $\Sigma \cup \Sigma' \models \neg(z = x)$. Let

$$\Sigma_x = \{u = u_1 \mid (\exists z)(\vartheta(z) = u \in \Sigma' \wedge \Sigma \cup \Sigma' \models z = x)\}$$
$$\Sigma_0 = \Sigma' - \{\vartheta(z) = u \in \Sigma' \mid \Sigma \cup \Sigma' \models z = x\}.$$

Then

$$\Sigma \cup \Sigma_0 \cup \Sigma_x \cup \{\vartheta(x) = u_0\} \vdash \Phi \qquad \text{by induction hypothesis}$$
$$\Sigma \cup \Sigma_0 \cup \Sigma_x \cup \{\vartheta(x) = u_1\} \vdash \mathtt{seq}(set\vartheta(x, u_0), \Phi) \qquad \text{by } (\textbf{set.vii})$$
$$\Sigma \cup \Sigma' \cup \{\vartheta(x) = u_1\} \vdash \mathtt{seq}(set\vartheta(x, u_0), \Phi) \qquad \text{by } (\textbf{S}).$$

The proof for application of (**set.vii**) in the reverse direction is similar. $\square_{\textbf{set.vii}}$

$\square_{\textbf{Mon}}$

Lemma (Equality):

$$\Sigma \models x = y \wedge \Sigma \vdash \Phi \Rightarrow \Sigma \vdash \Phi\{x := y\}$$

Proof (Equality): By induction on the length of proof. Again we consider only the interesting cases.

(S.i) If $\Sigma \models x = y$ then $\Sigma \cup \{\varphi\} \models \varphi\{x := y\}$.

(S.ii) If $\Sigma \models \varphi$ then $\Sigma \cup \{\varphi\} \models x = y \Leftrightarrow \Sigma \models x = y$.

(set.vii) Note that if $\Sigma \cup \{\vartheta(z) = u\} \models x = y$ and Σ is ϑ-less for z then $\Sigma \models x = y$ and use (**Equiv**) in the case $z \in \{x, y\}$.

$\square_{\textbf{Equality}}$

Lemma (Examples):

(i) $\{\neg atom(x)\} \vdash setcar(x, car(x)) \sim x$

(ii) $\Sigma \vdash cdr(cons(x, y)) \simeq y$

(iii) $\Sigma \vdash \mathtt{seq}(\mathtt{seq}(e_0, e_1), e_2) \sim \mathtt{seq}(e_0, \mathtt{seq}(e_1, e_2))$

(iv) $\Sigma \vdash eq(x, y) \sim eq(y, x)$

(v) $\Sigma \vdash eq(x, x) \sim \mathtt{T}$

(vi) $\Sigma \vdash atom(a) \sim \mathtt{T}$

(vii) $\dfrac{\Sigma \vdash a_0 \neq a_1}{\Sigma \vdash eq(a_0, a_1) \sim \mathtt{Nil}}$

(viii) $\dfrac{\Sigma \vdash atom(x) \sim \mathtt{Nil} \quad \Sigma \vdash atom(y) \sim \mathtt{T}}{\Sigma \vdash eq(x, y) \sim \mathtt{Nil}}$

(ix) $\Sigma \vdash \mathtt{let}\{x_0 := cons(u_0^a, u_0^d)\}\mathtt{let}\{x_1 := cons(u_1^a, u_1^d)\}e$

$\sim \mathtt{let}\{x_1 := cons(u_1^a, u_1^d)\}\mathtt{let}\{x_0 := cons(u_0^a, u_0^d)\}e$

provided $\{x_0, x_1\} \cap \mathrm{FV}(u_0^a, u_0^d, u_1^a, u_1^d) = \emptyset$

(x) $\Sigma \vdash \mathtt{let}\{y := e_0\}\mathtt{let}\{x := e_1\}e_2 \sim \mathtt{let}\{x := \mathtt{let}\{y := e_0\}e_1\}e_2 \quad \text{if} \quad y \notin \mathrm{FV}(e_2)$

Proof (Examples):

(i) Let $\Sigma' = \{\neg atom(x), car(x) = y\}$ then by (**set.ii,iii**) we have

$$\Sigma' \vdash \mathtt{seq}(setcar(x,y), setcar(x,y)) \sim \mathtt{seq}(setcar(x,y), x)$$

and by (**set.vii**) we have $\Sigma' \vdash setcar(x,y) \sim x$. Now, using (**L,S,E,R**) we obtain $\Sigma \vdash setcar(x, car(x)) \sim x$.

(ii) $\Sigma \vdash \mathtt{let}\{z := cons(x,y)\}cdr(z) \sim \mathtt{let}\{z := cons(x,y)\}y$ by (**S.i, cons.iii**). Thus $\Sigma \vdash cdr(cons(x,y)) \sim \mathtt{let}\{z := cons(x,y)\}y$ by (**R.iii,let.i**). The result now follows by (**E,G**) and the simple exercise showing that

$$\mathtt{let}\{z := cons(x,y)\}y \sim \mathtt{let}\{z := cons(\mathtt{T},\mathtt{T})\}\mathtt{seq}(setcar(z,x), setcdr(z,y), y).$$

(iii) By (**R.ii**) and the definition of $\mathtt{seq}$.

(iv) By (**L.ii,S,E**).

(v,vi,vii) By (**S**).

(viii) To show that $\Sigma \vdash eq(x,y) \sim \mathtt{Nil}$ it suffices by (**L.ii, S.i**) to show that $\Sigma \cup \{x = y\} \vdash eq(x,y) \sim \mathtt{Nil}$. By (**Equality, E, S.i**) and the assumptions we have that $\Sigma \cup \{x = y\} \vdash \mathtt{T} \sim \mathtt{Nil}$. The result now follows by (**S.i, E**).

(ix) This is left to the reader. A similar derivation can be found in the proof of completeness.

(x) This is an instance of (**R.iii**).

$\square_{\textbf{Examples}}$

4. Soundness

In this section we present the semantics of our formal system. We begin by defining what it means for a model to satisfy an assertion or a constraint set. The semantic consequence relation between constraint sets and assertions is defined naturally in terms of these satisfaction relations.

Definition (model): A model is an environment-memory pair such that cells in the range of the environment are in the domain of the memory. We let $\beta;\mu$, $\beta_0;\mu_0$, ...range over models.

Definition ($\sim$): Two descriptions with the same model are defined to be equi-valued, written $e_0;\beta;\mu \sim e_1;\beta;\mu$, if both diverge or both evaluate to the same object:

1. $\uparrow e_1;\beta;\mu$ and $\uparrow e_2;\beta;\mu$, or
2. $(\exists v;\mu')(e_1;\beta;\mu \hookrightarrow v;\mu' \wedge e_2;\beta;\mu \hookrightarrow v;\mu')$

Definition ($\simeq$): Two descriptions with the same model are strongly isomorphic, written $e_0;\beta;\mu \simeq e_1;\beta;\mu$, if both diverge or both evaluate to the same object up to production of cells not accessible from the value:

1. $\uparrow e_1;\beta;\mu$ and $\uparrow e_2;\beta;\mu$, or

2. $(\exists v; \mu' \in \mathbb{O} \mid \mathrm{Dom}(\mu) \subseteq \mathrm{Dom}(\mu'))(\bigwedge_{i<2}(\exists \mu_i \mid \mu' \subseteq \mu_i)(e_i; \beta; \mu \hookrightarrow v; \mu_i))$

The model-theoretic equivalence strong isomorphism ($\simeq$) was introduced in [Mason 1986] and used as the basis for studying program equivalence. The relation between strong isomorphism and operational equivalence is given by the following theorem. [This theorem holds for the full first-order language, not just the fragment with no recursively defined functions.]

Theorem (Strong Isomorphism): If $e_0, e_1 \in \mathbb{E}$, then $e_0 \cong e_1$ if and only if for every $\beta; \mu$ such that $\mathrm{FV}(e_0, e_1) \subseteq \mathrm{Dom}(\beta)$ we have that $e_0; \beta; \mu \simeq e_1; \beta; \mu$.

