J. Functional Programming **1** (1): 1–000, January 1993 © 1993 Cambridge University Press 1

Structured Operational Semantics of a fragment of the language Scheme †

Furio Honsell
Università di Udine, Dipartimento di Matematica e Informatica,
via Delle Scienze 208 - Udine, Italy.
E-mail `honsell@dimi.uniud.it`

Alberto Pravato, Simona Ronchi Della Rocca
Università di Torino, Dipartimento di Informatica,
C.so Svizzera 185 - Torino, Italy.
E-mail `{pravato,ronchi}@di.unito.it`

Abstract

In this paper we give a big-step *structured operational semantics* (SOS), in the style of Plotkin, Kahn and Milner, of a significant fragment of the functional programming language *Scheme*, including `quote`, `eval`, `quasiquote` and `unquote`. The SOS formalism allows us to discuss incrementally the various features of the language and to keep a low mathematical overhead, thus producing a rigorous account of the semantics of a "real" programming language, which nonetheless has a pedagogical value. More specifically, we formalize four strictly increasing fragments of Scheme, using a number of formal systems which express the evaluation of expressions, the display of output results, and the handling of errors.

1 Introduction

In order to use correctly, implement uniformly, or simply understand a programming language, we need a precise and unambiguous (formal) description of its semantics. In the literature, there are various *formal language description languages* and *techniques* which can be used for this purpose. Two important such techniques are the *denotational* and the *operational semantics*. According to the former, a program is interpreted as a function, living in a suitable mathematical space. This has proved to be very fruitful for proving properties of programs like correctness, termination, equality, etc. However the "dynamics" of program evaluation is often buried in a complex mathematical structure and it is not immediately apparent. In order to express this more clearly, the formalism of operational semantics seems more appropriate. In the operational approach, the semantics of a program is specified using the transition function of a suitable abstract machine (automaton). The transition

† This research was partially supported by EC contract CHRX-CT92.0046 Lambda Calul Typé and by EC Working Group 21900 Types.

function maps a configuration of the abstract machine to the configuration resulting from the execution of a computation step (*small step semantics*). However, in order to understand the behaviour of a program, the behaviour of the abstract machine itself has to be understood first, and this can be difficult if the programming language has elaborate features.

Structural Operational Semantics (SOS), introduced by Plotkin in (Plotkin, 1981) and further developed by Harper et al. (Harper *et al.*, 1987) and Kahn (Kahn, 1987), is a tool for specifying semantics, which combines positive features of operational and denotational semantics. The gist of SOS is to view the semantics of a programming language as a *formal system*, thereby reducing the processes of elaboration and execution to that of formally *deriving* particular *judgements* (*assertions*). SOS allows to describe naturally the dynamics of the execution of terminating programs, as operational semantics does. But this, without the need to define the details of an abstract machine, which are often cumbersome and not strictly necessary for understanding the language itself. On the other hand, SOS is a form of *reified* denotational semantics, which can be defined directly on the abstract syntax of the language, and can be naturally embedded into a program logic. But this, without any heavy mathematical overhead.

Many of the virtues of SOS semantics are particularly apparent when we focus on the relation between input and output, i.e. when we give the *big step semantics*. In this case one reduces the very process of *evaluation* to that of proof derivation and many interesting properties of programs (apart, of course, from non-termination) can be completely analyzed just by induction on the structure of proofs. This style of SOS semantics is particularly suitable for pedagogical purposes in that the global effect of each program phrase is immediately visible.

In this paper, by developing a case study of considerable size, we illustrate the applicability of big-step SOS semantics, and the *pedagogical value* deriving from the readability of its format. More specifically, we formalize the semantics of the core of a real functional programming language: *Scheme*, a LISP dialect. The fragment of the language Scheme, that we consider, includes the one introduced in (Abelson and Sussman, 1985), which amounts to the continuation-free fragment of the Standard Scheme (see (IEEE, 1990)). In addition we consider also the intriguing and problematic special symbols `quasiquote` and `eval`.

We feel that even a superficial comparison of our structured operational semantics of Scheme with its denotational semantics as given in (Rees and Clinger, 1986), illustrates clearly the differences in scope and enphasis of the two approaches to semantics, as discussed above.

In (Abelson and Sussman, 1985) the evaluation process of Scheme is described by means of a *meta-interpreter*. Various aspects of our SOS specification, and especially the representation of the environment, have been inspired by it. However, as a means of specifying the evaluation of Scheme programs, we think that our SOS approach is, in a sense, more abstract and more general, since a meta-interpreter can be easily designed starting from it.

Our SOS semantics of Scheme should be compared to other formal specifications of the operational semantics of untyped functional programming languages with imperative features appearing in the literature, such as those of Felleisen et al. (Felleisen and Friedman, 1989) and of Mason and Talcott (Mason and Talcott, 1991). We tried to stick as closely as possible to the syntax of a *real* programming language, including all its idiosyncrasies. They have considered somewhat more purified syntaxes, which are more directly embeddable into a logical system/calculus for reasoning about program equivalences.

It is important to point out that the operational semantics that we give agrees with the Standard description of Scheme given in (IEEE, 1990), except for the special symbol `eval`, for which there is no standard interpretation.

In recent years, a lot of energy has been put in trying to obtain a well justified semantics for `eval`, but as yet no definitive agreement has been reached. In all Lisp-like languages, such as Scheme, `eval` can be considered as a "built-in interpreter". In order to make this precise, however, the differences between a Lisp expression, to be processed by the interpreter, and its internal representation, to be interpreted by `eval`, have to be clarified. This issue is particularly explicit when specifying the behaviour of `eval` on internal representations of abstractions. In the present work we take a very liberal attitude and *do* evaluate internal representations of abstractions. Our definition of `eval` should be contrasted with those of Muller (Muller, 1992) and Smith (Smith, 1994).

Throughout the paper we use freely standard notions and terminology from proof theory. For the sake of completeness however we briefly recall a few crucial concepts which will be extensively used in the sequel.

We take a *formal system* to be a set of *rule schemata* of the shape:

$$\frac{\text{premise}_1 \ldots \text{premise}_n}{\text{conclusion}}$$

Where $\text{premise}_1, \ldots, \text{premise}_n$, and conclusion are *judgements*. The intended meaning of a rule schema is the following:

> for every instantiation S of the *schematic variables* occurring in the judgements, if the instantiations $S(\text{premise}_1) \ldots S(\text{premise}_n)$, have been *established* (derived), then we are allowed to establish $S(\text{conclusion})$.

A *derivation* is a decorated tree, whose nodes are labeled by instantiations of rules, which furthermore satisfies the *consequence constraint*, i.e. the premises of the parent node coincide with the conclusions of the children nodes.

In this paper we are not concerned with scanning or parsing issues. We assume, throughout the paper, that input expressions be strings of characters correctly parsed. On the other hand, we discuss in detail how the output is presented. For languages like Scheme, which have abstract objects as first class citizens, what is actually displayed on the monitor is not immediate.

In giving the semantics of Scheme, we proceed incrementally. We discuss, separately, four strictly increasing fragments gradually introducing the semantical concepts we need. Thus, we illustrate one of the most appealing features of SOS specifications. Namely, the possibility of tailoring the metalanguage to the language currently under consideration, keeping a low mathematical overhead.

In Section 2, we define and discuss the fragment of Scheme which has only integer numbers and booleans as basic data types. In Section 3, we consider an extension of this fragment obtained by introducing the fundamental operations on pairs. In Section 4, we extend the language with two special symbols, `quote` and `eval`. In order to specify their semantics, we have to formalize an intermediate compilation from correctly parsed expressions to expressions which are fed as input to the interpreter. In Section 5, we extend furthermore the language with the special symbols `quasiquote` and `unquote`. Section 6 is devoted to the handling of errors. Finally in Section 7, we formalize the semantics of proper Scheme Programs, consisting of lists of declarations and expressions.

An earlier version of this paper (Honsell and Ronchi, 1989), has been used, by two of the authors, as class notes for introductory courses in Programming Languages at the University of Torino and Udine (Italy).

2 The numerical fragment

In this section we give the operational semantics of the fragment of the language Scheme which manipulates only integer numbers as atomic data type. The evaluation process is captured essentially by a single formal system. Another formal system is necessary in order to formalize the output.

2.1 Syntax

We utilize freely BNF notation to define syntactic domains. Terminal symbols are written in typewriter style, this leaves us, of course, with some ambiguity concerning the blank character. We denote the generic objects of a given syntactic domain by the non-terminal symbol of the corresponding category, possibly with indices or primes. Non-terminal symbols, used as generic objects of a given category, are taken as pattern variables one matches against. We use bold characters to denote syntactic domains.

2.1.1 Syntactic domains

- **Char**, the domain of *characters*, ranged over by c, consists of arbitrary keyboard characters different from `(`, `)` and “ ” (blank).
- **Sym**, the domain of *symbols*, ranged over by s, consists of arbitrary strings of characters, i.e.
  $s ::= c | cs$
  The following are distinguished subdomains of **Sym**:
  - **K**, the subdomain of *constants*, ranged over by k,
    $k ::= \mathtt{0} | \mathtt{1} | \mathtt{-1} | \ldots | \mathtt{\#t} | \mathtt{\#f} | \ldots$
  - $\mathbf{Op}^{(n)}$, the subdomain of *standard operators of arity n*, ranged over by $op^{(n)}$ for $n \geq 1$,
    $op^{(1)} ::= \mathtt{abs} | \mathtt{even?} | \ldots$
    $op^{(2)} ::= * | + | \mathtt{eqv?} | \mathtt{max} | \ldots$
    $op^{(3)} ::= + | \ldots$

...
Let us denote $\bigcup_n \mathbf{Op}^{(n)}$ by **Op**.
— **Sp**, the subdomain of *special symbols*, ranged over by sp,
$sp ::=$ `lambda`|`define`|`if`|`begin`|`set!`
— **Var**, the subdomain of *identifiers*, ranged over by x, y, z or f, consists of those symbols which start with a character that cannot begin a numerical constant and which not belong neither to **K** nor to **Sp**. (Let us stress that $\mathbf{Op} \subseteq \mathbf{Var}$.)

- **S-symbols**, the domain of *Scheme Sentences*, ranged over by S, consists of those strings of `(`, `)`, " " and elements of **Sym**, defined by the following grammar:
$S ::= s|()|(S \ldots S)$
We define two subdomains of **S-symbols**: the subdomain of *declarations*, ranged over by d, and the subdomain of *expressions*, ranged over by e. These subdomains are defined respectively by the following grammars:
— **Declarations**
$d ::= (\texttt{define}\, x\, e)|(\texttt{define}\,(f\, x_1 \ldots x_n)\; \mathit{body}) \quad (n \geq 0)$
— **Expressions**
$e ::= x \mid k \mid (\texttt{lambda}\,(x_1 \ldots x_m)\; \mathit{body}) \mid (e_1 \ldots e_n) \mid$
$\quad (\texttt{if}\, e_1\, e_2\, e_3) \mid (\texttt{set!}\, x\, e) \mid (\texttt{begin}\, e_1 \ldots e_n) \quad (m \geq 0, n \geq 1).$
where *body* has the following shape:
$\mathit{body} ::= d_1 \ldots d_m\, e \quad (m \geq 0).$

Given a declaration d, we define $\mathcal{V}(d)$ to be the set of variables defined by d as follows:

$$\mathcal{V}(d) = \begin{cases} \{x\} & \text{if } d \equiv (\texttt{define}\, x\, e) \\ \{f\} & \text{if } d \equiv (\texttt{define}\,(f\, x_1 \ldots x_n)\; \mathit{body}) \end{cases}$$

2.2 Semantics

We utilize set theoretic notation or BNF to define semantic domains. In particular we denote with $[A \Rightarrow B]$ the set of partial functions whose domain is a *finite* subset of A and whose range is in B. If f is a partial function, $dom(f)$ denotes its domain. The everywhere undefined function is denoted by $\emptyset$. We use bold characters to denote semantic domains.

2.2.1 Semantic domains

- The domain **V** of *values* is defined as follows:
$\mathbf{V} = \underline{\mathbf{K}} \cup \underline{\mathbf{Op}} \cup \mathbf{C} \cup \{?\}$
where:
— $\underline{\mathbf{K}}$ is the subdomain of *semantic constants*. We will assume that $\mathbf{Z} \subseteq \underline{\mathbf{K}}$, where **Z** is the set of integer numbers. If $k \in \mathbf{K}$ is a (syntactic) constant, then $\underline{k} \in \underline{\mathbf{K}}$ will denote the corresponding semantic constant.

— $\underline{\mathbf{Op}}$ is the subdomain of *semantic operators*. If $op^{(n)} \in \mathbf{Op}^{(n)}$ is a standard operator, then $\underline{op^{(n)}} \in \underline{\mathbf{Op}}$ will denote the corresponding semantic operator. (e.g. addition is the semantic operator corresponding to the standard operator $+$.)

