CIS 705 - Language Design Project
Assignment 2

Élodie-Jane Sims

The language we study for the project is a simple while-language with Lisp-like expressions for accessing and creating `cons` cells and functions definitions and call.

1 Syntax of the langage

The commands we consider are as follows.

$$
\begin{array}{rl}
C ::= & x := E \\
| & x := y.i \\
| & x.i := E \\
| & x := \texttt{cons}(E_1, E_2) \\
| & \texttt{dispose}(x) \\
| & C_1; C_2 \\
| & if\ \ E\ \ then\ \ C_1\ \ else\ \ C_2 \\
| & \texttt{skip} \\
| & while\ \ E\ \ do\ \ C_1 \\
| & \texttt{fun}\ f\ (v1, ..., v_n)\{\ C\ \} \\
| & \texttt{call}\ f\ (a1, .., a_n)
\end{array}
$$

$$i\ \ ::= 1|2$$

Here, x, v_i, a_i range over variables. Here, f range over function names.
An expression can denote an integer, an atom, or a `cons` cell.

$$
\begin{array}{rll}
E ::= & x & \text{Variable} \\
| & \texttt{nil} & \text{Atom} \\
| & n & \text{Integer} \\
| & E1\ \ +\ \ E_2 & \text{Integers addition} \\
| & E1\ \ -\ \ E_2 & \text{Integers substraction} \\
| & E1\ \ \times\ \ E_2 & \text{Integers multiplication} \\
| & E1\ \ \div\ \ E_2 & \text{Integers division} \\
| & True & \text{Boolean true} \\
| & False & \text{Boolean false} \\
| & E1\ \ ||\ \ E_2 & \text{Boolean disjunction} \\
| & E1\ \ \&\&\ \ E_2 & \text{Boolean conjunction} \\
| & E1\ \ \Rightarrow\ \ E_2 & \text{Boolean implication} \\
| & \neg\ E_2 & \text{Boolean negation} \\
| & E1\ \ =\ \ E_2 & \text{Polymorphique equality}
\end{array}
$$

The second and third assignment statements read and update the heap, respectively. The fourth creates a new `cons` cell in the heap, and places a pointer to it in x.

Notice that in our language we do not handle two dereferencing in a simple statement (no $x.i.j$, no $x.i := y.j$), we assign exactly two cells at each assignment statements; those restrictions are just for simplicity and do not limit the expressivity of the language, we might change this in future or may be just at the implementation step.

2 Typing relation

2.1 Types Syntax

Let $TVAR$ be an infinite set of name for types. We will suppose that we record all the already used one and can always pick a fresh one.

$$
\begin{array}{lll}
BT & := & INT \\
 & & \mid BOOL \\
LT & := & BT\ LIST \\
 & & \mid LT\ LIST \\
 & & \mid LT\ NIL \\
 & & \mid TVAR\ PNIL \\
 & & \mid PAIR(BT, BT) \\
 & & \mid PAIR(BT, LT) \\
 & & \mid PAIR(LT, BT) \\
 & & \mid PAIR(LT, LT) \\
DT & := & BT \\
 & & \mid NIL \\
 & & \mid LT \\
CT & := & UNIT \\
 & & \mid ERROR \\
 & & \mid CT \times CT \\
T & := & DT \\
 & & \mid CT
\end{array}
$$

2.2 Subtyping relation

$$T <: T$$

$$\frac{T_1 <: T_1'}{PAIR(T_1, T_2) <: PAIR(T_1', T_2)}$$

$$\frac{T_2 <: T_2'}{PAIR(T_1, T_2) <: PAIR(T_1, T_2')}$$

$$\frac{T_1 <: T_2}{T_1\ LIST <: T_2\ LIST}$$

$$PAIR(T, T\ NIL) <: T\ LIST$$

$$T\ NIL <: T\ LIST$$

$$TVAR\ PNIL <: T\ NIL$$

$$NIL <: T\ NIL$$

2.3 Typing relation for expressions

Our typing relation is of the form $\Gamma \vdash_E E$ with $\Gamma : Var \rightharpoonup_{fin} Types$.

