The Imperative imPL Language

YSC3208: Programming Language Implementation

Răzvan Voicu

Week 9, March 13-17

Introduction

- simPL, dPL: an identifier refers to a value
- once computed, the value does not change
- pass-by-need exploits this fact
- referential transparency
- good for formal reasoning

Motivation

- many algorithms are more efficient when presented using random-access memory
- let us support a new class of mutable values that are stored in memory locations that can be updated.
- assignment can change the value stored in the memory location associated with each identifier of a mutable value
- imPL0 allows assignment to mutable values
- imPL1 allows assignment to mutable fields of data constructor
- many interesting variants...

Syntax of imPL0

Assignment
$$\frac{E}{x \ := \ E}$$

Sequence
$$\frac{E_1 \quad E_2}{E_1 \ ; \ E_2}$$

Loop
$$\frac{E_1 \quad E_2}{\texttt{while}\ E_1\ \texttt{do}\ E_2\ \texttt{end}}$$

Example

Add a qualifier for mutable values. Without qualifier, the values are assumed to be immutable.

```
let mut x = 0
        y = 3
in
   x := 1;
   x := x + 2;
   x := x + y;
   x
end
```

Another Example

```
fun x ->
   let mut i = 1
       mut f = 1 in
       while i <= x do
          f := f * i;
          i := i + 1
       end;
       f
   end
end
```

Yet Another Example

```
let gcd = fun mut a mut b ->
            while (a = b) do
               if a > b
               then a := a - b
               else b := b - a
               end
            end;
            a
         end
    mut c = 6
    mut d = 10
in
   (gcd c d)
end
```

Denotational Semantics: Then How?

```
let mut x = 0
    y = 3
in
   x := 1;
   x := x + 2;
   x := x + y;
   x
end
```

Denotational Semantics: Idea

- mutable identifiers refer to locations
- a store maps locations to values
- the store is passed to and returned from the semantic function

Semantic Domains

Domain name	Definition
EV	$\textbf{Int} + \textbf{Bool} + \textbf{Fun} + \bot(\textbf{Int})$
SV	$\textbf{Int} + \textbf{Bool} + \textbf{Fun}$
DV	$\textbf{SV} + \textbf{Loc}$
Fun	$\textbf{DV} * \cdots * \textbf{DV} * \textbf{Store} \leadsto (\textbf{EV}, \textbf{Store})$
Store	$\textbf{Loc} \leadsto \textbf{SV}$
Env	$\textbf{Id} \leadsto \textbf{DV}$

Example

Let us say we have a store with the value 1 at location l

$$\Sigma = \emptyset_{\mathbf{Store}}[l \leftarrow 1]$$

and an environment that carries location l at identifier x

$$\Delta = \emptyset_{\mathbf{Env}}[x \leftarrow l]$$

Then we can access the value of x in the store as follows:

$$\Sigma(\Delta(x)) = 1$$

The Main Semantic Function

$$\cdot \rightarrowtail \cdot : \textbf{imPL0} \rightarrow \textbf{EV}$$

$$\frac{\emptyset_{\textbf{Store}} \mid \emptyset_{\textbf{Env}} \Vdash E \rightarrowtail (v, \Sigma)}{E \rightarrowtail v}$$

$$\cdot \mid \cdot \Vdash \cdot \rightarrowtail \cdot : \textbf{Store} * \textbf{Env} * \textbf{imPL0} \rightarrow \textbf{EV} * \textbf{Store}$$

Let Expressions (by value)

$$\frac{\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v_1, \Sigma') \qquad \Sigma' \mid \Delta[x_1 \leftarrow v_1] \Vdash E \rightarrowtail (v, \Sigma'')}{\Sigma \mid \Delta \Vdash \texttt{let } x_1 = E_1 \texttt{ in } E \texttt{ end} \rightarrowtail (v, \Sigma'')}$$

Mutable Let Expressions

$$\frac{\textit{fresh } l_1 \qquad \Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v_1, \Sigma') \qquad \Sigma'[l_1 \leftarrow v_1] \mid \Delta[x_1 \leftarrow l_1] \Vdash E \rightarrowtail (v, \Sigma'')}{\Sigma \mid \Delta \Vdash \texttt{let}\ \textit{mut}\ x_1 = E_1\ \texttt{in}\ E\ \texttt{end} \rightarrowtail (v, \Sigma'')}$$

since l_1 is a new location,
which means $\Sigma'(l_1)$ is undefined

Identifiers of Immutable Values

$$\frac{\Delta(x) \notin dom(\Sigma)}{\Sigma \mid \Delta \Vdash x \rightarrowtail (\Delta(x), \Sigma)}$$