Proof (Strong Isomorphism): The key idea is to show that if there is a context that distinguishes two expressions then there is a simple memory context, see section 5 for the definition, that distinguishes them. See [Mason 1986] for further analysis of strong isomorphism. $\square_{\mathbf{StrongIsomorphism}}$

Definition ($\models_{\mathbb{L}}$): The notion of a model satisfying an assertion, $\beta; \mu \models_{\mathbb{L}} \Phi$, is defined for $\mathrm{FV}(\Phi) \subseteq \mathrm{Dom}(\beta)$ by

$$\beta; \mu \models_{\mathbb{L}} \Phi \;\Leftrightarrow\; \begin{cases} \uparrow e; \beta; \mu & \text{if } \Phi = \uparrow e \\ e_0; \beta; \mu \sim e_1; \beta; \mu & \text{if } \Phi = e_0 \sim e_1 \\ e_0; \beta; \mu \simeq e_1; \beta; \mu & \text{if } \Phi = e_0 \simeq e_1. \end{cases}$$

The notion of a model satisfying a set of constraints $\beta; \mu \models_{\mathcal{L}} \Sigma$ is simply first-order satisfaction adapted to the memory structure framework. For any memory μ we define the corresponding first-order structure $\mathcal{M}_\mu$ by

$$\mathcal{M}_\mu = \langle \mathrm{Dom}(\mu) \cup \mathbb{A}, \mathit{car}_\mu, \mathit{cdr}_\mu, \mathit{atom} \rangle$$

where $\mathrm{Dom}(\mu) \cup \mathbb{A}$ is the domain of $\mathcal{M}_\mu$, $\mathit{car}_\mu, \mathit{cdr}_\mu$ are treated as binary relations, and *atom* is a unary relation. For $\beta \in \mathbb{B}$, $\varphi \in \mathcal{L}$ such that $\mathrm{FV}(\varphi) \subseteq \mathrm{Dom}(\beta)$ and $\mathrm{Rng}(\beta) \subseteq \mathrm{Dom}(\mu) \cup \mathbb{A}$ we write $\mathcal{M}_\mu \models \varphi[\beta]$ for the usual first-order satisfaction relation where $\varphi[\beta]$ is the interpretation of φ relative to the environment β, thought of as a Tarskian assignment. Thus

$$\mathcal{M}_\mu \models \varphi[\beta] \;\Leftrightarrow\; \begin{cases} \beta(x) \in \mathbb{A} & \text{if } \varphi \text{ is } \mathit{atom}(x) \\ \beta(x) \in \mathrm{Dom}(\mu) & \text{if } \varphi \text{ is } \neg\mathit{atom}(x) \\ \beta(u_0) = \beta(u_1) & \text{if } \varphi \text{ is } u_0 = u_1 \\ \beta(u_0) \neq \beta(u_1) & \text{if } \varphi \text{ is } \neg(u_0 = u_1) \\ \vartheta_\mu(\beta(x)) = \beta(u) & \text{if } \varphi \text{ is } \vartheta(x) = u \text{ and } \vartheta \in \{\mathit{car}, \mathit{cdr}\} \end{cases}$$

Definition ($\models_{\mathcal{L}}$): $\beta; \mu \models_{\mathcal{L}} \Sigma$ if there is a $\beta' \supseteq \beta$ with $\mathrm{FV}(\Sigma) \subseteq \mathrm{Dom}(\beta')$ and $\mathrm{Rng}(\beta') \subseteq \mathbb{A} \cup \mathrm{Dom}(\mu)$ such that $\mathcal{M}_\mu \models \varphi[\beta']$ for $\varphi \in \Sigma$.

Definition ($\Sigma \models \Phi$): The semantic consequence relation $\Sigma \models \Phi$ is defined by

$$\Sigma \models \Phi \;\Leftrightarrow\; (\forall \beta; \mu \mid \mathrm{FV}(\Phi) \subseteq \mathrm{Dom}(\beta))(\beta; \mu \models_{\mathcal{L}} \Sigma \Rightarrow \beta; \mu \models_{\mathbb{L}} \Phi).$$

A constraint set Σ is consistent just if $\beta; \mu \models_{\mathcal{L}} \Sigma$ for some model $\beta; \mu$. In order to make explicit the consequences of the above definition of satisfaction, we state the following definition and lemma.

Definition (Σ^m): The memory structure theory, Σ^m, corresponding to Σ is defined by

$$\Sigma^m = \Sigma \cup \mathrm{Th}(\mathbb{A}) \cup \{\neg atom(x) \bigm| (\exists u \in \mathbb{U})((car(x) = u) \in \Sigma \vee (cdr(x) = u) \in \Sigma)\}$$

Lemma (Sat): For $\varphi \in \mathcal{L}$ we have $\Sigma^m \models \varphi \Leftrightarrow \Sigma \models T(\varphi)$.

Proof (Sat): This is an easy consequence of properties of first-order satisfaction and the fact that if φ has a model then it has a model with the same $\mathcal{L}$ consequences that corresponds to a memory structure. $\square_{\mathbf{Sat}}$

Theorem (Soundness): If $\Sigma \vdash \Phi$ then $\Sigma \models \Phi$.

Proof (Soundness): It suffices to show that each rule preserves soundness, i.e. soundness of the premisses implies soundness of the conclusion. We restrict our attention to those rules for which this result is non-trivial. The proofs for the remaining rules are either trivial or else minor variations on the ones given.

Lemma (S): $\Sigma \cup \{\varphi\} \models T(\varphi)$ for $\varphi \in \mathcal{L}$.

Proof (S): Suppose $\beta; \mu \models_{\mathcal{L}} \Sigma \cup \{\varphi\}$ and without loss of generality that $\mathrm{FV}(\Sigma \cup \{\varphi\}) \subseteq \mathrm{Dom}(\beta)$. Then by definition $\beta; \mu \models_{\mathcal{L}} \varphi$. This together with the definition of T is sufficient to force that $\beta; \mu \models_{\mathbb{L}} T(\varphi)$. $\square_{\mathbf{S}}$

Lemma (L): Suppose that $\vartheta \in \{car, cdr\}$, $x \in \mathrm{Dom}(\Sigma)$, and $z \notin \mathrm{FV}(\Phi) \cup \mathrm{FV}(\Sigma)$. Then

$$\frac{\Sigma \cup \{\vartheta(x) = z\} \models \Phi}{\Sigma \models \Phi}$$

Proof (L): Suppose that $\neg(\Sigma \models \Phi)$. Then without loss of generality we may assume that there is a $\beta; \mu$ such that $\mathrm{Dom}(\beta) = \mathrm{FV}(\Sigma) \cup \mathrm{FV}(\Phi)$ with $\beta; \mu \models_{\mathcal{L}} \Sigma$ and $\neg(\beta; \mu \models_{\mathbb{L}} \Phi)$. Since $z \notin \mathrm{FV}(\Sigma) \cup \mathrm{FV}(\Phi)$ we have that $\beta\{z := \vartheta_\mu(\beta(x))\}; \mu \models_{\mathcal{L}} \Sigma \cup \{\vartheta(x) = z\}$ and $\neg(\beta\{z := \vartheta_\mu(\beta(x))\}; \mu \models_{\mathbb{L}} \Phi)$. Thus $\neg(\Sigma \cup \{\vartheta(x) = z\} \models \Phi)$. $\square_{\mathbf{L}}$

Lemma (cons): Suppose that $\Phi \in (\mathbb{E} \sim \mathbb{E}) \cup (\uparrow \mathbb{E})$, $x \notin (\mathrm{FV}(\Sigma) \cup \{u_a, u_d\}) = Z$ and

$$\Delta = \{\neg atom(x), car(x) = u_a, cdr(x) = u_d, \neg(x = y) \bigm| y \in Z \cup (\mathrm{FV}(\Phi) - \{x\})\}.$$

Then

$$\frac{\Sigma \cup \Delta \models \Phi}{\Sigma \models \mathtt{let}\{x := cons(u_a, u_d)\}[\![\Phi]\!]}$$

Proof (cons): Suppose that $\mathrm{FV}(\Sigma) \subseteq \beta$, $x \notin \mathrm{Dom}(\beta)$ and that $\beta; \mu \models_{\mathcal{L}} \Sigma$. Furthermore assume that $\neg(\beta; \mu \models \mathtt{let}\{x := cons(u_a, u_d)\}[\![\Phi]\!])$. Thus choosing $c \notin \mathrm{Dom}(\mu)$ and letting $\beta'; \mu' = \beta\{x := c\}; \mu\{c := [\beta(u_a), \beta(u_d)]\}$ we have that $\neg(\beta'; \mu' \models \Phi)$. Consequently it suffices to show that $\beta'; \mu' \models_{\mathcal{L}} \Sigma \cup \Delta$. This is routine. $\square_{\mathbf{cons}}$