$$\frac{x \in dom(\Gamma)}{\Gamma \vdash_E x : \Gamma(x)}$$

$$\frac{}{\Gamma \vdash_E \texttt{nil} : NIL}$$

$$\frac{}{\Gamma \vdash_E n : INT}$$

$$\frac{\Gamma \vdash_E E1 : INT \ \ and \ \ \Gamma \vdash_E E2 : INT}{\Gamma \vdash_E E1 \ + \ E_2 : INT}$$

$$\frac{\Gamma \vdash_E E1 : INT \ \ and \ \ \Gamma \vdash_E E2 : INT}{\Gamma \vdash_E E1 \ - \ E_2 : INT}$$

$$\frac{\Gamma \vdash_E E1 : INT \ \ and \ \ \Gamma \vdash_E E2 : INT}{\Gamma \vdash_E E1 \ \times \ E_2 : INT}$$

$$\frac{\Gamma \vdash_E E1 : INT \ \ and \ \ \Gamma \vdash_E E2 : INT}{\Gamma \vdash_E E1 \ \div \ E_2 : INT}$$

$$\frac{}{\Gamma \vdash_E \texttt{true} : Bool}$$

$$\frac{}{\Gamma \vdash_E \texttt{false} : Bool}$$

$$\frac{\Gamma \vdash_E E1 : Bool \ \ and \ \ \Gamma \vdash_E E2 : Bool}{\Gamma \vdash_E E1 \ || \ E_2 : Bool}$$

$$\frac{\Gamma \vdash_E E1 : Bool \ \ and \ \ \Gamma \vdash_E E2 : Bool}{\Gamma \vdash_E E1 \ \&\& \ E_2 : Bool}$$

$$\frac{\Gamma \vdash_E E1 : Bool \ \ and \ \ \Gamma \vdash_E E2 : Bool}{\Gamma \vdash_E E1 \ \Rightarrow \ E_2 : Bool}$$

$$\frac{\Gamma \vdash_E E2 : Bool}{\Gamma \vdash_E \neg E_2 : Bool}$$

$$\frac{\Gamma \vdash_E E1 : T_1 \ \ and \ \ \Gamma \vdash_E E2 : T_2 \ \ (T_1 <: T_2 \ or \ T_2 <: T_1)}{\Gamma \vdash_E E1 \ = \ E_2 : Bool}$$

2.4 Typing relation for commands

Our typing relation is of the form $\Gamma, fn \vdash C, \Gamma', fn'$ with $\Gamma \; : \; Var \rightharpoonup_{fin} Types$, and $fn \; : \; Fun \rightharpoonup_{fin} (\mathcal{P}(Var), Commands)$.

We first define three functions :
A function `untypeNIL` which we will use to untype the `nil` assigned to a stack variable.

$$
\begin{array}{lll}
\texttt{untypeNIL} : & DT & \rightarrow DT \\
& = T' \; NIL & \rightarrow NIL \\
& = V \; PNIL & \rightarrow NIL \\
& \mid \; T' & \rightarrow T'
\end{array}
$$

Let's call `unifyENV` the function which take an environnment Γ, a type variable V and a type T and do the syntactic substitution of any $V \; PNIL$ by T in Γ, we do not write it down here but it's a simple substitution. Let's call `freshNIL` the function which take an environnment Γ and a set of type variables SV to be avoided and provide a type variable which does not accure in Γ neither in SV .

A function `unifyHEAP` which we will use to check weather we can assign some type in the HEAP and if so to provide the unified type. We present it "à la Caml" (meaning that a rule could only be applied if the rules above it can not be apply).

The first argument should be viewed as the type of what we want to put in the heap and the second of the type of what was expected and the environnement is here because the the unification of $PNIL$. $PNIL$ should be viewed as a way to mark in the typing of different variables that this cell is aliassed, so once we assign something in it, we have to update this new information in all the types.