Identifiers of Mutable Values

$$\frac{\Delta(x) \in dom(\Sigma)}{\Sigma \mid \Delta \Vdash x \rightarrowtail (\Sigma(\Delta(x)), \Sigma)}$$

Assignment

$$\frac{\Delta(x) \in dom(\Sigma) \qquad \Sigma \mid \Delta \Vdash E \rightarrowtail (v, \Sigma')}{\Sigma \mid \Delta \Vdash x \; \texttt{:=} \; E \; \rightarrowtail (v, \Sigma'[\Delta(x) \leftarrow v])}$$

Example

$\emptyset_{\textbf{Store}}[l \leftarrow 1] \mid \emptyset_{\textbf{Env}}[\mathtt{a} \leftarrow l] \Vdash$
$\mathtt{a\ :=\ 2} \rightarrowtail (2, \emptyset_{\textbf{Store}}[l \leftarrow 1][l \leftarrow 2])$
The resulting store $\emptyset_{\textbf{Store}}[l \leftarrow 1][l \leftarrow 2])$ is of course the same as $\emptyset_{\textbf{Store}}[l \leftarrow 2]$.
The original binding of l to 1 is overwritten by the new value 2.

Function Definition

$$\frac{}{\Sigma \mid \Delta \Vdash \texttt{fun } x \texttt{ -> } E \texttt{ end} \rightarrowtail (f, \Sigma)}$$

where $f(v, \Sigma') = (v', \Sigma'')$,
where $\Sigma' \mid \Delta[x \leftarrow v] \Vdash E \rightarrowtail (v', \Sigma'')$

Note that v may either be a value or a mutable location.

Sequence

$$\frac{\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v_1, \Sigma') \qquad \Sigma' \mid \Delta \Vdash E_2 \rightarrowtail (v_2, \Sigma'')}{\Sigma \mid \Delta \Vdash E_1 ; E_2 \rightarrowtail (v_2, \Sigma'')}$$

While Loop

$$\frac{\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (\mathit{false}, \Sigma')}{\Sigma \mid \Delta \Vdash \texttt{while } E_1 \texttt{ do } E_2 \texttt{ end} \rightarrowtail (\mathit{true}, \Sigma')}$$

While Loop

$$\frac{\begin{array}{c}\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (\mathit{true}, \Sigma') \\ \Sigma' \mid \Delta \Vdash E_2 \rightarrowtail (v_2, \Sigma'') \\ \Sigma'' \mid \Delta \Vdash \texttt{while } E_1 \texttt{ do } E_2 \texttt{ end} \rightarrowtail (v, \Sigma''')\end{array}}{\Sigma \mid \Delta \Vdash \texttt{while } E_1 \texttt{ do } E_2 \texttt{ end} \rightarrowtail (v, \Sigma''')}$$

Declaring Constructor with Mutable Field

Let us declare a pair type with a mutable field.

```
type pair 'a 'b = Pair 'a (mut 'b)
```

Pattern Matching on Mutable Field

Mutable fields may be updated.

```
let p = Pair 1 2
in match p with
     Pair x y -> y:=y+1
   end
end
```

Semantic Domains

Domain name	Definition
EV	**Int** + **Bool** + **Fun** + **Dat** + $\{\perp\}$
SV	**Int** + **Bool** + **Fun** + **Dat**
DV	**SV** + **Loc**
Dat	**c DV** $\cdots$ **DV**

Rules for imPL1

$$\frac{\begin{array}{c}\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v_1, \Sigma_1) \quad \cdots \quad \Sigma_{n-1} \mid \Delta \Vdash E_n \rightarrowtail (v_n, \Sigma_n) \\ C \;\; t_1 \ldots t_n \in t \qquad \Sigma_n, [v_1 : t_1; ..; v_n : t_n] \;\Rightarrow\; \Sigma', [w_1; ..; w_n]\end{array}}{\Sigma \mid \Delta \Vdash (C \;\; E_1 \ldots E_n) : t \rightarrowtail (C \;\; w_1 \ldots w_n, \Sigma')}$$

