CIS 705 - Language Design Project
Assignment 1

Élodie-Jane Sims

The language I plan to 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 Command syntax

The commands we consider are as follows.

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

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

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.

$E ::=$	x	Variable
$\mid$	42	Integer
$\mid$	$\texttt{nil}$	Atom
$\mid$	a	Atom
$\mid$	$True$	Atom
$\mid$	$False$	Atm
$\mid$	$E1\;\; op_a\;\; E_2$	Arithmetic operation
$\mid$	$E1\;\; op_b\;\; E_2$	Boolean operation
$\mid$	$op_b\;\; E_2$	Boolean operation

The precise language for the expressions is not chosen yet.

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 Semantic domains

$$\begin{aligned} Val &= Int \cup Atoms \cup Loc \\ S &= Var \rightharpoonup_{fin} Val \\ H &= Loc \rightharpoonup_{fin} Val \times Val \end{aligned}$$

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, 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.

$$\begin{aligned} [\![x]\!]s &= s(x) \\ [\![42]\!]s &= 42 \\ [\![\texttt{true}]\!]s &= true \\ [\![E_1 + E_2]\!]s &= [\![E_1]\!]s + [\![E_2]\!]s \\ &\ldots \end{aligned}$$

$$F = Fun \rightharpoonup_{fin} (\mathcal{P}\ (Var), Commands))$$

Here, $Fun = \{f, g, ...\}$.

3 Small-step semantics

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.

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

$$\frac{[\![E]\!]s = l \in Loc \ \ h(l) = r}{x := E.i, fn, s, h \rightsquigarrow fn, [s|x \mapsto \pi_i r], h}$$

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

$$\frac{l \in Loc \quad l \in dom(h) \quad [\![E]\!]s = l}{\texttt{dispose}(E), fn, s, h \rightsquigarrow fn, s, (h - l)}$$

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

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

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

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

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

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

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

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

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

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

The location l in the fifth case is not specified uniquely, so a new location is chosen non-deterministically.
Let the set of error configurations be :

$$\Omega = \{C, fn, s, h \mid \; \nexists K.\; C, fn, s, h \rightsquigarrow K\}$$

We say that :

- "C, fn, s, h is safe" iff $\forall K.\; (C, fn, s, h \rightsquigarrow^* K \;\Rightarrow\; K \notin \Omega)$

For instance, an error state can exist by an attempt to dereference `nil` or an integer. Note also that the semantics allows dangling references, as in stack $[x \mapsto l]$ with empty heap $[]$.

4 Examples

4.1 Copy a non-cyclic list pointed by x and make y points to it

```
if ( x != nil)
then
      v1 := x.1;
      nc := cons (v1,nil);
      y := nc;
      cy := nc;
      cx := x.2;
      while (cx != nil) do
            (v1 := cx.1;
             nc := cons (v1, nil);
             cy.2 := nc;
             cy := nc;
             cx := cx.2)
else
     y := nil
```

4.2 Copy a non-cyclic list pointed by x and make y points to it with functions

```
if ( x != nil)
then
      fun copy-cons (c1,c2,c3,c4)
      {
        v1 := c1.1;
        c2 := cons (v1, nil);
        c3 := c2;
        c4 := c1.2
      }
      ;
      call copy-cons (x,nc,cy,cx);
      y := nc;
      while (cx != nil) do
             (call copy-cons (cx,nc,cy,cx);
              cy.2 := nc)
else
     y := nil
```