`unifyHEAP` : $(DT \times DT) \times (Var \rightharpoonup_{fin} Types) \rightarrow ((T \cup \texttt{UNDEFINED}) \times (Var \rightharpoonup_{fin} Types))$

$$
\begin{array}{ll}
= (T, T), \Gamma & \rightarrow T, \Gamma \\
\mid\ (NIL, T\ NIL), \Gamma & \rightarrow T\ NIL, \Gamma \\
\mid\ (NIL, V\ PNIL), \Gamma & \rightarrow V\ PNIL, \Gamma \\
\mid\ (NIL, LT\ LIST), \Gamma & \rightarrow LT\ NIL, \Gamma \\
\mid\ (LT\ LIST, LT'\ LIST), \Gamma & \rightarrow let\ (x, e) = \texttt{unifyHEAP}((LT, LT'), \Gamma) \\
 & \quad in \\
 & \quad if\ (x = \texttt{UNDEFINED}) \\
 & \quad then\ (x, e) \\
 & \quad else\ x\ LIST, e \\
\mid\ (LT\ LIST, LT\ NIL), \Gamma & \rightarrow LT\ LIST, \Gamma \\
\mid\ (LT\ LIST, V\ PNIL), \Gamma & \rightarrow LT\ LIST, \texttt{unifyENV}(\Gamma, V, LT\ LIST) \\
\mid\ (LT\ NIL, LT'\ LIST), \Gamma & \rightarrow let\ (x, e) = \texttt{unifyHEAP}((LT, LT'), \Gamma) \\
 & \quad in \\
 & \quad if\ (x = \texttt{UNDEFINED}) \\
 & \quad then\ (x, e) \\
 & \quad else\ x\ NIL, e \\
\mid\ (LT\ NIL, LT'\ NIL), \Gamma & \rightarrow let\ (x, e) = \texttt{unifyHEAP}((LT, LT'), \Gamma) \\
 & \quad in \\
 & \quad if (x = \texttt{UNDEFINED}) \\
 & \quad then\ (x, e) \\
 & \quad else\ (x\ NIL, e) \\
\mid\ (LT\ NIL, V\ PNIL), \Gamma & \rightarrow LT\ NIL, \texttt{unifyENV}(\Gamma, V, LT\ NIL) \\
\mid\ (V\ PNIL, LT\ LIST), \Gamma & \rightarrow LT\ NIL, \Gamma \\
\mid\ (V\ PNIL, LT\ NIL), \Gamma & \rightarrow LT\ NIL, \Gamma \\
\mid\ (V\ PNIL, V'\ PNIL), \Gamma & \rightarrow V'\ NIL, \Gamma \\
\mid\ (PAIR(T_1, T_2), T\ LIST), \Gamma & \rightarrow let\ (x_1, e1) = \texttt{unifyHEAP}((T_1, T), \Gamma)\ and \\
 & \quad (x_2, e2) = \texttt{unifyHEAP}((T_2, T\ LIST), e1) \\
 & \quad in \\
 & \quad if\ ((x_1 = \texttt{UNDEFINED}) \\
 & \quad or\ (x_2 = \texttt{UNDEFINED})) \\
 & \quad then\ (x_2, e_2) \\
 & \quad else\ PAIR(x_1, x_2), e2 \\
\mid\ (PAIR(T_1, T_2), PAIR(T_1', T_2')), \Gamma & \rightarrow let\ (x_1, e1) = \texttt{unifyHEAP}((T_1, T_1'), \Gamma) \\
 & \quad and\ (x_2, e2) = \texttt{unifyHEAP}((T_2, T_2'), e1) \\
 & \quad in \\
 & \quad if\ ((x_1 = \texttt{UNDEFINED}) \\
 & \quad or\ (x_2 = \texttt{UNDEFINED})) \\
 & \quad then\ x_2, e2 \\
 & \quad else\ PAIR(x_1, x_2), e2 \\
\mid\ (\_, \_) & \rightarrow \texttt{UNDEFINED}
\end{array}
$$

A function `typeCONS` which we will use to provide a type when allocating a cell by `cons` in the environnement Γ. We present it “à la Caml”.