Lemma (set): Suppose that $\Phi \in (\mathbb{E} \sim \mathbb{E}) \cup (\uparrow \mathbb{E})$, $x \in \mathrm{Dom}(\Sigma)$ and x is cdr-less in Σ. Then

$$\frac{\Sigma \cup \{cdr(x) = u_0\} \models \Phi}{\Sigma \cup \{cdr(x) = u_1\} \models \mathtt{seq}(setcdr(x, u_0), [\![\Phi]\!])}$$

Proof (set): Pick $\beta;\mu$ such that $\beta;\mu \models_{\mathcal{L}} \Sigma$, $\mathrm{FV}(\Phi) \cup \mathrm{FV}(\Sigma) \cup \{x, u_i\} \subseteq \mathrm{Dom}(\beta)$, $\beta(x) = c$ and for $i < 2$ put

$$\Sigma_i = \Sigma \cup \{cdr(x) = u_i\}$$
$$\mu_i = \mu\{c := [car_\mu(c), \beta(u_i)]\}$$
$$\Phi_0 = \Phi$$
$$\Phi_1 = \mathtt{seq}(setcdr(x, u_0), [\![\Phi]\!]).$$

Furthermore, without loss of generality, assume that $\mathrm{FV}(\Sigma_i) \subseteq \mathrm{Dom}(\beta)$. We show that $\beta;\mu_0 \models_{\mathcal{L}} \Sigma_0$ iff $\beta;\mu_1 \models_{\mathcal{L}} \Sigma_1$. The result then follows by observing that $\beta;\mu_0 \models \Phi_0$ iff $\beta_1;\mu_1 \models \Phi_1$. Clearly $\beta;\mu_i \models_{\mathcal{L}} \{cdr(x) = u_i\}$ since by construction $cdr_{\mu_i}(c) = \beta(u_i)$. Thus it suffices to show that for any $\varphi \in \Sigma$, $\mathcal{M}_{\mu_0} \models \varphi[\beta] \;\Leftrightarrow\; \mathcal{M}_{\mu_1} \models \varphi[\beta]$. This is trivially true if φ is of the form $atom(y), \neg atom(y), u_0 = u_1, \neg(u_0 = u_1)$ or $car(y) = u$, so suppose that $(cdr(y) = u) \in \Sigma$. Since x is cdr-less in Σ we have that $\Sigma \models \neg(x = y)$, consequently $cdr_{\mu_0}(\beta(y)) = cdr_{\mu_1}(\beta(y))$. Thus $\mathcal{M}_{\mu_0} \models (cdr(y) = u)[\beta]$ iff $\mathcal{M}_{\mu_1} \models (cdr(y) = u)[\beta]$. $\square_{\mathbf{set}}$

$\square_{\mathbf{Soundness}}$

5. Completeness

In this section we state and prove the completeness theorem.

Theorem (Completeness): $\Sigma \models \Phi$ implies $\Sigma \vdash \Phi$.

The proof of the completeness theorem is essentially an elaborate normal form theorem. Suppose $\Sigma \models e_0 \simeq e_1$. We define two forms of contexts which feature in the normal form proof: syntactic memory contexts, Γ, and modifications, M. Using these contexts we define, relative to Σ, a form of syntactic reduction, $\overset{*}{\mapsto}_\Sigma$. It is defined in such a way that

$$(e \overset{*}{\mapsto}_\Sigma e') \;\Rightarrow\; (\Sigma \vdash e \simeq e')$$

and if Σ contains enough information concerning the nature of the free variables of e_i, then

$$e_i \overset{*}{\mapsto}_\Sigma \Gamma_i; M_i; e_i',$$

and either $e_i' \in \{R[\![\vartheta(u_i)]\!], R[\![set\vartheta(u_i, u_i')]\!]\}$, $\vartheta \in \{car, cdr\}$ and $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u_i)$ or else $e_i' = u_i$. In the latter case the normal form of e_i is then defined simply to be $\Gamma_i; M_i; u_i$. We show that one can use the introduction on the left rules to force Σ to contain the necessary information. Consequently suppose, for arguments sake, that Σ does contain sufficient information and that the normal form of e_i is $\Gamma_i; M_i; u_i$. Then we have $\Sigma \vdash e_i \simeq \Gamma_i; M_i; u_i$. Thus by soundness $\Sigma \models e_i \simeq \Gamma_i; M_i; u_i$. Consequently $\Sigma \models \Gamma_0; M_0; u_0 \simeq \Gamma_1; M_1; u_1$. The completeness result then follows by showing that equivalent normal forms are provably equivalent.

To obtain additional insight, consider the semantic question of deciding for any Σ and Φ whether $\Sigma \models \Phi$. Since all computations terminate we can decide for any $\beta;\mu$ such that $\mathrm{FV}(\Phi) \subseteq \mathrm{Dom}(\beta)$ whether $\beta;\mu \models \Phi$. We say Σ is complete for Φ if Σ determines the structure of its models upto depth the size of Φ. If $\mathrm{FV}(e) \subseteq \mathrm{Dom}(\beta)$, the size of e is $\leq n$, and $\beta;\mu_0$ and $\beta;\mu_1$ are the same to depth n — agree on cells reachable from $\mathrm{Rng}(\beta)$ by paths of length $\leq n$ — then $e;\beta;\mu_0$ and $e;\beta;\mu_1$ have the same computation sequences. Thus if Σ is complete for Φ, to decide $\Sigma \models \Phi$ we need only pick some $\beta;\mu$ such that $\beta;\mu \models \Sigma$ and $\mathrm{FV}(\Phi) \subseteq \mathrm{Dom}(\beta)$ and check whether $\beta;\mu \models \Phi$ (For consistent Σ it is easy to find such models). Finally we note that for any Σ, Φ we can find a finite set of constraints $\{\Sigma_i \bigm| i < N\}$ such that

- for $i < N$, Σ_i is complete for Φ,
- for $i < N$, any model of Σ_i is a model of Σ, and
- any model of Σ is a model of Σ_i for some (unique) $i < N$.

Thus $\Sigma \models \Phi \Leftrightarrow (\forall i < N)\Sigma_i \models \Phi$ and we have seen how to decide the righthand side of the equivalence.

The completeness proof parallels the decidability argument using syntactic representations of memories and reduction. We begin by developing these representations. We then present the key lemmas for the proof of completeness and the proof itself. Finally we prove the lemmas.

5.1. Memory contexts and Modifications

Definition (Memory contexts): The syntactic analog of a memory is a memory context, Γ, which is a context of the form

$$\begin{array}{l}\mathtt{let}\{z_1 := \mathit{cons}(\mathtt{T},\mathtt{T})\}\ldots\mathtt{let}\{z_n := \mathit{cons}(\mathtt{T},\mathtt{T})\}\\ \quad\mathtt{seq}(\mathit{setcar}(z_1,u_1^a), \mathit{setcdr}(z_1,u_1^d), \ldots, \mathit{setcar}(z_n,u_n^a), \mathit{setcdr}(z_n,u_n^d), \varepsilon).\end{array}$$

where $z_i \neq z_j$ when $i \neq j$. In analogy to the semantic memories, we define the domain of Γ to be $\mathrm{Dom}(\Gamma) = \{z_1,\ldots,z_n\}$. For Γ as above we define the functions $\mathit{car}_\Gamma, \mathit{cdr}_\Gamma \in [\mathrm{Dom}(\Gamma) \to \mathbb{U}]$ by $\mathit{car}_\Gamma(z_i) = u_i^a$ and $\mathit{cdr}_\Gamma(z_i) = u_i^d$. Two memory contexts are considered the same if they have the same domain and contents. Thus a memory context is determined by its domain and selector functions. We also define extension and updating operations on memory contexts. $\Gamma\{z := [u_{car}, u_{cdr}]\}$ is defined for $z \notin \mathrm{Dom}(\Gamma)$ to be the memory context Γ' such that $\mathrm{Dom}(\Gamma') = \mathrm{Dom}(\Gamma) \cup \{z\}$ and for $\vartheta \in \{\mathit{car}, \mathit{cdr}\}$,

$$\vartheta_{\Gamma'}(z') = \begin{cases} u_\vartheta & \text{if } z' = z \\ \vartheta_\Gamma(z') & \text{otherwise.}\end{cases}$$

$\Gamma\{\mathit{car}(z) = u\}$ is defined for $z \in \mathrm{Dom}(\Gamma)$ to be the memory context Γ' such that $\mathrm{Dom}(\Gamma') = \mathrm{Dom}(\Gamma)$ and

$$\mathit{car}_{\Gamma'}(z') = \begin{cases} u & \text{if } z' = z \\ \mathit{car}_\Gamma(z') & \text{otherwise}\end{cases} \quad\text{and}\quad \mathit{cdr}_{\Gamma'}(z') = \mathit{cdr}_\Gamma(z').$$

Similarly for $\Gamma\{\mathit{cdr}(z) = u\}$.