— $\mathbf{C}$ is the subdomain of *closures* (location closures).
Closures are pairs $<l_n, e>$ where l_n is a *location* and e is an expression of the shape (`lambda` $(x_1 \ldots x_m)$ *body*), $(m \geq 0)$. The set of locations **Loc** is a countable set of tokens that we take to be a copy of the set of natural numbers. Whenever clear from the context we will refer to l_n as n.

— ? is a dummy value with which variables are initialized by default.

We suppose $\underline{\mathbf{K}}$, $\underline{\mathbf{Op}}$, $\mathbf{C}$, **Loc** and $\{?\}$ to be pairwise disjoint. Let $\mathcal{D}$ ranges over these semantic domains. We denote with $=_{\mathcal{D}}$ the equality relation in $\mathcal{D}$. For what concerns $\underline{\mathbf{K}}$, $\underline{\mathbf{Op}}$, **Loc** and $\{?\}$, $=_{\mathcal{D}}$ is the natural equality, while on $\mathbf{C}$ it is defined as: $<l_n, e> =_{\mathbf{C}} <l_m, e'>$ iff $l_n =_{\mathbf{Loc}} l_m$.

- The domain $\mathbf{F}$ of *frames*, ranged over by ρ, is defined as $[\mathbf{Var} \Rightarrow \mathbf{V}]$. Starting from a frame ρ, we denote with $\rho[x, v]$ the frame such that: $dom(\rho[x, v]) = dom(\rho) \cup \{x\}$, and
$\rho[x, v](y) = \rho(y)$ on every $y \not\equiv x$, while $\rho[x, v](x) = v$.
- The domain **Env**, of *environments*, ranged over by ζ, is defined as

$$\mathbf{Env} = [\mathbf{Loc} \Rightarrow ((\mathbf{Loc} \cup \{\perp\}) \times \mathbf{F})].$$

If ζ is an environment then $Next(\zeta)$ denotes the smallest location (integer) which does not belong to the domain of ζ. We denote by $\zeta[n, <n', \rho>]$ the environment such that $dom(\zeta[n, <n', \rho>]) = dom(\zeta) \cup \{n\}$, and
$\zeta[n, <n', \rho>](m) = \zeta(m)$ on every $m \not\equiv n$, while $\zeta[n, <n', \rho>](n) = <n', \rho>$.

This particular choice of **Env** is justified by the concept of *block structure* described in (Abelson and Sussman, 1985) (page 27). Namely, viewing locations as pointers, an environment ζ can be intuitively seen as a tree of frames. Given a location n, $\zeta(n)$ allows to access to a pair of the shape $<n', \rho>$ or to a pair of the shape $<\perp, \rho>$. In the first case we have a frame ρ which is a node of the tree whose parent is the pair $\zeta(n')$, while in the second case we have the root of the tree. The root of a tree is the *top-level frame*. In such a frame the associations between the standard operators and their correspondent semantic operators will be present.

Note that, for every environment ζ and location $n \in dom(\zeta)$, there must be exactly one path reaching the top-level frame.

2.2.2 Judgements

The operational semantics is given by the formal systems $\mathcal{E}_1$ and $\mathcal{D}_1$. These formal systems prove judgements, respectively, of the shapes:

$$\zeta, n \vdash_{\mathcal{E}} e \to v, \zeta'$$

$$\zeta, n \vdash_{\mathcal{D}} d \to \zeta'$$

where ζ and ζ' are environments, n is a location, e is an expression, d is a declaration and v is a value. The intended meaning of the judgement $\zeta, n \vdash_{\mathcal{E}} e \to v, \zeta'$ is:

the evaluation of the expression e in the environment ζ, accessed via the location n, produces the value v and modifies the existing environment into the new environment ζ'.

The intended meaning of the judgement $\zeta, n \vdash_{\mathcal{D}} d \to \zeta'$ is:

the processing of the declaration d in the environment ζ, accessed via the location n, modifies the existing environment into the new environment ζ'.

In the formal system $\mathcal{E}_1$ the Rule *set* which an auxiliary formal system $\mathcal{R}$ formalizing the operation of environment updating. The judgements proved by $\mathcal{R}$ are of the shape:

$$\zeta, n, x, v \leadsto \zeta'$$

where ζ and ζ' are environments, n is a location, x is a variable and v is a value. The intended meaning of the judgement $\zeta, n, x, v \leadsto \zeta'$ is:

if ρ is the first frame in the path from the location n to the root of the environment tree, such that $x \in dom(\rho)$, then ρ is replaced by $\rho[x, v]$.

Moreover, Rule *appl 2* utilizes an auxiliary formal system $\vdash_{\mathcal{O}p}$ formalizing the behaviour of the semantic interpretation of the standard operators. The judgements regulated by $\vdash_{\mathcal{O}p}$ are of the shape:

$$\vdash_{\mathcal{O}p} \underline{op}^m(v_1, \ldots, v_m) \to v$$

where op^m is an m-ary standard operator, and $v_1, \ldots, v_m, v$ are values. The intended meaning of the judgement $\vdash_{\mathcal{O}p} \underline{op}^m(v_1, \ldots, v_m) \to v$ is:

the value of the semantics operator $\underline{op}^m$ applied to the values $v_1, \ldots, v_m$ is v.

2.2.3 The formal system $\mathcal{D}_1$

$$\text{(dec 1)}\ \frac{x \notin \mathbf{Op} \qquad \zeta, n \vdash_{\mathcal{E}} e \to v, \zeta' \qquad \zeta'(n) = <n', \rho>}{\zeta, n \vdash_{\mathcal{D}} (\texttt{define}\, x\, e) \to \zeta'[n, <n', \rho[x, v]>]}$$

$$\text{(dec 2)}\ \frac{\begin{array}{c} f \notin \mathbf{Op} \qquad \zeta(n) = <n', \rho> \\ <n', \rho[f, <n, (\texttt{lambda}\,(x_1 \ldots x_m)\ body)>]> = w \end{array}}{\zeta, n \vdash_{\mathcal{D}} (\texttt{define}\,(f\, x_1 \ldots x_m)\ body) \to \zeta[n, w]}$$

2.2.4 The formal system $\mathcal{E}_1$

$$\text{(const)}\ \frac{k \in \mathbf{K}}{\zeta, n \vdash_{\mathcal{E}} k \to \underline{k}, \zeta}$$

$$\text{(var 1)}\ \frac{\zeta(n) = <n', \rho> \qquad x \in dom(\rho) \qquad \rho(x) \neq ?}{\zeta, n \vdash_{\mathcal{E}} x \to \rho(x), \zeta}$$

$$\text{(var 2)}\ \frac{\zeta(n) = <n', \rho> \qquad x \notin dom(\rho) \qquad n' \neq \bot \qquad \zeta, n' \vdash_{\mathcal{E}} x \to v, \zeta}{\zeta, n \vdash_{\mathcal{E}} x \to v, \zeta}$$

$$\text{(lambda)}\ \frac{<n, (\texttt{lambda}\,(x_1 \ldots x_m)\ body)> = c}{\zeta, n \vdash_{\mathcal{E}} (\texttt{lambda}\,(x_1 \ldots x_m)\ body) \to c, \zeta}$$

$$\text{(appl 1)}\ \frac{\begin{array}{c} \zeta_0, n \vdash_{\mathcal{E}} e_0 \to <n', (\texttt{lambda}\,(x_1 \ldots x_m)\, d_1 \ldots d_p\, e)>, \zeta_1 \\ \{\zeta_i, n \vdash_{\mathcal{E}} e_i \to v_i, \zeta_{i+1}\}_{(1 \leq i \leq m)} \\ \{y_1, \ldots, y_q\} = \bigcup_{(1 \leq i \leq p)} \mathcal{V}(d_i) \\ \zeta'_1 = \zeta_{m+1}[Next(\zeta_{m+1}), <n', \emptyset[x_1, v_1] \ldots [x_m, v_m][y_1, ?] \ldots [y_q, ?]>] \\ \{\zeta'_i, Next(\zeta_{m+1}) \vdash_{\mathcal{D}} d_i \to \zeta'_{i+1}\}_{(1 \leq i \leq p)} \\ \zeta'_{p+1}, Next(\zeta_{m+1}) \vdash_{\mathcal{E}} e \to v, \zeta'' \end{array}}{\zeta_0, n \vdash_{\mathcal{E}} (e_0\, e_1 \ldots e_m) \to v, \zeta''}$$

$$\text{(appl 2)}\ \frac{\begin{array}{c} \zeta_0, n \vdash_{\mathcal{E}} e_0 \to \underline{op^{(m)}}, \zeta_1 \\ \{\zeta_i, n \vdash_{\mathcal{E}} e_i \to v_i, \zeta_{i+1}\}_{(1 \leq i \leq m)} \\ \vdash_{\mathcal{O}p} \underline{op^{(m)}}(v_1, \ldots, v_m) \to v \end{array}}{\zeta_0, n \vdash_{\mathcal{E}} (e_0\, e_1 \ldots e_m) \to v, \zeta_{m+1}}$$

$$\text{(set)}\ \frac{x \notin \mathbf{Op} \qquad \zeta, n \vdash_{\mathcal{E}} e \to v, \zeta' \qquad \zeta', n, x, v \rightsquigarrow \zeta''}{\zeta, n \vdash_{\mathcal{E}} (\texttt{set!}\, x\, e) \to v, \zeta''}$$

$$\text{(sequence)}\ \frac{\{\zeta_i, n \vdash_{\mathcal{E}} e_i \to v_i, \zeta_{i+1}\}_{(1 \leq i \leq m)}}{\zeta_1, n \vdash_{\mathcal{E}} (\texttt{begin}\, e_1 \ldots e_m) \to v_m, \zeta_{m+1}}$$

$$\text{(if 1)}\ \frac{\begin{array}{c} \zeta, n \vdash_{\mathcal{E}} e_1 \to v', \zeta' \qquad v' \neq \underline{\texttt{\#f}} \\ \zeta', n \vdash_{\mathcal{E}} e_2 \to v, \zeta'' \end{array}}{\zeta, n \vdash_{\mathcal{E}} (\texttt{if}\, e_1\, e_2\, e_3) \to v, \zeta''}$$

$$\text{(if 2)}\ \frac{\begin{array}{c} \zeta, n \vdash_{\mathcal{E}} e_1 \to \underline{\texttt{\#f}}, \zeta' \\ \zeta', n \vdash_{\mathcal{E}} e_3 \to v, \zeta'' \end{array}}{\zeta, n \vdash_{\mathcal{E}} (\texttt{if}\, e_1\, e_2\, e_3) \to v, \zeta''}$$

Some remarks are in order here.

Among the premises of Rule *appl 1*, a "new" frame is needed for storing the internally defined variables, according to the block structure of the language. The internally defined variables $y_1, \ldots, y_q$ are initialized with ?, the undefined value. Following (Abelson and Sussman, 1985), page 441, a Scheme interpreter must go into an error state if an attempt is made to use the value ?.

Rules *var 1*, *var 2* and *appl 1* reflect the fact that Scheme adopts the *lexical* *scoping* rule w.r.t. the variables definition. Given a program written in some programming language, the lexical scoping determines that the definition of a variable can be obtained examining the program text alone, hence in a *static* way, furthermore the body of a function must be evaluated in the environment in which the function is called extended with the frame in effect when the function was defined. Rules *var 1* and *var 2* show that the value of a variable can be obtained following the *static* chain in the tree represented by the environment ζ. Moreover, in Rule *appl 1*, the expression part of the closure is evaluated accessing the environment at the location specified in the closure itself.

The standard in (IEEE, 1990) states that the evaluation of an expression of the shape $(\texttt{set!}\, x\, e)$ yields an unspecified value. In the present operational semantics Rule *set* yields, as result, the value obtained from the evaluation of the expression e.

The standard in (IEEE, 1990), moreover, states that the evaluation of an expression of the shape $(e_0\, e_1 \ldots e_n)$ does not follow a specified order evaluating the expressions $e_0, e_1, \ldots, e_n$. In the present operational semantics, for simplicity, Rules *appl 1* and *appl 2* adopt a left-to-right order of evaluation. In order to consider an unspecified order of evaluation, Rule *appl 2*, for example, could be:

$$\text{(appl 2)}\ \frac{\begin{array}{c}\sigma : \{0,\ldots,m\} \Rightarrow \{0,\ldots,m\} \quad \text{is a permutation} \\ \{\zeta_i, n \vdash_{\mathcal{E}} e_{\sigma(i)} \rightarrow v_{\sigma(i)}, \zeta_{i+1}\}_{(0\le i\le m)} \\ v_0 = \underline{op}^{(m)} \qquad\qquad \vdash_{\mathcal{O}p} \underline{op}^{(m)}(v_1,\ldots,v_m) \rightarrow v\end{array}}{\zeta_0, n \vdash_{\mathcal{E}} (e_0\, e_1 \ldots e_m) \rightarrow v, \zeta_{m+1}}$$

2.2.5 *The formal system $\mathcal{R}$*

$$\text{(scan 1)}\ \frac{\zeta(n) = <n', \rho> \qquad x \in dom(\rho)}{\zeta, n, x, v \rightsquigarrow \zeta[n, <n', \rho[x, v]>]}$$

$$\text{(scan 2)}\ \frac{\begin{array}{c}\zeta(n) = <n', \rho> \qquad x \notin dom(\rho) \qquad n' \neq \bot \\ \zeta, n', x, v \rightsquigarrow \zeta'\end{array}}{\zeta, n, x, v \rightsquigarrow \zeta'}$$

2.2.6 *The formal system $\vdash_{\mathcal{O}p}$*

In this subsection, only the rules for the standard operator `eqv?` are shown, because the rules for the other numeric and boolean operators are straightforward.