$$
\begin{array}{lll}
\texttt{typeCONS} : & (T \times T) \times (Var \rightharpoonup_{fin} Types) & \rightarrow (DT \cup \texttt{UNDEFINED}) \\
= & (NIL, NIL), \Gamma & \rightarrow let\ v1 = \texttt{freshNIL}(\Gamma, \emptyset) \\
 & & \quad and\ v2 = \texttt{freshNIL}(\Gamma, \{v1\}) in \\
 & & \quad PAIR(v1\ NIL, v2\ NIL) \\
\mid & (NIL, DT), \Gamma & \rightarrow PAIR(\texttt{freshNIL}(\Gamma, \emptyset), DT) \\
\mid & (DT, NIL), \Gamma & \rightarrow PAIR(DT, \texttt{freshNIL}(\Gamma, \emptyset)) \\
\mid & (DT, DT'), \Gamma & \rightarrow PAIR(DT, DT')
\end{array}
$$

$$
\frac{\Gamma \vdash_E E : T}{\Gamma, fn \vdash x := E : UNIT, [\Gamma \mid x \mapsto \texttt{untypeNIL}(T)], fn}
$$

$$
\frac{\Gamma \vdash_E y : PAIR(T_1, T_2)}{\Gamma, fn \vdash x := y.i : UNIT, [\Gamma \mid x \mapsto \texttt{untypeNIL}(T_i)], fn}
$$

$$
\frac{\Gamma \vdash_E y : T\ LIST}{\Gamma, fn \vdash x := y.1 : UNIT \times ERROR, [\Gamma \mid x \mapsto \texttt{untypeNIL}(T)], fn}
$$

$$
\frac{\Gamma \vdash_E y : T\ LIST}{\Gamma, fn \vdash x := y.2 : UNIT \times ERROR, [\Gamma \mid x \mapsto T\ LIST], fn}
$$

$$
\frac{\begin{array}{c}\Gamma \vdash_E E : T \\ \Gamma \vdash_E x : PAIR(T_1, T_2) \\ (T', \Gamma') = \texttt{unifyHEAP}(T, T_1) \\ T' \neq \texttt{UNDEFINED}\end{array}}{\Gamma, fn \vdash x.1 := E : UNIT, [\Gamma' \mid x \mapsto PAIR(T', T_2)], fn}
$$

$$
\frac{\begin{array}{c}\Gamma \vdash_E E : T \\ \Gamma \vdash_E x : PAIR(T_1, T_2) \\ (T', \Gamma') = \texttt{unifyHEAP}(T, T_1) \\ T' \neq \texttt{UNDEFINED}\end{array}}{\Gamma, fn \vdash x.2 := E : UNIT, [\Gamma' \mid x \mapsto PAIR(T_1, T')], fn}
$$

$$
\frac{\begin{array}{c}\Gamma \vdash_E E : T' \\ \Gamma \vdash_E x : T\ LIST \\ (T'', \Gamma'') = \texttt{unifyHEAP}(T', T) \\ T'' \neq \texttt{UNDEFINED}\end{array}}{\Gamma, fn \vdash x.1 := E : UNIT \times ERROR, [\Gamma'' \mid x \mapsto PAIR(T'', T\ LIST)], fn}
$$

$$
\frac{\begin{array}{c}\Gamma \vdash_E E : T' \\ \Gamma \vdash_E x : T\ LIST \\ (T'', \Gamma'') = \texttt{unifyHEAP}(T', T\ LIST) \\ T'' \neq \texttt{UNDEFINED}\end{array}}{\Gamma, fn \vdash x.2 := E : UNIT \times ERROR, [\Gamma'' \mid x \mapsto PAIR(T, T'')], fn}
$$

$$
\frac{\Gamma \vdash_E E_i : T_i}{\Gamma, fn \vdash x := \texttt{cons}(E_1, E_2) : UNIT, [\Gamma \mid x \mapsto \texttt{typeCONS}((T_1, T_2), \Gamma)], fn}
$$

$$
\frac{\begin{array}{c}\Gamma, fn \vdash C_1 : UNIT \times ERROR, \Gamma_1, fn_1 \\ \Gamma_1, fn_1 \vdash C_2 : UNIT \times ERROR, \Gamma_2, fn_2\end{array}}{\Gamma, fn \vdash C_1; C_2 : UNIT \times ERROR, \Gamma_2, fn_2}
$$

$$\frac{\begin{array}{c}\Gamma, fn \vdash C_1 \ : UNIT \times ERROR, \Gamma_1, fn_1 \\ \Gamma_1, fn_1 \vdash C_2 \ : UNIT, \Gamma_2, fn_2\end{array}}{\Gamma, fn \vdash C_1; C_2 : UNIT \times ERROR, \Gamma_2, fn_2}$$

$$\frac{\begin{array}{c}\Gamma, fn \vdash C_1 \ : UNIT, \Gamma_1, fn_1 \\ \Gamma_1, fn_1 \vdash C_2 \ : UNIT, \Gamma_2, fn_2\end{array}}{\Gamma, fn \vdash C_1; C_2 : UNIT, \Gamma_2, fn_2}$$