To express the constraints implicit in a memory context Γ we define for any Σ the extension of Σ by Γ relative to a given set of variables X.

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

$$\Sigma_\Gamma^X = \Sigma \cup \Delta_{ad} \cup \Delta_{ne}$$

$$\Delta_{ad} = \bigcup_{z \in \mathrm{Dom}(\Gamma)} \{\neg atom(z), \vartheta(z) = u_\vartheta \mid u_\vartheta = \vartheta_\Gamma(z), \vartheta \in \{car, cdr\}\}$$

$$\Delta_{ne} = \bigcup_{z \in \mathrm{Dom}(\Gamma)} \{\neg(z = y) \mid y \in \mathrm{FV}(\Sigma) \cup X \cup (\mathrm{Dom}(\Gamma) - \{z\})\}.$$

It is natural and convenient to extend memory contexts by sequences of assignments to variables that are not in the domain of the memory context, but are assumed to be cells. We call such extensions modifications.

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

$$\mathtt{seq}(set\vartheta_1(z_1, u_1), \ldots, set\vartheta_n(z_n, u_n), \varepsilon)$$

where $set\vartheta_i \in \{setcar, setcdr\}$ and $z_i = z_j$ implies $i = j$ or $set\vartheta_i \neq set\vartheta_j$. We define $\mathrm{Dom}(M) = \{z_1, \ldots, z_n\}$ and $\vartheta_M(z_i) = u_i$ if $set\vartheta_i = set\vartheta$ for $\vartheta \in \{car, cdr\}$. Thus $\mathrm{Dom}(\vartheta_M) = \{z_i \in \mathrm{Dom}(M) \mid set\vartheta_i = set\vartheta\}$ for $\vartheta \in \{car, cdr\}$.

5.2. Σ-Reduction

In analogy to the semantic reduction relations we define the relations $\rightharpoondown_\Sigma$, $\mapsto_\Sigma$, and $\overset{*}{\mapsto}_\Sigma$. In order to ensure that definitions are meaningful we introduce the notion of coherence. Roughly a constraint set and a memory-modification context are coherent (written $Coh(\Sigma, \Gamma; M)$) if $\mathrm{Dom}(\Gamma) \cap \mathrm{FV}(\Sigma) = \emptyset$, modifications in M are to elements of $\mathrm{Dom}(\Sigma)$, Σ decides equality on $\mathrm{Dom}(\Sigma)$, distinct elements of $\mathrm{Dom}(M)$ are provably distinct in Σ and Σ contains at most one *car* or *cdr* assertion for any z in $\mathrm{Dom}(\Sigma)$. (The last condition is a technicality to make various definitions and proofs simpler.) Note that coherence ensures that ϑ_M is single–valued modulo Σ equivalence.

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

(1) $\mathrm{Dom}(\Gamma) \cap \mathrm{FV}(\Sigma) = \emptyset$

(2) $\mathrm{Dom}(M) \subseteq \mathrm{Dom}(\Sigma)$

(3) If $x_0, x_1 \in \mathrm{Dom}(\vartheta_M)$ are distinct, then $\Sigma \models \neg(x_0 = x_1)$ for $\vartheta \in \{car, cdr\}$.

(4) If $x_0, x_1 \in \mathrm{Dom}(\Sigma)$ then $\Sigma \models (x_0 = x_1)$ or $\Sigma \models \neg(x_0 = x_1)$.

(5) If $x \in \mathrm{Dom}(\Sigma)$ and $\vartheta \in \{car, cdr\}$, then there is at most one formula $(\vartheta(z) = u) \in \Sigma$ with $\Sigma \models (z = x)$.

We write $Coh(\Sigma, M)$ for $Coh(\Sigma, \Gamma; M)$ when $\mathrm{Dom}(\Gamma) = \emptyset$; $Coh(\Sigma, \Gamma)$ for $Coh(\Sigma, \Gamma; M)$ when $\mathrm{Dom}(M) = \emptyset$. and $Coh(\Sigma)$ for $Coh(\Sigma, \Gamma; M)$ when $\mathrm{Dom}(\Gamma) = \mathrm{Dom}(M) = \emptyset$.

Definition ($M\{\vartheta(z) = u\}_\Sigma$): Suppose that M is a modification, $Coh(\Sigma, M)$ and $z \in \mathrm{Dom}(\Sigma)$. Then $M\{car(z) = u\}_\Sigma$ is defined to be the modification M' with $\mathrm{Dom}(car_{M'}) = \mathrm{Dom}(car_M) \cup \{z\}$, $\mathrm{Dom}(cdr_{M'}) = \mathrm{Dom}(cdr_M)$, and for $z' \in \mathrm{Dom}(\vartheta_{M'})$

$$car_{M'}(z') = \begin{cases} car_M(z') & \text{if } \Sigma \models \neg(z = z') \\ u & \text{if } \Sigma \models (z = z') \end{cases} \quad \text{and} \quad cdr_{M'}(z') = cdr_M(z').$$

Similarly for $M\{cdr(z) = u\}_\Sigma$.

Definition ($\rightharpoondown_\Sigma$): For Σ and $\Gamma; M$ such that $Coh(\Sigma, \Gamma; M)$ we define the relation $\Gamma; M[\![e]\!] \rightharpoondown_\Sigma \Gamma'; M'[\![e']\!]$ as follows (letting $X = \mathrm{FV}(\Gamma; M[\![e]\!])$ and $\vartheta \in \{car, cdr\}$)

$$\Gamma; M[\![atom(u)]\!] \rightharpoondown_\Sigma \begin{cases} \Gamma; M[\![\mathtt{T}]\!] & \text{if } \Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u) \\ \Gamma; M[\![\mathtt{Nil}]\!] & \text{if } \Sigma_\Gamma^X \models \neg atom(u) \end{cases}$$

$$\Gamma; M[\![\vartheta(u)]\!] \rightharpoondown_\Sigma \begin{cases} \Gamma; M[\![\vartheta_\Gamma(u)]\!] & \text{if } u \in \mathrm{Dom}(\Gamma) \\ \Gamma; M[\![\vartheta_M(u)]\!] & \text{if } (\exists u' \in \mathrm{Dom}(\vartheta_M))(\Sigma \models (u' = u)) \\ \Gamma; M[\![u']\!] & \text{otherwise if } u \in \mathrm{Dom}(\Sigma) \wedge \Sigma \models (\vartheta(u) = u') \end{cases}$$

$$\Gamma; M[\![eq(u_0, u_1)]\!] \rightharpoondown_\Sigma \begin{cases} \Gamma; M[\![\mathtt{T}]\!] & \text{if } \Sigma \cup \mathrm{Th}(\mathbb{A}) \models u_0 = u_1 \\ \Gamma; M[\![\mathtt{Nil}]\!] & \text{if } \Sigma_\Gamma^X \cup \mathrm{Th}(\mathbb{A}) \models \neg(u_0 = u_1) \end{cases}$$

$$\Gamma; M[\![cons(u_0, u_1)]\!] \rightharpoondown_\Sigma \Gamma\{z := [u_0, u_1]\}; M[\![z]\!] \qquad \text{if } z \in \mathbb{X} - (\mathrm{Dom}(\Gamma) \cup \mathrm{FV}(\Sigma) \cup X)$$

$$\Gamma; M[\![set\vartheta(u, u')]\!] \rightharpoondown_\Sigma \begin{cases} \Gamma\{\vartheta(u) = u'\}; M[\![u]\!] & \text{if } u \in \mathrm{Dom}(\Gamma) \\ \Gamma; M\{\vartheta(u) = u'\}_\Sigma[\![u]\!] & \text{if } u \in \mathrm{Dom}(\Sigma) \end{cases}$$

For general use in reasoning about programs one would want to strengthen the definition of syntactic reduction by using full semantic satisfaction rather than first–order satisfaction in the side conditions. The weaker definition is adequate for proving completeness and simplifies the proof.