In the following rules, $\mathcal{D}$, $\mathcal{D}_1$ and $\mathcal{D}_2$ denote semantic domains ranging over $\underline{\mathbf{K}}$, $\underline{\mathbf{Op}}$, **C**, **Loc** and $\{?\}$.

$$\text{(eqv 1)}\ \frac{v_1, v_2 \in \mathcal{D} \qquad v_1 =_{\mathcal{D}} v_2}{\vdash_{\mathcal{O}p} \underline{\texttt{eqv?}}(v_1, v_2) \rightarrow \texttt{\#t}}$$

$$\text{(eqv 2)}\ \frac{v_1, v_2 \in \mathcal{D} \qquad v_1 \neq_{\mathcal{D}} v_2}{\vdash_{\mathcal{O}p} \underline{\texttt{eqv?}}(v_1, v_2) \rightarrow \texttt{\#f}}$$

$$(\text{eqv } 3)\ \frac{v_1 \in \mathcal{D}_1 \quad v_2 \in \mathcal{D}_2 \quad \mathcal{D}_1 \neq \mathcal{D}_2}{\vdash_{\mathcal{O}p} \underline{\texttt{eqv?}}(v_1, v_2) \rightarrow \#\texttt{f}}$$

Example 1
We want to evaluate the expression $e \equiv (e_0\, e_1)$ where $e_1 \equiv \texttt{2}$ and $e_0 \equiv (\texttt{lambda}\,(x)\ (\texttt{define}\,(f)\,(+\,z\,z))\,(\texttt{define}\,z\,x)\ (f))$, in an environment ζ accessed through a location n, namely we search for a value v and an environment ζ' such that $\zeta, n \vdash_{\mathcal{E}} e \rightarrow v, \zeta'$. We obtain the following derivation:

$$\frac{\overline{\zeta, n \vdash_{\mathcal{E}} e_0 \rightarrow <n, e_0>, \zeta}^{(lambda)} \quad \overline{\zeta, n \vdash_{\mathcal{E}} \texttt{2} \rightarrow \underline{2}, \zeta}^{(const)} \quad \mathcal{D}er_1 \quad \mathcal{D}er_2 \quad \mathcal{D}er_3}{\zeta, n \vdash_{\mathcal{E}} e \rightarrow \underline{4}, \zeta_4}(appl1)$$

where $\mathcal{D}er_1$ is the following derivation:

$$\overline{\zeta_1, n' \vdash_{\mathcal{D}} (\texttt{define}\,(f)\,(+\,z\,z)) \rightarrow \zeta_2}^{(dec2)}$$

where $\zeta_1 = \zeta[Next(\zeta), <n, \emptyset[x, \underline{2}][f, ?][z, ?]>]$ and, posing $n' = Next(\zeta)$, $\zeta_2 = \zeta[n', <n, \emptyset[x, \underline{2}][z, ?][f, <n', (\texttt{lambda}\,(\,)\,(+\,z\,z))>]>]$. $\mathcal{D}er_2$ is the following derivation:

$$\frac{\overline{\zeta_2, n' \vdash_{\mathcal{E}} x \rightarrow \underline{2}, \zeta_2}^{(var1)}}{\zeta_2, n' \vdash_{\mathcal{D}} (\texttt{define}\,z\,x) \rightarrow \zeta_3}(dec1)$$

where $\zeta_3 = \zeta[n', <n, \emptyset[x, \underline{2}][f, <n', (\texttt{lambda}\,(\,)\,(+\,z\,z))>][z, \underline{2}]>]$. $\mathcal{D}er_3$ is the following derivation:

$$\frac{\overline{\zeta_3 n' f <n', (\texttt{lambda}\,(\,)\,(+\,z\,z))> \zeta_3}^{(var1)} \quad \mathcal{D}er_4}{\zeta_3, n' \vdash_{\mathcal{E}} (f) \rightarrow \underline{4}, \zeta_4}(appl1)$$

where $\mathcal{D}er_4$ is the following derivation:

$$\frac{\dfrac{\vdots \qquad \vdots}{\zeta_4, n'' \vdash_{\mathcal{E}} + \rightarrow \underline{+}, \zeta_4}(var2) \quad \dfrac{\vdots \qquad \vdots}{\zeta_4, n'' \vdash_{\mathcal{E}} z \rightarrow \underline{2}, \zeta_4}(var2) \quad \overline{\vdash_{\mathcal{O}p} \underline{+}(\underline{2}, \underline{2}) \rightarrow \underline{4}}}{\zeta_4, n'' \vdash_{\mathcal{E}} (+\,z\,z) \rightarrow \underline{4}, \zeta_4}(appl2)$$

where $\zeta_4 = \zeta_3[n'', <n', \emptyset>]$, for $n'' = Next(\zeta_3)$. □

2.3 The display

The *display value* of an expression e is the string of characters which is displayed on the monitor upon termination of the evaluation process of e. The domain of display values **DV**, ranged over by dv, is defined as follows:

$dv ::= n | x | \#\texttt{t} | \#\texttt{f} | \# < \texttt{PROCEDURE} >$

where n ranges over natural numbers in decimal notation and x ranges over identifiers.

The formal system $\mathcal{DS}_1$, defined below, is used to establish judgements of the shape:

$$\zeta, n \vdash_{\mathcal{D}s} e \rightarrow dv$$

where ζ is an environment, n is a location, e is an expression or a declaration and $dv \in \mathbf{DV}$. The intended meaning of the judgement $\zeta, n \vdash_{\mathcal{D}s} e \to dv$ is:

after the evaluation of the expression or declaration e, in the environment ζ accessed through the location n, the display value dv appears on the monitor.

2.3.1 The system $\mathcal{DS}_1$

$$\text{(definition)}\ \frac{\zeta, n \vdash_{\mathcal{D}} d \to \zeta' \quad \{x\} = \mathcal{V}(d)}{\zeta, n \vdash_{\mathcal{D}s} d \to x}$$

$$\text{(const)}\ \frac{\zeta, n \vdash_{\mathcal{E}} e \to \underline{k}, \zeta' \qquad \underline{k} \in \mathbf{N}}{\zeta, n \vdash_{\mathcal{D}s} e \to k}$$

$$\text{(bool 1)}\ \frac{\zeta, n \vdash_{\mathcal{E}} e \to \underline{\#\mathtt{t}}, \zeta'}{\zeta, n \vdash_{\mathcal{D}s} e \to \#\mathtt{t}}$$

$$\text{(bool 2)}\ \frac{\zeta, n \vdash_{\mathcal{E}} e \to \underline{\#\mathtt{f}}, \zeta'}{\zeta, n \vdash_{\mathcal{D}s} e \to \#\mathtt{f}}$$

$$\text{(procedure)}\ \frac{\zeta, n \vdash_{\mathcal{E}} x \to v, \zeta' \qquad v \in \mathbf{C} \cup \underline{\mathbf{Op}}}{\zeta, n \vdash_{\mathcal{D}s} x \to \# < \mathtt{PROCEDURE} >}$$

3 The extension with “pairs”

In this section we discuss the fragment of the language Scheme obtained by extending the one in Section 2 with the standard operations on pairs. Consequently, the operational semantics is expressed by a formal system which is a conservative extension of System $\mathcal{E}_1$.

3.1 Syntax

In order to include pairs and their standard operators, only the set **Op** of Section 2 must be augmented. Namely, let **Char**, **Sym**, **K**, **Var** and **S-symbols** be defined as in Section 2, and let the subdomain of standard operators be extended by the operators `car` and `cdr` belonging to $\mathbf{Op}^{(1)}$ and `cons`, `set–car!`, `set–cdr!` and `equal?` belonging to $\mathbf{Op}^{(2)}$. Let us define $\mathcal{P} = \{\mathtt{car}, \mathtt{cdr}, \mathtt{cons}, \mathtt{set\text{–}car!}, \mathtt{set\text{–}cdr!}, \mathtt{equal?}\}$.

3.2 Semantics

3.2.1 Semantic domains

- The domain **V** of *values* is extended to include the set **Adr** of addresses, namely:
  $\mathbf{V} = \underline{\mathbf{K}} \cup \underline{\mathbf{Op}} \cup \mathbf{Adr} \cup \mathbf{C} \cup \{?\}$
  where:

- $\underline{\mathbf{K}}$, $\underline{\mathbf{Op}}$, $\mathbf{C}$ and $\mathbf{Loc}$ are defined as before. $\underline{\mathbf{Op}}$ must contain the set of semantic operators corresponding to the set $\mathcal{P}$. The semantics of these operators will be given through the rules of System $\vdash_{\mathcal{O}p}$.
- $\mathbf{Adr}$ is the domain of *addresses*. $\mathbf{Adr}$ is into one-one correspondence with the set $\mathbf{N}$ of natural numbers except for the special address NIL. We denote the n^{th} address (different from NIL) by adr_n, and by adr a generic address. $\mathbf{Adr}$ is assumed to be disjoint from $\mathbf{C} \cup \mathbf{Loc}$.

• $\mathbf{F}$ and $\mathbf{Env}$ are defined as before.
• The domain $\mathbf{Str}$, of *structures*, ranged over by h, is defined as

$$\mathbf{Str} = [(\mathbf{Adr} \setminus \{\textsc{nil}\}) \Rightarrow (\mathbf{V} \times \mathbf{V})].$$

We denote by $Next(h)$ the address with the smallest index not in $dom(h)$. We denote by $h[adr_n, <v_1, v_2>]$ the structure obtained from h as follows: $dom(h[adr_n, <v_1, v_2>]) = dom(h) \cup \{adr_n\}$, and $h[adr_n, <v_1, v_2>](adr_m) = h(adr_m)$ for $n \neq m$, while
$h[adr_n, <v_1, v_2>](adr_n) = <v_1, v_2>$.

Notice that the presence of addresses is justified only by the semantics of the operators `set-car`! and `set-cdr`!, described after.

3.2.2 Judgements

The operational semantics is given by the formal systems $\mathcal{E}_2$ and $\mathcal{D}_2$. These formal systems prove judgements, respectively, of the shapes:

$$\zeta, n, h \vdash_{\mathcal{E}} e \to v, \zeta', h'$$

$$\zeta, n, h \vdash_{\mathcal{D}} d \to \zeta', h'$$

where ζ and ζ' are environments, n is a location, h and h' are structures, e is an expression, d is a declaration and v is a value. The intended meaning of the judgement $\zeta, n, h \vdash_{\mathcal{E}} e \to v, \zeta', h'$ is:

> the evaluation of the expression e in the environment ζ, accessed via the location n, w.r.t. the structure h, produces the value v, modifies the existing environment producing the new environment ζ' and modifies the existing structure yielding the new structure h'.

The intended meaning of the judgement $\zeta, n, h \vdash_{\mathcal{D}} d \to \zeta', h'$ is:

> the processing of the declaration d in the environment ζ, accessed via the location n, w.r.t. the structure h, modifies the existing environment producing the new environment ζ' and modifies the existing structure yielding the new structure h'.

The auxiliary formal system $\vdash_{\mathcal{O}p}$, defined in Subection 2.2.6 and utilized by Rule *appl 2* of $\mathcal{E}_2$, must be extended since the behaviour of some operator can depend on a particular structure which can be modified. The judgements have the following new shape:

$$h \vdash_{\mathcal{O}p} \underline{op}^m(v_1, \ldots, v_m) \to v, h'$$

where op^m is an m-ary standard operator, $v_1, \ldots, v_m$, v are values and h and h' are structures. The intended meaning of the judgement $h \vdash_{\mathcal{O}p} \underline{op}^m(v_1, \ldots, v_m) \to v, h'$ is:

the interpretation of the standard operator op^m applied to the values $v_1, \ldots, v_m$ in the structure h, yields the value v and modifies the structure h into h'.