We define $\Gamma <: \Gamma'$ iff

- $dom(\Gamma') \subseteq dom(\Gamma)$
- if $x \in dom(\Gamma')$ then $\Gamma(x) <: \Gamma'(x)$

$$\frac{\begin{array}{c}\Gamma \vdash_E E : Bool \\ \Gamma, fn \vdash C_1 \ : UNIT \times ERROR, \Gamma_1, fn_1 \\ \Gamma, fn \vdash C_2 \ : UNIT \times ERROR, \Gamma_2, fn_2 \\ \Gamma_1 <: \Gamma' \\ \Gamma_2 <: \Gamma'\end{array}}{\Gamma, fn \vdash if \ \ E \ \ then \ \ C_1 \ \ else \ \ C_2 : UNIT \times ERROR, \Gamma', fn_1 \cap fn_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash_E E : Bool \\ \Gamma, fn \vdash C_1 \ : UNIT \times ERROR, \Gamma_1, fn_1 \\ \Gamma, fn \vdash C_2 \ : UNIT, \Gamma_2, fn_2 \\ \Gamma_1 <: \Gamma' \\ \Gamma_2 <: \Gamma'\end{array}}{\Gamma, fn \vdash if \ \ E \ \ then \ \ C_1 \ \ else \ \ C_2 : UNIT, \Gamma', fn_1 \cap fn_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash_E E : Bool \\ \Gamma, fn \vdash C_1 \ : UNIT, \Gamma_1, fn_1 \\ \Gamma, fn \vdash C_2 \ : UNIT \times ERROR, \Gamma_2, fn_2 \\ \Gamma_1 <: \Gamma' \\ \Gamma_2 <: \Gamma'\end{array}}{\Gamma, fn \vdash if \ \ E \ \ then \ \ C_1 \ \ else \ \ C_2 : UNIT \times ERROR, \Gamma', fn_1 \cap fn_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash_E E : Bool \\ \Gamma, fn \vdash C_1 \ : UNIT, \Gamma_1, fn_1 \\ \Gamma, fn \vdash C_2 \ : UNIT, \Gamma_2, fn_2 \\ \Gamma_1 <: \Gamma' \\ \Gamma_2 <: \Gamma'\end{array}}{\Gamma, fn \vdash if \ \ E \ \ then \ \ C_1 \ \ else \ \ C_2 : UNIT, \Gamma', fn_1 \cap fn_2}$$

In fact in the above rules we do not exactly find a Γ' by $<:$ on environnements but we have to do some unifications while uncoutering some $PNIL$.

We define $fn_1 \cap fn_2$ as :

– $dom(fn_1 \cap fn_2) \subseteq dom(fn_1) \cap dom(fn_2)$

– if $\begin{pmatrix} fn_1(f) = \langle (v_1, ..., v_n), Cf_1 \rangle \\ fn_2(f) = \langle (w_1, ..., w_m), Cf_2 \rangle \\ n = m \end{pmatrix}$

then $\begin{pmatrix} (fn_1 \cap fn_2)(f) = \langle (u_1, ..., u_n), \\ if\ \texttt{true}\ then\ Cf_1\{u_i/v_i\}\ else\ Cf_2\{u_i/w_i\} \rangle \end{pmatrix}$

with u_i variables which do not occur in Cf_1 and Cf_2.