Definition ($\mapsto_\Sigma$): For Σ and $\Gamma; M$ such that $Coh(\Sigma, \Gamma; M)$ we define the relation $\Gamma; M; R[\![e]\!] \mapsto_\Sigma \Gamma'; M'; R[\![e']\!]$ as follows. Let $X = \mathrm{FV}(\Gamma; M; R[\![e]\!])$. Then

(if) $\quad \Gamma; M; R[\![\mathtt{if}(u, e_1, e_2)]\!] \mapsto_\Sigma \begin{cases} \Gamma; M; R[\![e_1]\!] & \text{if } \Sigma_\Gamma^X \cup \mathrm{Th}(\mathbb{A}) \models \neg(u = \mathtt{Nil}) \\ \Gamma; M; R[\![e_2]\!] & \text{if } \Sigma \models (u = \mathtt{Nil}) \end{cases}$

(beta) $\quad \Gamma; M; R[\![\mathtt{let}\{x := u\}e]\!] \mapsto_\Sigma \Gamma; M; R[\![e\{x := u\}]\!]$

(delta) $\quad \Gamma; M; R[\![\delta(u_1, \ldots, u_n)]\!] \mapsto_\Sigma \Gamma'; M'; R[\![u']\!]$

where in (**delta**) we assume that $\delta \in \mathbb{F}_n$, $\Gamma; M[\![\delta(u_1, \ldots, u_n)]\!] \rightharpoondown_\Sigma \Gamma'; M'; u'$ and $\mathrm{Dom}(\Gamma') - \mathrm{Dom}(\Gamma)$ is disjoint from $\mathrm{FV}(\Gamma; M; R[\![\delta(u_1, \ldots, u_n)]\!])$.

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

If a modification M is coherent with a constraint set Σ then the modification of Σ implicit in M is carried out explicitly in defining Σ_M.

Definition (Σ_M): For $Coh(\Sigma, M)$ we define Σ_M as follows

$$\Sigma_M = (\Sigma - \Delta_M^{forget}) \cup \Delta_M^{assign}$$

$$\Delta_M^{assign} = \{\vartheta(z) = u_\vartheta \mid u_\vartheta = \vartheta_M(z), z \in \mathrm{Dom}(\vartheta_M), \vartheta \in \{car, cdr\}\}$$

$$\Delta_M^{forget} = \{(\vartheta(x) = u) \in \Sigma \mid (\exists z \in \mathrm{Dom}(\vartheta_M))(\Sigma \models x = z), \vartheta \in \{car, cdr\}\}$$

The Context Modification Introduction lemma combines and generalizes the *cons* and $set\vartheta$ introduction rules to arbitrary memory–modification contexts.

Lemma (CMI): If $Coh(\Sigma, \Gamma; M)$, $\Phi \in (\mathbb{E} \sim \mathbb{E}) \cup (\uparrow \mathbb{E})$, and $X = \mathrm{FV}(\Gamma; M; R[\![\Phi]\!])$ then

$$\frac{(\Sigma_\Gamma^X)_M \vdash \Phi}{\Sigma \vdash \Gamma; M; R[\![\Phi]\!]}$$

is derivable.

Proof (CMI): This is a simple consequence of the introduction rules (**cons.iii**) and (**set.vii**), together with the congruence rules and the definition of coherence (particularly the fifth condition). The only point to observe is that if Σ is the disjoint union of Σ' and $\{car(z_i) = w_i^a, cdr(z_i) = w_i^d\}_{z_i \in \mathrm{Dom}(M)}$, then each z_i is *car*-less and *cdr*-less in Σ'. $\square_{\mathbf{CMI}}$

5.3. Proof of Completeness

Before we state the key lemmas, we require one last set of definitions. The rank of an expression $r(e)$ is just its size. The rank of an assertion $r(\Phi)$ is the maximum rank of the expressions occurring in Φ. $\mathrm{At}(X)$ is the set of atoms occuring in X. A *car-cdr* chain of length $\leq n$ is a reduction context of the form $\Theta = \vartheta_0(\vartheta_1(\ldots\vartheta_k(\varepsilon)\ldots))$ where $\vartheta_j \in \{car, cdr\}, j \leq k$, and $k < n$. Finally we define the notion of n-completeness for constraint sets relative to a finite set of variables and atoms. 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. $[\bar{x}, A]$): Σ is n-complete w.r.t. $[\bar{x}, A]$ if for every Θ, Θ_0, *car-cdr* chains of length $\leq n$, and $y, y_0 \in \bar{x}$, if $\Sigma \models \Theta[\![y]\!] = u$ and $\Sigma \models \Theta_0[\![y_0]\!] = u_0$, then

$$(\Sigma \models atom(u)) \vee (\Sigma \models \neg atom(u))$$

$$(\Sigma \models u = \alpha) \vee (\Sigma \models \neg(u = \alpha)) \quad \alpha \in A \cup \{\mathtt{T}, \mathtt{Nil}, u_0\}$$

$$(\Sigma \models \neg atom(u)) \Rightarrow (\exists u_a, u_d \in \mathbb{U})((\Sigma \models car(u) = u_a) \wedge (\Sigma \models cdr(u) = u_d))$$

$$(\Sigma \models atom(u)) \Rightarrow \neg(\exists u_a, u_d \in \mathbb{U})((\Sigma \models car(u) = u_a) \vee (\Sigma \models cdr(u) = u_d))$$

The following five lemmas enable a straightforward proof of the completeness theorem.

Lemma (0): If Σ is inconsistent, then $\Sigma \vdash \Phi$, for any $\Phi \in \mathbb{L}$.

Lemma (1): If $e \stackrel{*}{\mapsto}_\Sigma e'$, then $\Sigma \vdash e \sim e'$.

Lemma (2): If Σ is $r(e)$-complete w.r.t. $[\mathrm{FV}(e), \mathrm{At}(\Sigma, e)]$ and $Coh(\Sigma)$, then there exists $\Gamma; M$ and an e' such that $e \stackrel{*}{\mapsto}_\Sigma \Gamma; M[\![e']\!]$ and exactly one of the following holds:

1. $e' = R[\![\vartheta(u)]\!], \vartheta \in \{car, cdr\}$ and $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u)$.
2. $e' = R[\![set\vartheta(u_0, u_1)]\!], set\vartheta \in \{setcar, setcdr\}$ and $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u_0)$.
3. $e' = u$, and $Coh(\Sigma, \Gamma; M)$.

Lemma (3): For any consistent $\Sigma, \bar{x}, \Phi \in \mathbb{L}$, and $n \in \mathbb{N}$ there exists $N \in \mathbb{N}$ and a family of constraint sets $\{\Sigma_i\}_{i<N}$ such that

1. Each Σ_i is n-complete w.r.t. $[\bar{x}, \mathrm{At}(\Sigma_i, \Phi)]$, and $Coh(\Sigma_i)$.

2. $(\forall\beta;\mu)(\beta;\mu \models_{\mathcal{L}} \Sigma \Leftrightarrow (\exists i < N)(\beta;\mu \models_{\mathcal{L}} \Sigma_i))$

3. $\dfrac{\Sigma_i \vdash \Phi \quad i < N}{\Sigma \vdash \Phi}$ is a derived rule.

Lemma (4): Let $e_i = \Gamma_i; M_i[\![u_i]\!]$ with $Coh(\Sigma, \Gamma_i; M_i)$ for $i < 2$. If $\Sigma \models e_0 \sim e_1$ then $\Sigma \vdash e_0 \sim e_1$. Similarly if $\Sigma \models e_0 \simeq e_1$ then $\Sigma \vdash e_0 \simeq e_1$.