3.2.3 The formal system $\mathcal{D}_2$

$$\text{(dec 1)}\ \frac{x \notin \mathbf{Op} \qquad \zeta, n, h \vdash_{\mathcal{E}} e \rightarrow v, \zeta', h' \qquad \zeta'(n) = <n', \rho>}{\zeta, n, h \vdash_{\mathcal{D}} (\texttt{define}\, x\, e) \rightarrow \zeta'[n, <n', \rho[x, v]>], h'}$$

$$\text{(dec 2)}\ \frac{\begin{array}{c} f \notin \mathbf{Op} \qquad \zeta(n) = <n', \rho> \\ <n', \rho[f, <n, (\texttt{lambda}\, (x_1 \ldots x_m)\ \mathit{body})>]> = w \end{array}}{\zeta, n, h \vdash_{\mathcal{D}} (\texttt{define}\, (f\, x_1 \ldots x_m)\ \mathit{body}) \rightarrow \zeta[n, w], h'}$$

3.2.4 The formal system $\mathcal{E}_2$

$$\text{(const)}\ \frac{k \in \mathbf{K}}{\zeta, n, h \vdash_{\mathcal{E}} k \rightarrow \underline{k}, \zeta, h}$$

$$\text{(var 1)}\ \frac{\zeta(n) = <n', \rho> \qquad x \in \mathit{dom}(\rho) \qquad \rho(x) \neq ?}{\zeta, n, h \vdash_{\mathcal{E}} x \rightarrow \rho(x), \zeta, h}$$

$$\text{(var 2)}\ \frac{\begin{array}{c} \zeta(n) = <n', \rho> \qquad x \notin \mathit{dom}(\rho) \qquad n' \neq \bot \\ \zeta, n', h \vdash_{\mathcal{E}} x \rightarrow v, \zeta, h \end{array}}{\zeta, n, h \vdash_{\mathcal{E}} x \rightarrow v, \zeta, h}$$

$$\text{(lambda)}\ \frac{<n, (\texttt{lambda}\, (x_1 \ldots x_m)\ \mathit{body})> = c}{\zeta, n, h \vdash_{\mathcal{E}} (\texttt{lambda}\, (x_1 \ldots x_m)\ \mathit{body}) \rightarrow c, \zeta, h}$$

$$\text{(appl 1)}\ \frac{\begin{array}{c} \zeta_0, n, h_0 \vdash_{\mathcal{E}} e_0 \rightarrow <n', (\texttt{lambda}\, (x_1 \ldots x_m)\, d_1 \ldots d_p\, e)>, \zeta_1, h_1 \\ \{\zeta_i, n, h_i \vdash_{\mathcal{E}} e_i \rightarrow v_i, \zeta_{i+1}, h_{i+1}\}_{(1 \leq i \leq m)} \\ \{y_1, \ldots, y_q\} = \bigcup_{(1 \leq i \leq p)} \mathcal{V}(d_i) \\ \zeta'_1 = \zeta_{m+1}[\mathit{Next}(\zeta_{m+1}), <n', \emptyset[x_1, v_1] \ldots [x_m, v_m][y_1, ?] \ldots [y_q, ?]>] \\ \{\zeta'_i, \mathit{Next}(\zeta_{m+1}), h'_i \vdash_{\mathcal{D}} d_i \rightarrow \zeta'_{i+1}, h'_{i+1}\}_{(1 \leq i \leq p)} \\ \zeta'_{p+1}, \mathit{Next}(\zeta_{m+1}), h'_{p+1} \vdash_{\mathcal{E}} e \rightarrow v, \zeta'', h'' \end{array}}{\zeta_0, n, h_0 \vdash_{\mathcal{E}} (e_0\, e_1 \ldots e_m) \rightarrow v, \zeta'', h''}$$

$$\text{(appl 2)}\ \frac{\begin{array}{c} \zeta_0, n, h_0 \vdash_{\mathcal{E}} e_0 \rightarrow \underline{op^{(m)}}, \zeta_1, h_1 \\ \{\zeta_i, n, h_i \vdash_{\mathcal{E}} e_i \rightarrow v_i, \zeta_{i+1}, h_{i+1}\}_{(1 \leq i \leq m)} \\ h_{m+1} \vdash_{\mathcal{O}p} \underline{op^{(m)}}(v_1, \ldots, v_m) \rightarrow v, h_{m+2} \end{array}}{\zeta_0, n, h_0 \vdash_{\mathcal{E}} (e_0\, e_1 \ldots e_m) \rightarrow v, \zeta_{m+1}, h_{m+2}}$$

$$\text{(set)}\ \frac{x \notin \mathbf{Op} \qquad \zeta, n, h \vdash_{\mathcal{E}} e \rightarrow v, \zeta', h' \qquad \zeta', n, x, v \rightsquigarrow \zeta''}{\zeta, n, h \vdash_{\mathcal{E}} (\texttt{set!}\, x\, e) \rightarrow v, \zeta'', h'}$$

$$\text{(sequence)}\ \frac{\{\zeta_i, n, h_i \vdash_{\mathcal{E}} e_i \to v_i, \zeta_{i+1}, h_{i+1}\}_{(1\le i\le m)}}{\zeta_1, n, h_1 \vdash_{\mathcal{E}} (\texttt{begin}\, e_1 \ldots e_m) \to v_m, \zeta_{m+1}, h_{m+1}}$$

$$\text{(if 1)}\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e_1 \to v', \zeta', h' \qquad v' \neq \underline{\texttt{\#f}} \quad \zeta', n, h' \vdash_{\mathcal{E}} e_2 \to v, \zeta'', h''}{\zeta, n, h \vdash_{\mathcal{E}} (\texttt{if}\, e_1\, e_2\, e_3) \to v, \zeta'', h''}$$

$$\text{(if 2)}\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e_1 \to \underline{\texttt{\#f}}, \zeta', h' \quad \zeta', n, h' \vdash_{\mathcal{E}} e_3 \to v, \zeta'', h''}{\zeta, n, h \vdash_{\mathcal{E}} (\texttt{if}\, e_1\, e_2\, e_3) \to v, \zeta'', h''}$$

3.2.5 The new version of the formal system $\vdash_{\mathcal{O}p}$

All the rules introduced in Subsection 2.2.6 are still valid, once every judgement of the shape $\vdash_{\mathcal{O}p} \underline{op}^m(v_1,\ldots,v_m) \to v$ has been replaced by one of the shape $h \vdash_{\mathcal{O}p} \underline{op}^m(v_1,\ldots,v_m) \to v, h$, and $\mathcal{D}$ ranges also over **Adr**. The new rules are:

$$\text{(cons)}\ \frac{}{h \vdash_{\mathcal{O}p} \underline{\texttt{cons}}(v_1, v_2) \to \mathit{Next}(h), h[\mathit{Next}(h), <v_1, v_2>]}$$

$$\text{(car)}\ \frac{v \in \mathbf{Adr} \setminus \{\text{NIL}\} \qquad h(v) = <v_1, v_2>}{h \vdash_{\mathcal{O}p} \underline{\texttt{car}}(v) \to v_1, h}$$

$$\text{(cdr)}\ \frac{v \in \mathbf{Adr} \setminus \{\text{NIL}\} \qquad h(v) = <v_1, v_2>}{h \vdash_{\mathcal{O}p} \underline{\texttt{cdr}}(v) \to v_2, h}$$

$$\text{(setcar)}\ \frac{v' \in \mathbf{Adr} \setminus \{\text{NIL}\} \qquad h(v') = <v_1, v_2>}{h \vdash_{\mathcal{O}p} \underline{\texttt{set-car!}}(v', v'') \to v'', h[v', <v'', v_2>]}$$

$$\text{(setcdr)}\ \frac{v' \in \mathbf{Adr} \setminus \{\text{NIL}\} \qquad h(v') = <v_1, v_2>}{h \vdash_{\mathcal{O}p} \underline{\texttt{set-cdr!}}(v', v'') \to v'', h[v', <v_1, v''>]}$$

$$\text{(equal 1)}\ \frac{v_1, v_2 \notin \mathbf{Adr} \quad h \vdash_{\mathcal{O}p} \underline{\texttt{eqv?}}(v_1, v_2) \to v, h}{h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_1, v_2) \to v, h}$$

$$\text{(equal 2)}\ \frac{\bar{\imath} \in \{0,1\} \quad v_{\bar{\imath}} \notin \mathbf{Adr} \quad v_{1-\bar{\imath}} \in \mathbf{Adr}}{h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_0, v_1) \to \underline{\texttt{\#f}}, h}$$

$$\text{(equal 3)}\ \frac{}{h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(\text{NIL}, \text{NIL}) \to \underline{\texttt{\#t}}, h}$$

$$(\text{equal } 4)\ \frac{\bar{\imath} \in \{0,1\} \quad v_{\bar{\imath}} \in \{\text{NIL}\} \quad v_{1-\bar{\imath}} \in \mathbf{Adr} \setminus \{\text{NIL}\}}{h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_0, v_1) \to \underline{\texttt{\#f}}, h}$$

$$(\text{equal } 5)\ \frac{\begin{array}{c} v_1, v_2 \in \mathbf{Adr} \setminus \{\text{NIL}\} \\ h(v_1) = <v_1', v_1''> \quad h(v_2) = <v_2', v_2''> \\ h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_1', v_2') \to \underline{\texttt{\#t}}, h \quad h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_1'', v_2'') \to v, h \end{array}}{h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_1, v_2) \to v, h}$$

$$(\text{equal } 6)\ \frac{\begin{array}{c} v_1, v_2 \in \mathbf{Adr} \setminus \{\text{NIL}\} \\ h(v_1) = <v_1', v_1''> \quad h(v_2) = <v_2', v_2''> \\ h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_1', v_2') \to \underline{\texttt{\#f}}, h \end{array}}{h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_1, v_2) \to \underline{\texttt{\#f}}, h}$$

Let us note that the rules for the evaluation of the operator `equal?` correspond to a recursive scanning of the arguments, if they belong to **Adr**. Consequently, there exist no a derivation for a judgement $h \vdash_{\mathcal{O}p} \underline{\texttt{equal?}}(v_1, v_2) \to v, h'$ if the addresses v_1 and v_2 access to a circular structure in h.

3.3 The display

The domain of display values **DV**, ranged over by dv, is extended as follows:

$dv ::= sdv | (list)$
$list ::= \epsilon | lne | lne.dv$
$lne ::= dv | dv\ lne$
$sdv ::= n | x | \texttt{\#t} | \texttt{\#f} | \# < \texttt{PROCEDURE} >$

where $list$ ranges over sequences of display values (ϵ denotes the empty sequence), lne ranges over non empty sequences of display values and sdv ranges over display values that are not lists.

The formal system $\mathcal{DS}_2$, defined below, is used to establish judgements of the shape:

$$\zeta, n, h \vdash_{\mathcal{D}s} e \to dv$$

where ζ is an environment, n is a location, h is a structure, e is an expression or a declaration and $dv \in \mathbf{DV}$. The intended meaning of the judgement $\zeta, n, h \vdash_{\mathcal{D}s} e \to dv$ is:

after the evaluation of the expression or declaration e, in the environment ζ accessed through the location n over the structure h, the display value dv appears on the monitor.

3.3.1 The system $\mathcal{DS}_2$

$$(\text{definition})\ \frac{\zeta, n, h \vdash_{\mathcal{D}} d \to \zeta', h' \quad \{x\} = \mathcal{V}(d)}{\zeta, n, h \vdash_{\mathcal{D}s} d \to x}$$

$$(\text{const})\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e \to \underline{k}, \zeta', h' \qquad \underline{k} \in \mathbf{N}}{\zeta, n, h \vdash_{\mathcal{D}s} e \to k}$$

$$(\text{bool 1})\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e \to \underline{\#\mathtt{t}}, \zeta', h'}{\zeta, n, h \vdash_{\mathcal{D}s} e \to \#\mathtt{t}}$$

$$(\text{bool 1})\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e \to \underline{\#\mathtt{f}}, \zeta', h'}{\zeta, n, h \vdash_{\mathcal{D}s} e \to \#\mathtt{f}}$$

$$(\text{procedure})\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e \to v, \zeta', h' \qquad v \in \mathbf{C} \cup \underline{\underline{\mathbf{Op}}}}{\zeta, n, h \vdash_{\mathcal{D}s} e \to \# < \mathtt{PROCEDURE} >}$$

$$(\text{address})\ \frac{\begin{array}{c}\zeta, n, h \vdash_{\mathcal{E}} e \to v, \zeta', h' \\ v \in \mathbf{Adr} \qquad h', v \vdash_h list\end{array}}{\zeta, n, h \vdash_{\mathcal{D}s} e \to (list)}$$

The third premise in Rule *address* above is a new judgement form:

$$h, adr \vdash_h list$$

where h is a structure, adr is an address and $list$ is a sequence of display values. The intended meaning of the judgement $h, adr \vdash_h list$ is:

> $list$ is the sequence of display values contained in the structure h starting from address adr, as it appears on the monitor.