$$\frac{}{\Gamma, fn \vdash \texttt{skip} : UNIT, \Gamma, fn}$$

$$\frac{\begin{array}{c} \Gamma \vdash_E E : Bool \\ \Gamma' \vdash_E E : Bool \\ \exists \Gamma_n \begin{pmatrix} \Gamma <: \Gamma' \\ \Gamma_n <: \Gamma' \\ \Gamma', fn \vdash C : T, \Gamma_n, fn' \end{pmatrix} \end{array}}{\Gamma, fn \vdash while\ \ E\ \ do\ \ C : T, \Gamma', fn'}$$

$$\frac{}{\Gamma, fn \vdash \texttt{fun}\ f\ (v1, ..., v_n)\{\ C\ \} : UNIT, \Gamma, [fn | f \mapsto ((v1, ..., v_n), C)]}$$

$$\frac{fn(f) = ((v1, .., v_n), C)\ \ and\ \ \Gamma, fn \vdash C\{a_i/v_i\} : T, \Gamma', fn'}{\Gamma, fn \vdash \texttt{call}\ f\ (a1, ..., a_n) : T, \Gamma', fn'}$$

3 Evaluation relation

$$\begin{aligned} Val &= Int \cup \{nil, true, false\} \cup Loc \\ S &= Var \rightharpoonup_{fin} Val \\ H &= Loc \rightharpoonup_{fin} Val \times Val \\ F &= Fun \rightharpoonup_{fin} (\mathcal{P} \qquad\qquad\qquad (Var), Commands)) \end{aligned}$$

Here, $Loc = \{l, ...\}$ is an infinite set of locations, $Var = \{x, y, ...\}$ is a set of variables, $Atoms = \{\texttt{nil}, \texttt{true}, \texttt{false}, a, ...\}$ is a set of atoms, $Fun = \{f, g, ...\}$ is a set of function's variables, and $\rightharpoonup_{fin}$ is for finite partial functions. We call an element $s \in S$ a stack, and $h \in H$ a heap.

We use $dom(h)$ to denote the domain of definition of a heap $h \in H$, and $dom(s)$ to denote the domain of a stack $s \in S$.

An expression is interpreted as a heap-independent value

$$[\![E]\!]s \in Val$$

where $dom(s)$ includes the free variables of E.

$$
\begin{aligned}
[\![x]\!]s &= s(x)\\
[\![\texttt{nil}]\!]s &= nil\\
[\![n]\!]s &= n\\
[\![E_1 + E_2]\!]s &= [\![E_1]\!]s + [\![E_2]\!]s\\
[\![E_1 - E_2]\!]s &= [\![E_1]\!]s - [\![E_2]\!]s\\
[\![E_1 \times E_2]\!]s &= [\![E_1]\!]s \times [\![E_2]\!]s\\
[\![E_1 \div E_2]\!]s &= [\![E_1]\!]s \div [\![E_2]\!]s\\
[\![\texttt{true}]\!]s &= true\\
[\![\texttt{false}]\!]s &= false\\
[\![E_1 \mid\mid E_2]\!]s &= [\![E_1]\!]s \vee [\![E_2]\!]s\\
[\![E_1 \&\& E_2]\!]s &= [\![E_1]\!]s \wedge [\![E_2]\!]s\\
[\![E_1 \Rightarrow E_2]\!]s &= [\![E_1]\!]s \Rightarrow [\![E_2]\!]s\\
[\![\neg E_2]\!]s &= \neg [\![E_1]\!]s\\
[\![E_1 = E_2]\!]s &= [\![E_1]\!]s = [\![E_2]\!]s
\end{aligned}
$$

The functions $+, -, \times, \div$ are the usual functions on integers and $\vee, \wedge, \Rightarrow, \neg$ the usual functions on booleans and $=$ is the usual polymorphique equality function. We not add an error value so $[\![E]\!]$ is not always defined.

The semantics of our statements are given in a small-step semantic defined by the relation $\rightsquigarrow$ on configurations. The configurations include quadruples C, fn, s, h and terminal configurations fn, s, h for $s \in S$ and $h \in H$ and $fn \in F$.

In the following rules we use r to range over elements of $Val \times Val$, $\pi_i r$ for the first or second projection, and $(r|i \mapsto v)$ to indicate the pair like r except that the i'th component is replaced with v.