Proof (Completeness): Assume $\Sigma \models \Phi$. By lemma 0 we may assume that Σ is consistent. By lemma 3 it suffices to prove that $\Sigma \vdash \Phi$ under the added assumptions that $Coh(\Sigma)$ and Σ is $r(\Phi)$-complete w.r.t. $[\mathrm{FV}(\Phi), \mathrm{At}(\Sigma, \Phi)]$. By lemma 2 we have that for each e_i in Φ there exists $\Gamma_i; M_i$ and an e'_i such that $e_i \overset{*}{\mapsto}_\Sigma \Gamma_i; M_i[\![e'_i]\!]$ and exactly one of the following holds:

1. $e'_i = R_i[\![\vartheta_i(u_i)]\!]$, $\vartheta_i \in \{car, cdr\}$, and $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u_i)$.
2. $e'_i = R_i[\![set\vartheta_i(u_i, u'_i)]\!]$, $set\vartheta_i \in \{setcar, setcdr\}$ and $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u_i)$.
3. $e'_i = u_i$, and $Coh(\Sigma, \Gamma_i; M_i)$.

By lemma 1 we have $\Sigma \vdash e_i \sim \Gamma_i; M_i[\![e'_i]\!]$ and by soundness we have $\Sigma \models e_i \sim \Gamma_i; M_i[\![e']\!]$. We consider three cases, depending on the nature of Φ.

$(\Phi = \uparrow e)$ Since Σ is consistent $e' \in \mathbb{U}$ is impossible.

In the other two cases we use (**D**) and (**CMI**) to show that $\Sigma \vdash \uparrow \Gamma; M[\![e']\!]$, and hence that $\Sigma \vdash \uparrow e$.

$(\Phi = (e_0 \sim e_1))$ We may assume that $\neg(\Sigma \models \uparrow e_i)$ since the case when $\Sigma \models \uparrow e_i$ follows directly from the previous case. Hence we have $\Sigma \vdash e_i \sim \Gamma_i; M_i[\![u_i]\!]$ and $\Sigma \models e_i \sim \Gamma_i; M_i[\![u_i]\!]$ for $i < 2$. Thus $\Sigma \models \Gamma_0; M_0[\![u_0]\!] \sim \Gamma_1; M_1[\![u_1]\!]$ and by lemma 4 $\Sigma \vdash \Gamma_0; M_0[\![u_0]\!] \sim \Gamma_1; M_1[\![u_1]\!]$.

$(\Phi = (e_0 \simeq e_1))$ similar.

$\square_{\text{Completeness}}$

5.4. Proofs of the Lemmas

Lemma (0): If Σ is inconsistent, then $\Sigma \vdash \Phi$, for any $\Phi \in \mathbb{L}$.

Proof (0): If Σ is inconsistent then by (**Sat**) either $\Sigma \models \mathtt{T} = \mathtt{Nil}$ in the usual first-order interpretation, or else $\Sigma \models atom(x)$ and $\Sigma \models \vartheta(x) = z$ for some $x, z \in \mathbb{U}$. In the former case the result follows by the structural rules and properties of `if`. In the later case it suffices to observe that $\Sigma \vdash \uparrow z$ and so since $\Sigma \vdash e\{y := z\} \sim \mathtt{let}\{y := z\}e$ we can conclude, by choosing y new, that $\Sigma \vdash \uparrow e$ for any e. The result follows without much effort. $\square_0$

Lemma (1): If $e \overset{*}{\mapsto}_\Sigma e'$, then $\Sigma \vdash e \sim e'$.

Proof (1): It suffices to show that if $Coh(\Sigma, \Gamma; M)$, then

$$\Gamma; M[\![e]\!] \mapsto_\Sigma \Gamma'; M'[\![e']\!] \;\Rightarrow\; (\Sigma \vdash \Gamma; M[\![e]\!] \sim \Gamma'; M'[\![e']\!]).$$

Let $X = \mathrm{FV}(M[\![e]\!])$, $\Sigma' = (\Sigma^X_\Gamma)_M$, and note that the proof naturally divides up into three cases depending on the decomposition of e into $R[\![e_p]\!]$.

(if) In this case $e = R[\![\mathtt{if}(u, e_1, e_2)]\!]$ and by hypothesis either $\Sigma \models (u = \mathtt{Nil})$ or $\Sigma_\Gamma^X \cup \mathrm{Th}(\mathbb{A}) \models \neg(u = \mathtt{Nil})$. Thus either $\Sigma' \vdash u \sim \mathtt{Nil}$ or $\Sigma' \vdash eq(u, \mathtt{Nil}) \sim \mathtt{Nil}$, by (**S.i**). In the former case $\Sigma' \vdash \mathtt{if}(u, e_1, e_2) \sim e_2$ by (**if.ii,R.i,E**), and so by (**CMI**)

$$\Sigma \vdash \Gamma; M; R[\![\mathtt{if}(u, e_1, e_2)]\!] \sim \Gamma; M; R[\![e_2]\!].$$

In the latter case $\Sigma' \vdash \mathtt{if}(u, e_1, e_2) \sim e_1$ by (**if.iii**) so again by (**CMI**)

$$\Sigma \vdash \Gamma; M; R[\![\mathtt{if}(u, e_1, e_2)]\!] \sim \Gamma; M; R[\![e_1]\!].$$

(beta) In this case $e = R[\![\mathtt{let}\{x := u\}e_0]\!]$, and by (**let.ii**) $\Sigma' \vdash \mathtt{let}\{x := u\}e_0 \sim e_0\{x := u\}$. Hence by (**CMI**)

$$\Sigma \vdash \Gamma; M; R[\![\mathtt{let}\{x := u\}e_0]\!] \sim \Gamma; M; R[\![e_0\{x := u\}]\!].$$

(delta) In this case $e = \delta(\bar{u})$ and consequently we may assume that

$$\Gamma; M[\![\delta(\bar{u})]\!] \rightarrow_\Sigma \Gamma'; M'[\![u']\!]$$

and $(\mathrm{Dom}(\Gamma') - \mathrm{Dom}(\Gamma)) \cap \mathrm{FV}(\Gamma; M; R[\![\delta(\bar{u})]\!]) = \emptyset$. The proof naturally divides up into seven cases, depending on δ. In four of these cases, corresponding to when $\vartheta \in \{atom, car, cdr, eq\}$, we have that $\Gamma = \Gamma'$ and $M = M'$. Consequently in these cases we need only show that $\Sigma' \vdash \vartheta(\bar{u}) \sim u'$ and invoke (**CMI**) to obtain the result. We begin by considering these four cases.

($\delta(\bar{u}) = atom(u)$) In this case there are two possiblities, either $u' = \mathtt{T}$ or $u' = \mathtt{Nil}$. In the former case we have that $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u)$ and so $\Sigma' \cup \mathrm{Th}(\mathbb{A}) \models atom(u)$. Hence by (**S.i**) we have that $\Sigma' \vdash \vartheta(\bar{u}) \sim u'$. Similarly in the latter case we have that $\Sigma_\Gamma^X \models \neg atom(u)$ and so $\Sigma' \models \neg atom(u)$. Hence again by (**S.i**) we have that $\Sigma' \vdash \vartheta(\bar{u}) \sim u'$.

($\delta(\bar{u}) = eq(u_0, u_1)$) Again there are two possiblities, either $u' = \mathtt{T}$ or $u' = \mathtt{Nil}$. In the former case we have that $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models u_0 = u_1$ and so by construction of Σ' and (**S.i**) we have that $\Sigma' \vdash eq(u_0, u_1) \sim \mathtt{T}$. In the case where $u' = \mathtt{Nil}$, we have that $\Sigma' \cup \mathrm{Th}(\mathbb{A}) \models u_0 \neq u_1$ and so by (**S.i**) $\Sigma' \vdash eq(u_0, u_1) \sim \mathtt{Nil}$.

($\delta(\bar{u}) = car(u)$) In this case we have that $\Sigma' \models car(u) = u'$ and hence by (**S.i**) $\Sigma' \vdash car(u) \sim u'$.

($\delta(\bar{u}) = cdr(u)$) This case is a trivial variation on *car*.