This judgement obeys the rules in the following formal system $\vdash_h$:

$$(\text{rule 1})\ \frac{}{h, \text{NIL} \vdash_h \epsilon}$$

$$(\text{rule 2})\ \frac{\begin{array}{c} h(adr) = <v_1, v_2> \\ v_1, v_2 \in \mathbf{Adr} \\ h, v_1 \vdash_h list \\ h, v_2 \vdash_h lne \end{array}}{h, adr \vdash_h (list)lne}
\qquad
(\text{rule 3})\ \frac{\begin{array}{c} h(adr) = <v_1, v_2> \\ v_1, v_2 \in \mathbf{Adr} \\ h, v_1 \vdash_h list \\ h, v_2 \vdash_h lne.dv \end{array}}{h, adr \vdash_h (list)lne.dv}$$

$$(\text{rule 4})\ \frac{\begin{array}{c} h(adr) = <v_1, v_2> \\ v_1 \notin \mathbf{Adr},\ v_2 \in \mathbf{Adr} \\ v_1 \hookrightarrow sdv \\ h, v_2 \vdash_h lne \end{array}}{h, adr \vdash_h sdv\ \ lne}
\qquad
(\text{rule 5})\ \frac{\begin{array}{c} h(adr) = <v_1, v_2> \\ v_1 \notin \mathbf{Adr},\ v_2 \in \mathbf{Adr} \\ v_1 \hookrightarrow sdv \\ h, v_2 \vdash_h lne.dv \end{array}}{h, adr \vdash_h sdv\ \ lne.dv}$$

$$(\text{rule 6})\ \frac{\begin{array}{c} h(adr) = <v_1, v_2> \\ v_1 \in \mathbf{Adr},\ v_2 \notin \mathbf{Adr} \\ h, v_1 \vdash_h list \\ v_2 \hookrightarrow sdv \end{array}}{h, adr \vdash_h (list).\ sdv}
\qquad
(\text{rule 7})\ \frac{\begin{array}{c} h(adr) = <v_1, v_2> \\ v_1, v_2 \notin \mathbf{Adr} \\ v_1 \hookrightarrow sdv_1 \\ v_2 \hookrightarrow sdv_2 \end{array}}{h, adr \vdash_h sdv_1.sdv_2}$$

where, among the premises of the rules *1*, …, *7* above, there are judgements managed by the following formal system $\hookrightarrow$:

$$\text{(rule 1)}\ \frac{}{\underline{\#\mathtt{t}} \hookrightarrow \#\mathtt{t}} \qquad\qquad \text{(rule 2)}\ \frac{}{\underline{\#\mathtt{f}} \hookrightarrow \#\mathtt{f}}$$

$$\text{(rule 3)}\ \frac{\underline{k} \in \mathbf{N}}{\underline{k} \hookrightarrow k} \qquad\qquad \text{(rule 4)}\ \frac{c \in \mathbf{C} \cup \underline{\mathbf{Op}}}{c \hookrightarrow \# < \mathtt{PROCEDURE} >}$$

Example 2
Let us view the output after the evaluation of the expression
$e \equiv (\mathtt{cons}\,1\,(\mathtt{cons}\,2\,3))$ in a given environment ζ, accessed through a location n, and structure h. First of all we have the following evaluation in $\mathcal{E}_2$:

$$\frac{\dfrac{\vdots \qquad \vdots}{\zeta, n, h \vdash_{\mathcal{E}} \mathtt{cons} \to \underline{\mathtt{cons}}, \zeta, h}{\scriptstyle (var2)} \quad \dfrac{}{\zeta, n, h \vdash_{\mathcal{E}} 1 \to \underline{1}, \zeta, h}{\scriptstyle (const)} \quad \mathcal{D}er_1 \quad \mathcal{D}er_2}{\zeta, n, h \vdash_{\mathcal{E}} e \to adr^2, \zeta, h[adr^1, <\underline{2}, \underline{3}>][adr^2, <\underline{1}, adr^1>]}{\scriptstyle (appl2)}$$

where $\mathcal{D}er_1$ is the following derivation:

$$\frac{\dfrac{\cdots}{\cdots}{\scriptstyle (var2)} \quad \dfrac{\cdots}{\cdots}{\scriptstyle (const)} \quad \dfrac{\cdots}{\cdots}{\scriptstyle (const)} \quad \dfrac{}{h \vdash_{\mathcal{O}p} \underline{\mathtt{cons}}(\underline{2}, \underline{3}) \to adr^1, h[adr^1, <\underline{2}, \underline{3}>]}{\scriptstyle (cons)}}{\zeta, n, h \vdash_{\mathcal{E}} (\mathtt{cons}\,\underline{2}\,\underline{3}) \to adr^1, \zeta, h[adr^1, <\underline{2}, \underline{3}>]}{\scriptstyle (appl2)}$$

for $adr^1 = Next(h)$, while $\mathcal{D}er_2$ is the following one:

$$\frac{}{h[adr^1, <\underline{2}, \underline{3}>] \vdash_{\mathcal{O}p} \underline{\mathtt{cons}}(\underline{1}, adr^1) \to adr^2, h[adr^1, <\underline{2}, \underline{3}>][adr^2, <\underline{1}, adr^1>]}{\scriptstyle (cons)}$$

Now, since letting $h' = h[adr^1, <\underline{2}, \underline{3}>][adr^2, <\underline{1}, adr^1>]$ we have the following derivation in $\vdash_h$:

$$\frac{\dfrac{}{\underline{1} \hookrightarrow 1}{\scriptstyle (3)} \quad \dfrac{\dfrac{}{\underline{2} \hookrightarrow 2}{\scriptstyle (3)} \quad \dfrac{}{\underline{3} \hookrightarrow 3}{\scriptstyle (3)}}{h', adr^1 \vdash_h \mathtt{2\,.\,3}}{\scriptstyle (7)}}{h', adr^2 \vdash_h \mathtt{1\ 2\,.\,3}}{\scriptstyle (4)}$$

in System $\mathcal{DS}_2$ the following judgement is derivable through Rule *address*:

$$\zeta, n, h \vdash_{\mathcal{D}s} (\mathtt{cons}\,1\,(\mathtt{cons}\,2\,3)) \to (\mathtt{1\ 2\,.\,3})$$

□

4 The fragment with `quote` and `eval`

In this section we extend further the fragment of the language Scheme under analysis. Namely, we discuss the special symbols `quote` and `eval`. To this end we need to consider as basic data types also literals. Indeed, if S is an S-sysmbol, the evaluation of $(\mathtt{quote}\,S)$ gives a literal as result, namely a list of symbolic data. The special symbol `eval` is considered as a “built-in” interpreter that processes literals as input.

Since `eval` is not specified in the standard definition of Scheme (see (IEEE, 1990)), various treatments of `eval` are possible. Our proposal is to minimize the differences between the semantics of (`eval` (`quote` S)) and the semantics of S itself. Moreover, we do not want to introduce any additional notation w.r.t. the orginal syntax. To this aim, if an S-symbol submitted to `quote` contains a `lambda` expression, the special symbol `eval` will process the resulting literal representation as if it contained a closure.

Other semantical accounts of the behaviour of `eval`/`quote`, can be found in the literature. For example, Muller, in (Muller, 1992), gives a complete coding of an input expression in terms of new syntactical operators. This is extremely natural in a setting in which the operational semantics is expressed as a term rewriting system, but it is awkward in an "input-output" (big-step) SOS setting, such as ours. Furthermore, in Muller's approach the symbol `eval` does not evaluate closures, since closures are not part of his internal representation of S-symbols.

As we anticipated in the introduction, in order to give the operational semantics of `eval` and `quote` we need two distinct formal systems: one to formalize the "read" process and one to formalize the "evaluation" process.

4.1 Syntax

Let all the subdomains of **Sym** be defined as in the previous section, but for that of special symbols, **Sp**, which is now extended with the two new elements `quote` and `eval`. The domain **S-symbols**, ranged over by S, is now defined as before starting from the new **Sym**. We define two special subdomains of **S-symbols**, the subdomain of *input declarations*, ranged over by *dcl*, and the subdomain of *input expressions*, ranged over by *exp*.

- **Input-Declarations**
  $dcl ::= (\mathtt{define}\, x\ exp) | (\mathtt{define}\, (f\, x_1 \ldots x_n)\ bexp)\ \ (n \geq 0)$
- **Input-Expressions**
  $exp ::= x | k | (\mathtt{lambda}\, (x_1 \ldots x_m)\ bexp) | (exp_1 \ldots exp_n) |$
  $(\mathtt{if}\ exp_1\ exp_2\ exp_3) | (\mathtt{set!}\, x\ exp) | (\mathtt{begin}\ exp_1 \ldots exp_n) |$
  $(\mathtt{quote}\, S) | (\mathtt{eval}\ exp) \qquad (m \geq 0, n \geq 1).$

where *bexp* has the following shape:
$bexp ::= dcl_1 \ldots dcl_m\ exp \quad (m \geq 0).$

4.2 Semantics

4.2.1 Semantic domains

- Let the set **Adr** be defined as before.
- **Tab** is the domain of *symbol-table addresses.* **Tab** is into one-one correspondence with the syntactic domain **Sym** and is assumed to be disjoint from **Adr**. We denote the symbol-table address corresponding to the symbol s as $\tau(s)$.
- The domain **InV**, of *input values*, consists of those objects which are fed to the interpreter, i.e. the output of a *precompilation* (reading) process. These

can either be *expression input values*, expressions for short, denoted by e, or *declaration input values*, declarations for short, denoted by d. These are inductively defined as:

— **Declarations**
$d ::= (\texttt{define}\, x\, e) | (\texttt{define}\, (f\, x_1 \ldots x_n)\, body) \quad (n \geq 0)$

— **Expressions**
$e ::= adr | \tau(s) | x | k$
$(\texttt{lambda}\, (x_1 \ldots x_m)\, body) | (e_1 \ldots e_n) |$
$(\texttt{if}\, e_1\, e_2\, e_3) | (\texttt{set!}\, x\, e) | (\texttt{begin}\, e_1 \ldots e_n) |$
$(\texttt{eval}\, e) \qquad (m \geq 0, n \geq 1).$

where *adr* are suitable objects different from strings of characters, denoting addresses, and *body* has the following shape:
$body ::= d_1 \ldots d_m\, e \quad (m \geq 0).$

- The domain **V** of *values* is now extended to include the domain of symbol table addresses **Tab**:
  $\mathbf{V} = \underline{\mathbf{K}} \cup \underline{\mathbf{Op}} \cup \mathbf{Adr} \cup \mathbf{C} \cup \mathbf{Tab} \cup \{?\}$
  where:
  — $\underline{\mathbf{K}}$, $\underline{\mathbf{Op}}$ and **Adr** are defined as before.
  — The subdomain of closures **C** is now defined considering expression input values, i.e. pairs $<l, e>$ where l is a location belonging to the subdomain **Loc** (defined as in Section 2) and e is an expression (input value) of the shape $(\texttt{lambda}\, (x_1 \ldots x_n)\, body)$, $(n \geq 0)$.

  The sets $\underline{\mathbf{K}}$, $\underline{\mathbf{Op}}$, **Adr**, **Tab**, **C**, **Loc** and $\{?\}$ are assumed to be pairwise disjoint.
- **F** and **Env** are defined as in the previous section using the new values.
  If $\zeta \in \mathbf{Env}$, then $NewVar(\zeta)$ denotes the smallest identifier which does not belong to the domains of the frames in the range of ζ, w.r.t. some fixed order relation between identifiers.
- **Str** is now defined in order to mark some addresses as *read-only*, namely:

$$\mathbf{Str} = [\mathbf{Adr} \setminus \{\text{NIL}\} \Rightarrow (\mathbf{V} \times \mathbf{V}) \times \mathbf{Bool}],$$

  where **Bool** is the boolean set $\{\underline{\texttt{\#t}}, \underline{\texttt{\#f}}\} \subset \underline{\mathbf{K}}$.

The new definition of **Str** is necessary because a pair obtained from the evaluation of a `quote` expression (a *literal* pair) cannot be modify by the standard operators `set-car!` and `set-cdr!`. Such a pair will be stored in a triple of the form $<v_1, v_2, \underline{\texttt{\#f}}>$.

4.2.2 Judgements

The operational semantics is given by three formal systems $\mathcal{I}exp$, $\mathcal{E}_3$ and $\mathcal{D}_3$
$\mathcal{I}exp$ is used for establishing judgements of the shape:

$$h \vdash_{\mathcal{I}e} exp \rightarrow e, h'$$

where h and h' are structures, exp is an input expression or an input declaration and

e is an expression or a declaration input value. The intended meaning of establishing the judgement $h \vdash_{\mathcal{I}e} exp \to e, h'$ is:

the input expression (declaration) exp produces, during the "read" process, the expression (declaration) input value e and modifies the structure h to h'.