If we write a condition for example $h(s(y)) = r$ it means that $y \in dom(s)$ and $s(y) \in dom(h)$.

$$\frac{[\![E]\!]s = v}{x := E, fn, s, h \rightsquigarrow fn, [s|x \mapsto v], h}$$

$$\frac{h(s(y)) = r}{x := y.i, fn, s, h \rightsquigarrow fn, [s|x \mapsto \pi_i r], h}$$

$$\frac{h(s(x)) = r \quad [\![E]\!]s = v}{x.i {=} E, fn, s, h \rightsquigarrow fn, s, [h|s(x) \mapsto (r|i \mapsto v)]}$$

$$\frac{s(x) \in dom(h)}{\texttt{dispose}(x), fn, s, h \rightsquigarrow fn, s, (h - s(x))}$$

$$\frac{l \in Loc \quad l \notin dom(h) \quad [\![E_1]\!]s = v_1, [\![E_2]\!]s = v_2}{x := \texttt{cons}(E_1, E_2), fn, s, h \rightsquigarrow fn, [s|x \mapsto l], [h|l \mapsto \langle v_1, v_2 \rangle]}$$

$$\frac{C_1, fn, s, h \rightsquigarrow C', fn', s', h'}{C_1; C_2, fn, s, h \rightsquigarrow C'; C_2, fn', s', h'}$$

$$\frac{C_1, fn, s, h \rightsquigarrow fn', s', h'}{C_1; C_2, fn, s, h \rightsquigarrow C_2, fn', s', h'}$$

$$\frac{[\![E]\!]s = True}{if \;\; E \;\; then \;\; C_1 \;\; else \;\; C_2, fn, s, h \rightsquigarrow C_1, fn, s, h}$$

$$\frac{[\![E]\!]s = False}{if \;\; E \;\; then \;\; C_1 \;\; else \;\; C_2, fn, s, h \rightsquigarrow C_2, fn, s, h}$$

$$\frac{}{\texttt{skip}, fn, s, h \rightsquigarrow fn, s, h}$$

$$\frac{[\![E]\!]s = True}{while \;\; E \;\; do \;\; C, fn, s, h \rightsquigarrow C; while \;\; E \;\; do \;\; C, fn, s, h}$$

$$\frac{[\![E]\!]s = False}{while \;\; E \;\; do \;\; C, fn, s, h \rightsquigarrow fn, s, h}$$

$$\frac{}{\texttt{fun}\, f\ (v1, ..., v_n)\{\ C\ \}, fn, s, h \rightsquigarrow [fn|f \mapsto ((v1, ..., v_n), C)], s, h}$$

$$\frac{fn(f) = ((v1, .., v_n), C)}{\texttt{call}\, f\ (a1, ..., a_n), fn, s, h \rightsquigarrow C\{a_i/v_i\}, fn, s, h}$$

The location l in the fifth case is not specified uniquely, so a new location is chosen non-deterministically.

4 Theorems

We would like to prove some theorem like :

Theorem 1. *If $\emptyset, fn \vdash C : UNIT, \Gamma', fn'$ then the evaluation of $C, fn, \emptyset, []$ does not get stuck or if it does it is because of a division by 0. Also if it evaluate, the types of the data in a heap cell does not change during the run of the program (if a boolean, or an integer, or an pointer is allocated then it stays like that).*

We would need for this to change or evaluation relation to add some different kinds of errors.

We could also want to try to define some link between Γ and s, h and then prove a theorem with $\Gamma, fn \vdash C, \Gamma', fn$.

Notice that we did not make the $\vdash$ compositionnal which is we did not express a typing for functions and calls but we could try to do something in the implementation to reuse the typing of function which will consist in finding what are the minimal requirement for the typing of arguments and global variable and then what is the result of typing.

We also did not express what would be a procedure to apply the rule for typing a *while* loop and typing an if, we would take the smallest Γ' we could compute in both cases.

We also could improve our analysis in the case of $if\ then\ else$ to got some union types and then we would be able to say that if it does not type, we know for sure there is an error, if it does type $UNIT$ we know there is no error, if it does type $UNIT \times ERROR$ then we don't know. With this it will also allow us to point out the undeclared variables, which is what we can't track now since an if now can remove any information about a variable while this variable exists.

We could improve our analysing by treating the conditions E in the if and $while$ differently if the are of the form $x = E$ or $\neg(x = E)$ and treat them if there were some assignments.

We would like also to change our typing to add the integers values and the boolean values to be themselves types (subtypes of INT and $BOOL$) so that in some case we can keep some precision which could be usefull to detect that there are no divison error or even to find the result of the tests.