($\delta(\bar{u}) = cons(u_0, u_1)$) In this case we have that $\Gamma' = \Gamma\{u' := [u_0, u_1]\}$ and that $u' \notin \mathrm{Dom}(\Gamma) \cup X$. Now note that

$$\begin{aligned}
\Sigma' \vdash cons(u_0, u_1) &\sim setcdr(cons(u_0, \mathtt{T}), u_1) && \text{by } (\mathbf{set.vi}) \\
&\sim setcdr(setcar(cons(\mathtt{T}, \mathtt{T})), u_0), u_1) && \text{by } (\mathbf{set.v,R.i}) \\
&\sim \mathtt{let}\{u' := cons(\mathtt{T}, \mathtt{T})\}setcdr(setcar(u', u_0), u_1) && \text{by } (\mathbf{let.i,R.iii}) \\
&\sim \mathtt{let}\{u' := cons(\mathtt{T}, \mathtt{T})\}setcdr(\mathtt{seq}(setcar(u', u_0), u'), u_1) && \text{by } (\mathbf{set.iii,R.i,CMI}) \\
&\sim \mathtt{let}\{u' := cons(\mathtt{T}, \mathtt{T})\}\mathtt{seq}(setcar(u', u_0), setcdr(u', u_1)) && \text{by } (\mathbf{R.ii,CMI}) \\
&\sim \mathtt{let}\{u' := cons(\mathtt{T}, \mathtt{T})\}\mathtt{seq}(setcar(u', u_0), setcdr(u', u_1), u') && \text{by } (\mathbf{set.iii,CMI})
\end{aligned}$$

Thus we have shown that $\Sigma' \vdash cons(u_0, u_1) \sim \{u' := [u_0, u_1]\}; u'$ and so by (**CMI**)

$$\begin{aligned} \Sigma \vdash \Gamma; M; R[\![cons(u_0, u_1)]\!] &\sim \Gamma; M; R[\![\{u' := [u_0, u_1]\}; u']\!] \\ &\sim \Gamma; M; \{u' := [u_0, u_1]\}; R[\![u']\!] \qquad \text{by } (\mathbf{Rii,Riii,CMI}) \\ &\sim \Gamma\{u' := [u_0, u_1]\}; M; R[\![u']\!] \\ &\qquad \text{by } (\mathbf{Rii,Riii,cons.ii,cons.iii,CMI}) \text{ and } (\mathbf{Example.iii}) \end{aligned}$$

($\delta(\bar{u}) = setcar(u_0, u_1)$) In this case $u_0 = u'$ and there are two possibilities, either $u_0 \in \mathrm{Dom}(\Sigma)$ or $u_0 \in \mathrm{Dom}(\Gamma)$. In the latter case, assuming that $\Gamma(u_0) = [u_0^a, u_0^d]$ we have that $\Gamma' = \Gamma\{u_0 := [u_1, u_0^d]\}$. Now by (**set.iii**) $\Sigma' \vdash setcar(u_0, u_1) \sim \mathtt{seq}(setcar(u_0, u_1), u_0)$ and so by (**CMI**)

$$\begin{aligned} \Sigma \vdash \Gamma; M; R[\![setcar(u_0, u_1)]\!] &\sim \Gamma; M; R[\![\mathtt{seq}(setcar(u_0, u_1), u_0)]\!] \\ &\sim \Gamma; \mathtt{seq}(setcar(u_0, u_1), M; R[\![u_0]\!]) \qquad \text{by } (\mathbf{S.i,Rii,set.i,set.iv,CMI}) \\ &\sim \Gamma'; M; R[\![u']\!] \qquad \text{by } (\mathbf{S.i,Rii,set.ii,set.iv,CMI}) \end{aligned}$$

while in the former case, assuming that $u_0 \in \mathrm{Dom}(M)$ we have that $M' = M\{car(u_0) = u_1\}$. Now by (**set.iii**) $\Sigma' \vdash setcar(u_0, u_1) \sim \mathtt{seq}(setcar(u_0, u_1), u_0)$ and so by (**CMI**)

$$\begin{aligned} \Sigma \vdash \Gamma; M; R[\![setcar(u_0, u_1)]\!] &\sim \Gamma; M; R[\![\mathtt{seq}(setcar(u_0, u_1), u_0)]\!] \\ &\sim \Gamma; M; \mathtt{seq}(setcar(u_0, u_1), R[\![u_0]\!]) \qquad \text{by } (\mathbf{S.i,Rii,set.i,set.iv,CMI}) \\ &\sim \Gamma; M'; R[\![u']\!] \qquad \text{by } (\mathbf{S.i,Rii,set.ii,set.iv,set.ii,CMI}) \end{aligned}$$

The case when $(\exists z)(\Sigma \models z = u_0 \wedge z \in \mathrm{Dom}(M))$ is almost identical to the above argument.

($\delta(\bar{u}) = setcdr(u_0, u_1)$) This case is a trivial variation on *setcar*.

$\square_1$

Lemma (2): If Σ is $r(e)$-complete w.r.t. $[\mathrm{FV}(e), \mathrm{At}(\Sigma, e)]$ and $Coh(\Sigma)$, then there exists $\Gamma; M$ and an e' such that $e \overset{*}{\mapsto}_\Sigma \Gamma; M[\![e']\!]$ and exactly one of the following holds:

1. $e' = R[\![\vartheta(u)]\!]$, $\vartheta \in \{car, cdr\}$ and $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u)$.
2. $e' = R[\![set\vartheta(u_0, u_1)]\!]$, $set\vartheta \in \{setcar, setcdr\}$ and $\Sigma \cup \mathrm{Th}(\mathbb{A}) \models atom(u_0)$.
3. $e' = u$, and $Coh(\Sigma, \Gamma; M)$.

Proof (2): This follows from the simple observation that if $e \mapsto_\Sigma e'$ and Σ is $r(e)$-complete w.r.t. $[\mathrm{FV}(e), \mathrm{At}(\Sigma, e)]$ then Σ is $r(e')$-complete w.r.t. $[\mathrm{FV}(e'), \mathrm{At}(\Sigma, e')]$. Consequently the three cases above are the only ones in which further reduction is not possible. $\square_2$

Lemma (3): For any consistent $\Sigma, \bar{x}, \Phi \in \mathbb{L}$, and $n \in \mathbb{N}$ there exists $N \in \mathbb{N}$ and a family of constraint sets $\{\Sigma_i\}_{i<N}$ such that

1. Each Σ_i is n-complete w.r.t. $[\bar{x}, \mathrm{At}(\Sigma_i, \Phi)]$, and $Coh(\Sigma_i)$.
2. $(\forall \beta; \mu)(\beta; \mu \models_{\mathcal{L}} \Sigma \Leftrightarrow (\exists i < N)(\beta; \mu \models_{\mathcal{L}} \Sigma_i))$

3. $\dfrac{\Sigma_i \vdash \Phi \quad i < N}{\Sigma \vdash \Phi}$ is a derived rule.

Proof (3): This is a simple consequence of the introduction on the left rules. $\Box_3$

Lemma (4): Let $e_i = \Gamma_i; M_i[\![u_i]\!]$ with $Coh(\Sigma, \Gamma_i; M_i)$ for $i < 2$. If $\Sigma \models e_0 \sim e_1$ then $\Sigma \vdash e_0 \sim e_1$. Similarly if $\Sigma \models e_0 \simeq e_1$ then $\Sigma \vdash e_0 \simeq e_1$.

Proof (4): By lemma 0 we may assume that Σ is consistent. Using a simple construction from constants one can show that for any consistent Σ there is a $\beta; \mu$ such that

1. $\mathrm{Dom}(\beta) = \mathrm{FV}(\Sigma)$ and $\beta; \mu \models_{\mathcal{L}} \Sigma$,
2. $\beta(x) = \beta(y)$ iff $\Sigma \models x = y$.