The formal system $\mathcal{E}_3$ and $\mathcal{D}_3$ regulate respectively judgements of the shapes:

$$\zeta, n, h \vdash_{\mathcal{E}} e \to v, \zeta', h'$$

$$\zeta, n, h' \vdash_{\mathcal{E}} d \to \zeta', h$$

where ζ and ζ' are environments, n is a location, h and h' are structures, e is an expression input value, d is a declaration input value and v is a value. The intended meaning of the above judgements are exactly the intended meaning of the judgements of systems $\mathcal{E}_2$ and $\mathcal{D}_2$.

The formal system $\mathcal{E}_3$ utilizes two auxiliary formal systems which correspond to the system $\mathcal{R}$, defined in Subsection 2.2.5, with the appropriate modification concerning values, and to the system $\vdash_{\mathcal{O}p}$, defined in Subsection 2.2.6, with the appropriate modification concerning the structures. In particular, Rules *cons*, *setcar* and *setcdr* must be modified in order that they do not consider read-only addresses, namely:

$$(\text{cons})\ \frac{}{h \vdash_{\mathcal{O}p} \underline{\texttt{cons}}(v_1, v_2) \to \mathit{Next}(h), h[\mathit{Next}(h), <v_1, v_2, \underline{\#\texttt{t}}>]}$$

$$(\text{setcar})\ \frac{v' \in \mathbf{Adr} \setminus \{\text{NIL}\} \qquad h(v') = <v_1, v_2, \underline{\#\texttt{t}}>}{h \vdash_{\mathcal{O}p} \underline{\texttt{set-car!}}(v', v'') \to v'', h[v', <v'', v_2, \underline{\#\texttt{t}}>]}$$

$$(\text{setcdr})\ \frac{v' \in \mathbf{Adr} \setminus \{\text{NIL}\} \qquad h(v') = <v_1, v_2, \underline{\#\texttt{t}}>}{h \vdash_{\mathcal{O}p} \underline{\texttt{set-cdr!}}(v', v'') \to v'', h[v', <v_1, v'', \underline{\#\texttt{t}}>]}$$

4.2.3 The formal system $\mathcal{I}exp$

We shall not give all the rules which do not involve explicitly `quote` or `eval`. The omitted rules follow the same pattern of rules *eval 1* and *eval 2* below.

$$(\text{declaration})\ \frac{h \vdash_{\mathcal{I}e} exp \to e, h'}{h \vdash_{\mathcal{I}e} (\texttt{define}\, x\; exp) \to (\texttt{define}\, x\; e), h'}$$

$$(\text{symbol})\ \frac{}{h \vdash_{\mathcal{I}e} (\texttt{quote}\, s) \to \tau(s), h}$$

$$(\text{nil})\ \frac{}{h \vdash_{\mathcal{I}e} (\texttt{quote}\,(\,)) \to \text{NIL}, h}$$

$$(\text{pair})\ \frac{\begin{array}{c} h \vdash_{\mathcal{I}e} (\texttt{quote}\, S_1) \to e_1, h_1 \\ h_1 \vdash_{\mathcal{I}e} (\texttt{quote}\,(S_2 \ldots S_m)) \to e_2, h_2 \end{array}}{h \vdash_{\mathcal{I}e} (\texttt{quote}\,(S_1 \ldots S_m)) \to \mathit{Next}(h_2), h_2[\mathit{Next}(h_2), <e_1, e_2, \underline{\#\texttt{f}}>]}$$

$$\text{(eval 1)}\ \frac{h \vdash_{\mathcal{I}e}\ exp \rightarrow e, h'}{h \vdash_{\mathcal{I}e} (\mathtt{eval}\ exp) \rightarrow (\mathtt{eval}\ e), h'}$$

$$\text{(eval 2)}\ \frac{\{h_i \vdash_{\mathcal{I}e}\ exp_i \rightarrow e_i, h_{i+1}\}_{(1 \leq i \leq n)}}{h_1 \vdash_{\mathcal{I}e} (exp_1 \ldots exp_n) \rightarrow (e_1 \ldots e_n), h_{n+1}}$$

4.2.4 The formal systems $\mathcal{E}_3$ and $\mathcal{D}_3$

The formal systems $\mathcal{E}_3$ and $\mathcal{D}_3$ include all the rules of the formal systems $\mathcal{E}_2$ and $\mathcal{D}_2$, defined in Section 3, but the general variable e ranges now over expression input values, that d ranges over declaration input values and that environments and structures are defined over the new class of values. The system $\mathcal{E}_3$ consists moreover of the following extra rules:

$$\text{(address)}\ \frac{}{\zeta, n, h \vdash_{\mathcal{E}}\ adr \rightarrow\ adr, \zeta, h}$$

$$\text{(symbol)}\ \frac{}{\zeta, n, h \vdash_{\mathcal{E}}\ \tau(s) \rightarrow \tau(s), \zeta, h}$$

$$\text{(eval 1)}\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e \rightarrow v, \zeta', h' \qquad v \notin (\mathbf{Adr} \setminus \{\text{NIL}\})}{\zeta, n, h \vdash_{\mathcal{E}} (\mathtt{eval}\ e) \rightarrow v, \zeta', h'}$$

$$\text{(eval 2)}\ \frac{\begin{array}{c}\zeta, n, h \vdash_{\mathcal{E}} e \rightarrow\ adr_m, \zeta', h' \\ NewVar(\zeta'), h', adr_m \triangleright exp \\ h' \vdash_{\mathcal{I}e}\ exp \rightarrow e', h'' \\ \zeta'[Next(\zeta'), <n, \emptyset[NewVar(\zeta'), adr_m]>], Next(\zeta'), h'' \vdash_{\mathcal{E}} e' \rightarrow v, \zeta'', h'''\end{array}}{\zeta, n, h \vdash_{\mathcal{E}} (\mathtt{eval}\ e) \rightarrow v, \zeta'', h'''}$$

Rule *eval 2* uses a new form of judgement:

$$S, h, v \triangleright S'$$

where h is a structure, v is a value, S and S' are Scheme-sentences. The intended meaning of this judgement is:

the structure h, accessed through v, encodes the Scheme-sentence S', where semantic values are represented relatively to S.

The rules for this kind of judgement are:

$$\text{(rule 1)}\ \frac{}{S, h, \tau(s) \triangleright s} \qquad\qquad \text{(rule 2)}\ \frac{v \in \mathbf{C} \cup \underline{\mathbf{K}}}{S, h, v \triangleright S}$$

$$\text{(rule 3)}\ \frac{\begin{array}{c} m \geq 2 \\ h(adr) = <v_1, adr'> \\ (\texttt{car}\, S), h, v_1 \triangleright S_1 \\ (\texttt{cdr}\, S), h, adr' \triangleright (S_2 \ldots S_m) \end{array}}{S, h, adr \triangleright (S_1\, S_2 \ldots S_m)} \qquad \text{(rule 4)}\ \frac{\begin{array}{c} h(adr) = <v_1, \text{NIL}> \\ (\texttt{car}\, S), h, v_1 \triangleright S_1 \end{array}}{S, h, adr \triangleright (S_1)}$$

Example 3

We want to evaluate the input expression $(\texttt{eval}\, exp)$ where exp is

$$(\texttt{cons}\,(\texttt{quote}\,\texttt{abs})(\texttt{cons}(\texttt{cons}(\texttt{lambda}(y)y)(\texttt{cons}\, -2()))())).$$

Let h be a structure. In System $\mathcal{I}_{exp}$ we have

$$h \vdash_{\mathcal{I}e} exp \rightarrow e, h$$

where $e = (\texttt{cons}\, \tau(\texttt{abs})(\texttt{cons}(\texttt{cons}(\texttt{lambda}(y)y)(\texttt{cons}\, -2()))()))$.

Let ζ, n be, respectively, an environment and a location. Now we want to evaluate the expression $(\texttt{eval}\, e)$ in ζ, n, h, namely we search for a value v such that $\zeta, n, h \vdash_{\mathcal{E}} (\texttt{eval}\, e) \rightarrow v, \zeta'', h'$ for some ζ'' and h'.

In System $\mathcal{E}_3$ we have

$$\zeta, n, h \vdash_{\mathcal{E}} e \rightarrow adr, \zeta', h'$$

where $h' = h[adr, <\tau(\texttt{abs}), adr'>][adr', <adr'', \underline{\text{NIL}}>][adr'', <c, adr'''>][adr''', <-2, \underline{\text{NIL}}>]$, with $c = <n', (\texttt{lambda}(y)y)>$. Let $x = NewVar(\zeta')$. The following is a derivation in System $\triangleright$:

$$\dfrac{\dfrac{}{(\texttt{car}\, x), h', \tau(\texttt{abs}) \triangleright \texttt{abs}}{}^{(1)} \quad \dfrac{\dfrac{\dfrac{}{(\texttt{caadr}\, x), h', c \triangleright (\texttt{caadr}\, x)}{}^{(2)} \quad \dfrac{\dfrac{}{(\texttt{cadadr}\, x), h', -2 \triangleright (\texttt{cadadr}\, x)}{}^{(2)}}{(\texttt{cdadr}\, x), h', adr''' \triangleright ((\texttt{cadadr}\, x))}{}^{(4)}}{(\texttt{cadr}\, x), h', adr'' \triangleright ((\texttt{caadr}\, x)\, (\texttt{cadadr}\, x))}{}^{(3)}}{(\texttt{cdr}\, x), h', adr' \triangleright (((\texttt{caadr}\, x)\, (\texttt{cadadr}\, x)))}{}^{(4)}}{x, h', adr \triangleright (\texttt{abs}\, ((\texttt{caadr}\, x)\, (\texttt{cadadr}\, x)))}{}^{(3)}$$

where $(\texttt{cadr}\, x)$ stands for $(\texttt{car}(\texttt{cdr}\, x))$, $(\texttt{caadr}\, x)$ stands for $(\texttt{car}(\texttt{car}(\texttt{cdr}\, x)))$, and so on. System $\mathcal{I}_{exp}$ and System $\mathcal{E}_3$ give us

$$h' \vdash_{\mathcal{I}e} (\texttt{abs}\, ((\texttt{caadr}\, x)\, (\texttt{cadadr}\, x))) \rightarrow (\texttt{abs}\, ((\texttt{caadr}\, x)\, (\texttt{cadadr}\, x))), h'$$
$$\zeta'[n'', <n, \emptyset[x, adr]>], n'', h' \vdash_{\mathcal{E}} (\texttt{abs}\, ((\texttt{caadr}\, x)\, (\texttt{cadadr}\, x))) \rightarrow 2, \zeta'', h'$$

where $n'' = Next(\zeta')$. Furthermore, through *eval 2* of System $\mathcal{E}_3$ we have

$$\zeta, n, h \vdash_{\mathcal{E}} (\texttt{eval}\, e) \rightarrow 2, \zeta'', h'$$

□

4.3 The display

The domain of display values **DV** is extended as follows:

$dv ::= sdv|(list)$
$list ::= \epsilon|lne|lne.dv$
$lne ::= dv|dv\, lne$
$sdv ::= s|n|x|\#\mathtt{t}|\#\mathtt{f}|\# < \mathtt{PROCEDURE} >$

where s ranges over symbols.

The formal system $\mathcal{DS}_3$, defined below, is used to establish judgements of the shape:

$$\zeta, n, h \vdash_{\mathcal{D}s} e \rightarrow dv$$

where ζ is an environment, n is a location, h is a structure, e is an expression or a declaration input value and $dv \in \mathbf{DV}$. The intended meaning of the judgement $\zeta, n, h \vdash_{\mathcal{D}s} e \rightarrow dv$ is:

after the evaluation of the expression or declaration e, in the environment ζ accessed through the location n over the structure h, the display value dv appears on the monitor.

4.3.1 The system $\mathcal{DS}_3$

The system $\mathcal{DS}_3$ includes all the rules of the system $\mathcal{DS}_2$, defined in Subsection 3.3. e and d now denote respectively an expression input value and a declaration input value. Moreover, environments and structures are defined using the new class of values. The system $\mathcal{DS}_3$ consists moreover of the following extra rule:

$$\text{(symbol)}\ \frac{\zeta, n, h \vdash_{\mathcal{E}} e \rightarrow \tau(s), \zeta', h'}{\zeta, n, h \vdash_{\mathcal{D}s} e \rightarrow s}$$

Accordingly, we need to reinterpret also the formal system $\vdash_h$, of subsection 3.3 (notice that although the definition of **Str** is changed, from the point of view of the display only the first two component of a triple $h(adr)$ are useful), and to extend the formal system $\hookrightarrow$ by the following rule:

$$\text{(rule 5)}\ \frac{}{\tau(s) \hookrightarrow s}$$

which reflects the "self-quoting" nature of the constant symbols.

5 Adding `quasiquote` and `unquote`

The special symbol `quasiquote` has a behaviour similar to that of the symbol `quote` but, when it is used in combination with the new special symbol `unquote`, it allows the construction of literals in which some components are evaluated. In a sense one can say that the semantics of `quasiquote` blurs the strict separation between the "read" and the "evaluation" processes which we had so far.

In order to give the semantics of these new symbols, the shape of the judgements of the formal system $\mathcal{I}_{exp}$ must be modified. In fact, now, some evaluations can be done also in the "read" process. Furthermore, the judgements of System $\mathcal{I}exp$ must keep track also of the environment.

In any case, we shall design the semantics in such a way, that the evaluation of $(\texttt{quasiquote}\,S)$ will produce the same results as $(\texttt{quote}\,S)$ when `unquote` does not occurr in S.

5.1 Syntax

Let all the subdomains of **Sym** be defined as in the previous section, but for that of special symbols, **Sp**, which is now extended with the two new elements `quasiquote` and `unquote`. One of the two subdomains of **S-symbols**, the subdomain of *input expressions*, is extended in order to contain also these two input expressions: $(\texttt{quasiquote}\,S)|(\texttt{unquote}\,\mathit{exp})$.

5.2 Semantics

5.2.1 Semantic domains

Every semantic domain is defined as before except for **InV**, the domain of *input values*, in which the expression input values are extended in order to contain the set of closures **C** and the set $\{?\}$.

5.2.2 Judgements

The formal system $\mathcal{I}exp$ has now judgement of the shape:

$$\zeta, n, h, k \vdash_{\mathcal{I}e} \mathit{exp} \to e, \zeta', h'$$

where h and h' are structures, *exp* is an input expression or an input declaration, e is an expression or a declaration input value, ζ and ζ' are environments, n is a location and k a natural number. The intended meaning of the judgement $\zeta, n, h, k \vdash_{\mathcal{I}e} \mathit{exp} \to e, \zeta', h'$ is:

> the input expression (declaration) *exp* produces, during the "read" process, the expression (declaration) input value e, evaluating the "unquoted" expressions occurring in *exp*, at the *nesting level* k, in the environment ζ accessed through location n, and modifies the structure h to h' and the environment ζ to ζ'.

The intended meaning of *nesting level* will be clear from the rules below.

5.2.3 The new formal system $\mathcal{I}exp$

Every rule of the formal system $\mathcal{I}exp$ of the previous section must be changed only in the shape of the judgements. In order to obtain the new System $\mathcal{I}exp$, the following rules must be added:

$$(\text{quasiquote 1})\ \frac{}{\zeta, n, h, k \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,s) \to \tau(s), \zeta, h}$$

$$\text{(quasiquote 2)}\ \frac{}{\zeta, n, h, k \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(\,))\rightarrow \textsc{nil}, \zeta, h}$$

$$\text{(quasiquote 3)}\ \frac{\begin{array}{c} k > 0 \\ \zeta, n, h, k-1 \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(S)) \rightarrow e, \zeta', h' \end{array}}{\begin{array}{l} \zeta, n, h, k \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(\texttt{unquote}\,S)) \rightarrow \\ \qquad\qquad \mathit{Next}(h'), \zeta', h'[\mathit{Next}(h'), <\tau(\texttt{unquote}), e, \underline{\#\texttt{f}}>] \end{array}}$$

$$\text{(quasiquote 4)}\ \frac{\zeta, n, h, k+1 \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(S)) \rightarrow e, \zeta', h'}{\begin{array}{l} \zeta, n, h, k \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(\texttt{quasiquote}\,S)) \rightarrow \\ \qquad\qquad \mathit{Next}(h'), \zeta', h'[\mathit{Next}(h'), <\tau(\texttt{quasiquote}), e, \underline{\#\texttt{f}}>] \end{array}}$$

$$\text{(quasiquote 5)}\ \frac{\begin{array}{c} \zeta, n, h, 0 \vdash_{\mathcal{I}e} S \rightarrow e, \zeta', h' \\ \zeta', n, h' \vdash_{\mathcal{E}} e \rightarrow v, \zeta'', h'' \end{array}}{\zeta, n, h, 0 \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(\texttt{unquote}\,S)) \rightarrow \nu(v), \zeta'', h''}$$

$$\text{(quasiquote 6)}\ \frac{\begin{array}{c} S_1 \notin \{\texttt{quasiquote}, \texttt{unquote}\} \\ \zeta, n, h, k \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,S_1) \rightarrow e_1, \zeta_1, h_1 \\ \zeta_1, n, h_1, k \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(S_2 \ldots S_m)) \rightarrow e_2, \zeta_2, h_2 \end{array}}{\begin{array}{l} \zeta, n, h, k \vdash_{\mathcal{I}e} (\texttt{quasiquote}\,(S_1 \ldots S_m)) \rightarrow \\ \qquad\qquad \mathit{Next}(h_2), \zeta_2, h_2[\mathit{Next}(h_2), <e_1, e_2, \underline{\#\texttt{f}}>] \end{array}}$$

where ν is the identity function on $\mathbf{Tab} \cup \mathbf{Adr} \cup \mathbf{C} \cup \{?\} \cup \underline{\mathbf{Op}}$ and yields the symbol corresponding to a semantic constant, namely:

$$\begin{array}{cc} \nu(\tau(s)) = \tau(s) & \nu(\underline{k}) = \tau(k) \\ \nu(\mathit{adr}) = \mathit{adr} & \nu(\underline{op^{(m)}}) = \underline{op^{(m)}} \\ \nu(?) = ? & \nu(c) = c \end{array}$$

As one can see, the rules governing `quote` are closely related to those of `quasiquote`, introduced in Section 4. In effect, one can easily show by induction on the structure of derivations that, whenever `unquote` does not occurr in S, the result of (`quasiquote` S) coincides with that of (`quote` S) for all structures and environments. Thus we can formally substantiate the remark made in the introduction to this Section,

5.2.4 The formal system $\mathcal{E}_3$

The formal systems $\mathcal{E}_3$ and $\mathcal{D}_3$ include all the rules of the formal systems $\mathcal{E}_3$ and $\mathcal{D}_3$ of the previous section. Only the following rule must be added to system $\mathcal{E}_3$:

$$\text{(closure)}\ \frac{c \in \mathbf{C} \cup \underline{\mathbf{Op}}}{\zeta, n, h \vdash_{\mathcal{E}} c \rightarrow c, \zeta, h}$$

Example 4

Let us use the following standard abbreviations: for each S-symbol S and input expression *exp*, we write ‘S instead of ($\texttt{quasiquote}\,S$) and ,*exp* instead of ($\texttt{unquote}\,exp$).

We want to evaluate the following expression:

$$exp \equiv \text{‘}(f\ \text{‘}(x\ ,(+\ 1\ 2)\ ,(z\ ,(+\ 1\ 3))))$$

for given variables f, x and z.

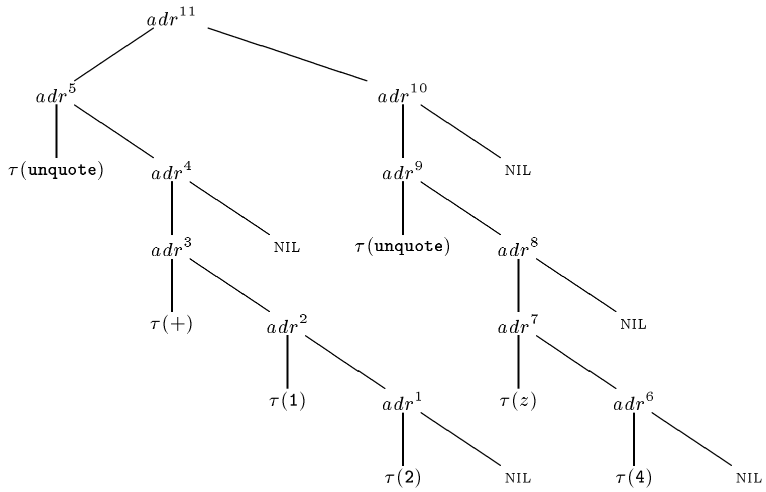
For lack of space, we write only two sub-derivations of the entire derivation for *exp* in $\mathcal{I}_{exp}$. Let ζ, n, h be, respectively, an environment, a location and a structure. Abbreviating in *qn* the name of the Rule *quasiquote* n, for $1 \leq n \leq 6$, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\zeta, n, h, 0 \vdash_{\mathcal{I}_e} \text{‘}(+\ 1\ 2) \to adr^3, \zeta, h^3} \quad \dfrac{}{\zeta, n, h^3, 0 \vdash_{\mathcal{I}_e} \text{‘}() \to \text{NIL}, \zeta, h^3}{\scriptstyle (q2)}}{\zeta, n, h, 0 \vdash_{\mathcal{I}_e} \text{‘}((+\ 1\ 2)) \to adr^4, \zeta, h^4}{\scriptstyle (q6)}}{\zeta, n, h, 1 \vdash_{\mathcal{I}_e} \text{‘},(+\ 1\ 2) \to adr^5, \zeta, h^5}{\scriptstyle (q3)} \quad \dfrac{\vdots \quad \mathcal{D}er \quad \vdots}{\zeta, n, h^5, 1 \vdash_{\mathcal{I}_e} \text{‘}(,(z\ ,(+\ 1\ 3))) \to adr^{10}, \zeta, h^{10}}{\scriptstyle (q6)}}{\zeta, n, h, 1 \vdash_{\mathcal{I}_e} \text{‘}(,(+\ 1\ 2)\ ,(z\ ,(+\ 1\ 3))) \to adr^{11}, \zeta, h^{11}}{\scriptstyle (q6)}$$

where a meaningful subderivation of $\mathcal{D}er$ is the following:

$$\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\zeta, n, h^5, 0 \vdash_{\mathcal{I}_e} (+\ 1\ 3) \to (+\ 1\ 3), \zeta, h^5} \quad \dfrac{\vdots \quad \vdots}{\zeta, n, h^5 \vdash_{\varepsilon} (+\ 1\ 3) \to \underline{4}, \zeta, h^5}}{\zeta, n, h^5, 0 \vdash_{\mathcal{I}_e} \text{‘},(+\ 1\ 3) \to \tau(4), \zeta, h^5}{\scriptstyle (q5)} \quad \dfrac{}{\zeta, n, h^5, 0 \vdash_{\mathcal{I}_e} \text{‘}() \to \text{NIL}, \zeta, h^5}{\scriptstyle (q2)}}{\zeta, n, h^5, 1 \vdash_{\mathcal{I}_e} \text{‘}(,(+\ 1\ 3)) \to adr^6, \zeta, h^6}{\scriptstyle (q6)}$$

where the structure h^{11}, accessed through location adr^{11}, can be viewed as the following tree:



Completing the derivation, we have

$$\zeta, n, h, 0 \vdash_{\mathcal{I}_e} \text{‘}(f\ \text{‘}(x\ ,(+\ 1\ 2)\ ,(z\ ,(+\ 1\ 3)))) \to adr^{14}, \zeta, h^{14}$$

where h^{14} is a modify of h^{11} and it such that the address adr^{14}, once evaluated in System $\mathcal{E}_3$, gives the following display:

$$\zeta, n, h \vdash_{\mathcal{D}s} adr^{14} \to (f\ \text{`}(x\ ,(+\ 1\ 2)\ ,(z\ 4)))$$

□

6 Errors

The systems introduced in the previous sections do not make distinctions between non-termination and "error." Indeed, assume that, for a given expression e, environment ζ, location n and structure h, there are no v, ζ' and h' such that the judgement $\zeta, n, h \vdash_{\mathcal{E}} e \to v, \zeta', h'$ can be derived. This means that either the evaluation of the expression e, with respect to ζ, n and h, does not terminate, or it gives rise to an error.

Since we suppose that the input expressions are correctly parsed, the only possible errors can be related either to the semantics of a standard operators or to an attempt of evaluating the value ?, or an undefined variable. For simplicity, we give to this last kynd of error the code 0, we give code 1 to the error concerning the value ?, and we give codes greater than 1 to errors arising from a bad application of a standard operator.

Let us see how System $\mathcal{E}_3$ must be modified in order to keep track of these errors. We omit the modifications on Systems $\mathcal{D}_3$, $\mathcal{I}exp$ and $\mathcal{DS}_3$, since they simply follow from the modifications on System $\mathcal{E}_3$. In particular, System $\mathcal{DS}_3$ must provide a suitable error message for each error code.

The judgements of System $\mathcal{E}_3$ must be extended to the following shape:

$$\zeta, n, h \vdash_{\mathcal{E}} e \to \texttt{error}, m$$

where ζ is an environment, n a location, h a structure, e an expression, `error` the *exceptional operational value* and m an error code. The intended meaning of the judgement $\zeta, n, h \vdash_{\mathcal{E}} e \to \texttt{error}, m$ is:

> the evaluation of the expression e in the environment ζ, accessed via the location n, w.r.t. the structure h, goes into the error whose code is m.