Given such a $\beta; \mu$ we show that if $e_0; \beta; \mu \sim e_1; \beta; \mu$ then $\Sigma \vdash e_0 \sim e_1$. First observe that by $e_0; \beta; \mu \sim e_1; \beta; \mu$ and lemma 2 we can construct a bijection $f : \mathrm{Dom}(\Gamma_0) \to \mathrm{Dom}(\Gamma_1)$ such that (extending f as the identity off $\mathrm{Dom}(\Gamma_0)$) $\Sigma \models f(\vartheta_{\Gamma_0}(x)) = \vartheta_{\Gamma_1}(f(x))$ for all $x \in \mathrm{Dom}(\Gamma_0)$ and $\vartheta \in \{car, cdr\}$ and $\Sigma \models f(u_0) = u_1$. Consequently $\Gamma_0; u_0$ and $\Gamma_1; u_1$ differ only upto α-conversion and Σ-equality and hence we may assume they are the same. For $y \in \mathrm{Dom}(\vartheta_{M_0}) \cap \mathrm{Dom}(\vartheta_{M_1})$ we have $\Sigma \models \vartheta_{M_0}(y) = \vartheta_{M_1}(y)$ and we may assume they are the same. If $y \in \mathrm{Dom}(\vartheta_{M_0}) - \mathrm{Dom}(\vartheta_{M_1})$ then there must be some u such that $\Sigma \models \vartheta(y) = u$ otherwise we could choose μ such that $\vartheta_\mu(\beta(y))$ is not the value assigned by M_0. Using the derived rule (**Example.i**), $\{\neg atom(x)\} \vdash setcar(x, car(x)) \sim x$, we can remove y from $\mathrm{Dom}(M_0)$. Repeating this we can transform M_0 and M_1 into the same modification. Hence $\Sigma \vdash e_0 \sim e_1$. $\Box_\sim$

Now we show that

$$e_0; \beta; \mu \simeq e_1; \beta; \mu \;\Rightarrow\; \Sigma \vdash e_0 \simeq e_1.$$

If $e_0; \beta; \mu \simeq e_1; \beta; \mu$ then there exists $v; \mu'$ with $\mathrm{Dom}(\mu) \subseteq \mathrm{Dom}(\mu')$, μ_0, μ_1 with $\mu' \subseteq \mu_i$, and $\beta_i \supseteq \beta$ with $\beta_i(u_i) = v$ such that $e_i; \beta; \mu \overset{*}{\mapsto} u_i; \beta_i; \mu_i$. Now put

$$G_i = \{x \in \mathrm{Dom}(\Gamma_i) \mid \beta_i(x) \in \mathrm{Dom}(\mu_i) - \mathrm{Dom}(\mu')\}$$

Then by construction $u_i \notin G_i$ and if $x \in \mathrm{Dom}(\Gamma_i) - G_i$ then $\vartheta_{\Gamma_i}(x) \notin G_i$, for $\vartheta \in \{car, cdr\}$. Similarly if $x \in \mathrm{Dom}(M_i)$ then $\vartheta_{M_i}(x) \notin G_i$ for $\vartheta \in \{car, cdr\}$. Consequently we can show that

$$\Sigma \vdash \Gamma_i; M_i[\![u_i]\!] \sim \Gamma_{G_i}; \Gamma'_i; M_i[\![u_i]\!]$$

for Γ_{G_i} and Γ'_i memory contexts with the property that $\mathrm{Dom}(\Gamma_{G_i}) = G_i$ and

$$G_i \cap \mathrm{FV}(\Gamma'_i; M_i[\![u_i]\!]) = \emptyset.$$

Also note that, putting $e'_i = \Gamma'_i; M_i[\![u_i]\!]$, that $e'_0; \beta; \mu \sim e'_1; \beta; \mu$. Thus by the previous argument $\Sigma \vdash e'_0 \sim e'_1$, the result now follows from the garbage collection axioms $(\Gamma_{G_i}; e'_i) \simeq e'_i$ for $i < 2$. $\Box_\simeq$ $\Box_4$

6. Summary and Conclusions

We have presented a formal system for reasoning about equivalence of first-order Lisp- or Scheme-like programs that act on objects with memory. The semantics of the system is defined in terms of a notion of memory model derived from the natural operational semantics for the language. Equivalence is defined relative to classes of memory models defined by sets of constraints. The system is complete for programs that use only memory operations (no recursively defined functions, arithmetic operations, etc.). Thus the system can be seen to adequately express the semantics of memory operations. Presumably this could be extended to a relative completeness result for expressions built from memory and other algebraic operations, or for the full language, but we have not explored this possibility.

Equivalence in all models is the same as operational equivalence. Thus we have a means for reasoning about operational equivalence of programs. The formal system provides a richer language than operational equivalence since it provides a method for reasoning about conditional equivalence and equivalence with respect to restricted sets of contexts. This is essential for developing a theory of program transformations, since most of the interesting transformations are based on having additional information, i.e. on being able to restrict the contexts of use.

We could have omitted mentioning the equi-value relation $\sim$ and simply formalized undefinedness and strong isomorphism (see [Mason and Talcott 1989a]). However, equi-valuedness is an interesting relation in its own right and we have the stronger result giving soundness and completeness for all three relations. Our formal system also directly reflects the informal characterization of strong isomorphism as equi-valuedness modulo garbage collection.

Implicit in the proof of completeness is a decision procedure for deciding when an expression is defined and whether two expressions are equivalent for all models of a set of constraints. There are three key algorithms. The first algorithm is an algorithm for deciding first-order consequence for constraints by a simple extension of an algorithm for putting a set of equations and inequations into a canonical form. The second algorithm generates a set of $r(e)$-complete constraints each of which completely determines the computational behavior of the expressions in question. The third algorithm finds a renaming of bound variables of a memory context that transforms one object expression into another that is equivalent modulo a set of constraints, or proves that no such bijection exists. Mindless application of these algorithms of course results in combinatorial explosion. An interesting open problem is to find strategies that are reasonably efficient for a useful class of queries and to incorporate this into a system for reasoning about programs.

Work is in progress to extend the formal system to a full higher-order Scheme-like language (with untyped lambda abstraction). [Felleisen 1987, 1988] gives an equational calculus for reasoning about Scheme-like programs but such calculi do not deal adequately with conditional equivalence. The success of our approach in the first-order case depended on being able to define a semantics for conditional equivalence. In this case there is a natural model-theoretic equivalence (strong isomorphism) such that equivalence in all models is the same as operational equivalence. The existence of such a model-theoretic equivalence in the higher-order case remains an open question. The naive extension of the notion of strong isomorphism to the higher-order case does not work. Also operational equivalence in the first-order fragment does not imply equivalence in the higher-order language since non-atoms are no longer necessarily cells. Thus some refinement of the rules will be required.

Acknowledgements

We would like to thank the following people for carefully reading earlier versions of this paper, and pointing out numerous mistakes and confusions: Louis Galbiati, Matthias Felleisen, Furio Honsell, Jussi Ketonen, and Elizabeth Wolf.

7. References

Boehm, H.–J.

[1985] Side effects and aliasing can have simple axiomatic descriptions, *ACM TOPLAS*, **7(4)**, pp. 637–655.

Felleisen, M.

[1987] The calculi of lambda-v-cs conversion: A syntactic theory of control and state in imperative higher-order programming languages, Ph.D. thesis, Indiana University.

[1988] λ-v-CS: An extended λ-calculus for Scheme, *Proceedings of the 1988 ACM conference on Lisp and functional programming,* pp. 72–85.

Jørring, U. and Scherlis, W. L.

[1986] Deriving and using destructive data types, *IFIP TC2 working conference on program specification and transformation*, (North–Holland).

Mason, I. A.

[1986a] Equivalence of first order Lisp programs: proving properties of destructive programs via transformation, *First Annual Symposium on logic in computer science*, (IEEE).

[1986] The semantics of destructive Lisp, Ph.D. Thesis, Stanford University. CSLI Lecture Notes No. 5, Center for the Study of Language and Information, Stanford University.

[1988] Verification of programs which destructively alter data, *Science of Computer Programing*, **10**, pp. 177–210.

Mason, I. A. and Talcott, C. L.

[1985] Memories of S-expressions: Proving properties of Lisp-like programs that destructively alter memory, Department of Computer Science, Stanford University Report No. STAN-CS-85-1057

[1989a] Axiomatizing Operational Equivalence in the presence of Side Effects. *Fourth Annual Symposium on logic in computer science*, (IEEE).

[1989b] Programming, Transforming, and Proving with function abstractions and memories. Proceedings of the 16th EATCS Colloquium on Automata, Languages and Programming. Stresa. 1989.

Morris, J. H.

[1968] Lambda calculus models of programming languages, Ph.D. thesis, Massachusetts Institute of Technology.

Mosses, P.

[1984] A basic abstract semantic algebra, in: *Semantics of data types, international symposium, Sophia-Antipolis, June 1984, proceedings*, edited by G. Kahn, D. B. MacQueen, and G. Plotkin, Lecture notes in computer science, no. 173 (Springer, Berlin) pp. 87–108.

Nelson, C. G. and Oppen, D. C.

[1977] Fast decision procedures based on congruence closure, Computer Science Department Report STAN–CS–77–647, Stanford University.

Plotkin, G.

[1975] Call-by-name, call-by-value and the lambda calculus, *Theoretical Computer Science*, **1**, pp. 125–159.

Oppen, D. C.

[1978] Reasoning about recursively defined data structures, Computer Science Department Report STAN–CS–78–678, Stanford University.