Moreover, also Systems $\mathcal{R}$ and $\vdash_{\mathcal{O}p}$ must have two different shapes of judgements. System $\mathcal{R}$ must have also judgements of this shape:

$$\zeta, n, x, v \rightsquigarrow \texttt{error}, m$$

while System $\vdash_{\mathcal{O}p}$ must have also judgements of this shape:

$$h \vdash_{\mathcal{O}p} \underline{op}^m(v_1, \ldots, v_m) \to \texttt{error}, m$$

where in both the cases `error` is the exceptional operational value and m an error code.

The rules that must be added to System $\mathcal{R}$ are:

$$(\text{fail})\ \frac{\zeta(n) = <\perp, \rho> \qquad x \notin dom(\rho)}{\zeta, n, x, v \rightsquigarrow \texttt{error}, 0}$$

and the following rule for the propagation of the exceptional operational value through a derivation:

$$(\text{prop})\ \frac{\zeta(n) = <n', \rho> \qquad x \notin dom(\rho) \qquad n' \neq \bot \qquad \zeta, n', x, v \rightsquigarrow \mathtt{error}, m}{\zeta, n, x, v \rightsquigarrow \mathtt{error}, m}$$

For what concerns System $\vdash_{\mathcal{O}p}$, the errors depend on the semantics of the standard operators. For example, the following rule must be added:

$$(\text{car-fail})\ \frac{v \notin \mathbf{Adr} \setminus \{\textsc{nil}\}}{h \vdash_{\mathcal{O}p} \underline{\mathtt{car}}(v) \to \mathtt{error}, m}$$

where m is a code grater than 1.

At the end, System $\mathcal{E}_3$. Here we must add the following rules:

$$(\text{var 1 fail})\ \frac{\zeta(n) = <n', \rho> \qquad x \in dom(\rho) \qquad \rho(x) = ?}{\zeta, n, h \vdash_{\mathcal{E}} x \to \mathtt{error}, 1}$$

$$(\text{var 2 fail})\ \frac{\zeta(n) = <\bot, \rho> \qquad x \notin dom(\rho)}{\zeta, n, h \vdash_{\mathcal{E}} x \to \mathtt{error}, 0}$$

$$(\text{appl 2 fail})\ \frac{\begin{array}{c}\zeta_0, n, h_0 \vdash_{\mathcal{E}} e_0 \to \underline{op^{(m)}}, \zeta_1, h_1 \\ \{\zeta_i, n, h_i \vdash_{\mathcal{E}} e_i \to v_i, \zeta_{i+1}, h_{i+1}\}_{(1 \leq i \leq m)} \\ h_{m+1} \vdash_{\mathcal{O}p} \underline{op^{(m)}}(v_1, \ldots, v_m) \to \mathtt{error}, m\end{array}}{\zeta_0, n, h_0 \vdash_{\mathcal{E}} (e_0\, e_1 \ldots e_m) \to \mathtt{error}, m}$$

$$(\text{set fail})\ \frac{x \notin \mathbf{Op} \qquad \zeta, n, h \vdash_{\mathcal{E}} e \to v, \zeta', h' \qquad \zeta', n, x, v \rightsquigarrow \mathtt{error}, 0}{\zeta, n, h \vdash_{\mathcal{E}} (\mathtt{set!}\, x\, e) \to \mathtt{error}, 0}$$

$$(\text{unreference})\ \frac{}{\zeta, n, h \vdash_{\mathcal{E}} ? \to \mathtt{error}, 1}$$

Moreover rules for the propagation of the error through a derivation must be added. These rules are simply a copy of the other rules of System $\mathcal{E}_3$ where the exceptional operational value is copied from a premise to the conclusion. We give only the following rule as an example:

$$(\text{prop sequence})\ \frac{\begin{array}{c}\{\zeta_i, n, h_i \vdash_{\mathcal{E}} e_i \to v_i, \zeta_{i+1}, h_{i+1}\}_{(1 \leq i \leq p < m)} \\ \zeta_{p+1}, n, h_{p+1} \vdash_{\mathcal{E}} e_{p+1} \to \mathtt{error}, k\end{array}}{\zeta_1, n, h_1 \vdash_{\mathcal{E}} (\mathtt{begin}\, e_1 \ldots e_m) \to \mathtt{error}, k}$$

7 Semantics of Scheme Programs

In this section we give the semantics of programs written with the fragment of the language Scheme considered in the previous section. To this end, we introduce a new formal system which uses, besides a new judgement, all the various forms of judgement introduced earlier. The corresponding system for defining the output of a program is introduced as usual.

A *Scheme Program* is a sequence: $<ed_1, \ldots, ed_m>$ $(m \geq 1)$, where the ed_i's are input expressions or input declarations. The *value* of the Scheme Program $<ed_1, \ldots, ed_m>$ w.r.t. a structure h and an environment ζ, accessible via a location n, is a list of values $<v_1, \ldots, v_p>$ with $p \leq m$. This value is defined using the formal system $\mathcal{S}$, which allows to establish judgements of the two forms:

$$\zeta, n, h \vdash_{\mathcal{S}} <ed_1, \ldots, ed_m> \rightarrow <v_1, \ldots, v_p>, \zeta', h'$$

$$\zeta, n, h \vdash_{\mathcal{S}} <ed_1, \ldots, ed_m> \rightarrow \texttt{error}, k$$

The intended meaning of the first judgement is:

the evaluation of the Scheme Program $<ed_1, \ldots, ed_m>$, w.r.t. the structure h and the environment ζ, accessed via the location n, produces the list of values $<v_1, \ldots, v_p>$, $v_i \in \mathbf{V}$, and moreover mutates the environment to ζ' and the structure to h',

while the intended meaning of the second judgement is:

the evaluation of the Scheme Program $<ed_1, \ldots, ed_m>$, w.r.t. the structure h and the environment ζ, accessed via the location n, reports the error with code k.

7.1 The formal system $\mathcal{S}$

$$(\text{decl } 1)\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} dcl \rightarrow d, \zeta', h' \\ \zeta', n, h'' \vdash_{\mathcal{E}} d \rightarrow \zeta'', h'\end{array}}{\zeta, n, h \vdash_{\mathcal{S}} <dcl> \rightarrow <\quad>, \zeta'', h''}$$

$$(\text{decl } 2)\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} dcl \rightarrow d, \zeta', h' \\ \zeta', n, h'' \vdash_{\mathcal{E}} d \rightarrow \zeta'', h' \\ \zeta'', n, h'' \vdash_{\mathcal{S}} <ed_1, \ldots, ed_m> \rightarrow <v_1, \ldots, v_p>, \zeta''', h'''\end{array}}{\zeta, n, h \vdash_{\mathcal{S}} <dcl, ed_1, \ldots, ed_m> \rightarrow <v_1, \ldots, v_p>, \zeta''', h'''}$$

$$(\text{exp } 1)\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} exp \rightarrow e, \zeta', h' \\ \zeta', n, h' \vdash_{\mathcal{E}} e \rightarrow v, \zeta'', h''\end{array}}{\zeta, n, h \vdash_{\mathcal{S}} <exp> \rightarrow <v>, \zeta'', h''}$$

$$(\text{exp } 2)\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} exp \rightarrow e, \zeta', h' \\ \zeta', n, h' \vdash_{\mathcal{E}} e \rightarrow v, \zeta'', h'' \\ \zeta'', n, h'' \vdash_{\mathcal{S}} <ed_1, \ldots, ed_m> \rightarrow <v_1, \ldots, v_p>, \zeta''', h'''\end{array}}{\zeta, n, h \vdash_{\mathcal{S}} <exp, ed_1, \ldots, ed_m> \rightarrow <v, v_1, \ldots, v_p>, \zeta''', h'''}$$

$$\text{(fail 1)}\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} exp \to e, \zeta', h' \\ \zeta', n, h \vdash_{\mathcal{E}} e \to \mathtt{error}, m\end{array}}{\zeta, n, h \vdash_{\mathcal{S}} <exp, ed_1, \ldots, ed_m> \to \mathtt{error}, <v, v_1, \ldots, v_p>\ k}$$

We omit the other rules for handling the exceptional operational value error.

Definition 1
Let $\zeta_\emptyset$ be the *initial environment*, i.e. $dom(\zeta_\emptyset) = \{0\}$ and $\zeta_\emptyset(0) = <\perp, \emptyset>$. A Scheme Program $<ed_1, \ldots, ed_m>$ is *terminating* if one of the following situations occurs:

1. $\exists \zeta, h$ such that: $\zeta_\emptyset, 0, \emptyset \vdash_{\mathcal{S}} <ed_1, \ldots, ed_m> \to <v_1, \ldots, v_p>, \zeta, h$,
2. $\exists k$ such that: $\zeta_\emptyset, 0, <ed_1, \ldots, ed_m> \vdash_{\mathcal{S}} <v_1, \ldots, v_p> \to \mathtt{error}, k$

In the first case the terminating Scheme Program is *correct* and *has value* $<v_1, \ldots, v_p>$ while in the second case it *reports the error with code k*.

7.2 The display

The *display value* of a terminating Scheme Program is defined using the formal system $\mathcal{SDS}$ below. The judgements of the system $\mathcal{SDS}$ are of the form:

$$\zeta, n, h \vdash_{\mathcal{S}dy} <ed_1, \ldots, ed_m> \to <dv_1, \ldots, dv_m>$$

where $<dv_1, \ldots, dv_m>$ is a list of display values.

7.2.1 The system $\mathcal{SDS}$

We omit the rules useful for display error messages.

$$\text{(rule 1)}\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} ed \to e, \zeta', h' \\ \zeta', n, h' \vdash_{\mathcal{D}s} e \to dv\end{array}}{\zeta, n, h \vdash_{\mathcal{S}dy} <ed> \to <dv>}$$

$$\text{(rule 2)}\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} dcl \to d, \zeta', h' \\ \zeta', n, h' \vdash_{\mathcal{D}s} d \to dv \\ \zeta', n, h'' \vdash_{\mathcal{E}} d \to \zeta'', h' \\ \zeta'', n, h'' \vdash_{\mathcal{S}dy} <ed_1, \ldots, ed_m> \to <dv_1, \ldots, dv_m>\end{array}}{\zeta, n, h \vdash_{\mathcal{S}dy} <dcl, ed_1, \ldots, ed_m> \to <dv, dv_1, \ldots, dv_m>}$$

$$\text{(rule 3)}\ \frac{\begin{array}{c}\zeta, n, h, 0 \vdash_{\mathcal{I}e} exp \to e, \zeta', h' \\ \zeta', n, h' \vdash_{\mathcal{D}s} e \to dv \\ \zeta', n, h' \vdash_{\mathcal{E}} e \to v, \zeta'', h'' \\ \zeta'', n, h'' \vdash_{\mathcal{S}dy} <ed_1, \ldots, ed_m> \to <dv_1, \ldots, dv_m>\end{array}}{\zeta, n, h \vdash_{\mathcal{S}dy} <exp, ed_1, \ldots, ed_m> \to <dv, dv_1, \ldots, dv_m>}$$

Acknowledgments The authors would like to thank John Hatcliff for his constructive criticism on an earlier draft of this paper.

References

Abelson, H. and Sussman, G.J. 1985. Structure and interpretation of computer programs. MIT Press.

Felleisen, M. and Friedman, D.P. 1989. A syntactic theory of sequential state. *Theoretical Computer Science* 69 (3): pp. 234–287.

Harper, R., Milner, R. and Tofte, M. 1987. The Semantics of Standard ML. LFCS Report Series, LFCS 87 - 36, Edinburgh, UK.

Honsell, F. and Ronchi Della Rocca, S. 1989. Semantica Operazionale di un frammento del linguaggio SCHEME. Università di Torino – Università di Udine, Notes.

IEEE Computer Society 1990. Standard for the Scheme Programming Language. IEEE Std 1178-1990.

Kahn, G. 1987. Natural Semantics. Proceedings of Symposium on Theoretical Aspects of Computer Science. LNCS, Springer-Verlag.

Mason, I. and Talcott, C. 1991. Equivalence in Functional with Effects. *Journal of Functional Programming* 1 (2): pp. 287–328.

Muller, R. 1992. M-LISP: A Representation-Independent Dialect of LISP with Reduction Semantics. *ACM T.o.P.L.a.S.* 14 (4): pp. 589–616.

Plotkin, G. 1981. Structured Operational Semantics. DAIMI report series, Aarhus.

Rees, J. and Clinger, W. (eds) 1986. Revised^3 Report on the algorithmic Language SCHEME.

Smith, B.C. 1994. Reflexion and Semantics in Lisp. In Proceeding of the Eleventh ACM Symposium on Principle on Programming Languages. ACM, New York: pp. 23–35.