Rules for imPL1

$$\frac{immutable(t) \qquad \Sigma, rest \;\Rightarrow\; \Sigma', rest'}{\Sigma, (v : t) :: rest \;\Rightarrow\; \Sigma', v :: rest'}$$

$$\frac{mutable(t) \qquad fresh\ l \qquad \Sigma, rest \;\Rightarrow\; \Sigma', rest'}{\Sigma, (v : t) :: rest \;\Rightarrow\; \Sigma'[l \rightarrow v], l :: rest'}$$

Pass by value

$$\frac{imm(param(f)) \qquad \Sigma \mid \Delta \Vdash E_1 \rightarrowtail (f, \Sigma') \qquad \Sigma' \mid \Delta \Vdash E_2 \rightarrowtail (v_2, \Sigma'')}{\Sigma \mid \Delta \Vdash (E_1\ E_2) \rightarrowtail f(v_2, \Sigma'')}$$

Pass-by-Reference

$$\frac{mut(param(f)) \qquad \Delta(x) \in dom(\Sigma) \qquad \Sigma \mid \Delta \Vdash E_1 \rightarrowtail (f, \Sigma')}{\Sigma \mid \Delta \Vdash (E_1\ x) \rightarrowtail f(\Delta(x), \Sigma')}$$

Pass-by-Reference

$$\frac{mut(param(f)) \qquad \Sigma \mid \Delta \Vdash E_1 \rightarrowtail (f, \Sigma') \qquad \Sigma' \mid \Delta \Vdash E_2 \rightarrowtail (v_2, \Sigma'')}{\Sigma \mid \Delta \Vdash (E_1\ E_2) \rightarrowtail f(l, \Sigma''[l \leftarrow v_2])}$$

if E_2 is not an identifier,
where l is a new location in Σ''.

1. imPL0
2. imPL1
3. Pass-by-value, Pass-by-reference
4. Imperative Programming and Exception Handling
   - Standard Semantics
   - Alternative Semantics
5. A Virtual Machine for imPL

Division by Zero

$$\frac{\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v_1, \Sigma') \qquad v_1 \notin \textbf{Exc} \qquad \Sigma' \mid \Delta \Vdash E_2 \rightarrowtail (0, \Sigma'')}{\Sigma \mid \Delta \Vdash E_1/E_2 \rightarrowtail (\bot, \Sigma'')}$$

Execution of try-expressions

$$\frac{\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v, \Sigma') \qquad v \notin \textbf{Exc}}{\Sigma \mid \Delta \Vdash \texttt{try}\ E_1\ \texttt{catch}\ x\ \texttt{with}\ E_2\ \texttt{end} \rightarrowtail (v, \Sigma')}$$

$$\frac{\begin{array}{c}\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v_1, \Sigma') \qquad v_1 \in \textbf{Exc} \\ \Sigma' \mid \Delta[x \leftarrow v_1] \Vdash E_2 \rightarrowtail (v_2, \Sigma'')\end{array}}{\Sigma \mid \Delta \Vdash \texttt{try}\ E_1\ \texttt{catch}\ x\ \texttt{with}\ E_2\ \texttt{end} \rightarrowtail (v_2, \Sigma'')}$$

Rolling back the State

$$\frac{\Sigma \mid \Delta \Vdash E_1 \rightarrowtail (v_1, \Sigma') \qquad v_1 \in \textbf{Exc} \qquad \Sigma \mid \Delta[x \leftarrow v_1] \Vdash E_2 \rightarrowtail (v_2, \Sigma'')}{\Sigma \mid \Delta \Vdash \texttt{try } E_1 \texttt{ catch } x \texttt{ with } E_2 \texttt{ end} \rightarrowtail (v_2, \Sigma'')}$$

Idea

Use a mutable heap for implementing imperative constructs

Translation of Assignment

$$\frac{E \hookrightarrow s}{x \; \texttt{:=} \; E \hookrightarrow s.\texttt{ASSIGNS} \; x}$$

Execution of Assignment

$$\frac{s(pc) = \texttt{ASSIGNS}\ x}{(v.os, pc, e, rs, h) \rightrightarrows_s (os, pc+1, e, rs, h[e[x] \leftarrow v])}$$

Translation of Sequences

$$\frac{E_1 \hookrightarrow s_1 \qquad E_2 \hookrightarrow s_2}{E_1\,;E_2 \hookrightarrow s_1.\mathtt{POP}.s_2}$$

Implementation of POP

$$\frac{s(pc) = \texttt{POP}}{(v.os, pc, e, rs, h) \rightrightarrows_s (os, pc+1, e, rs, h)}$$

Translation of Loops

$$\frac{E_1 \hookrightarrow s_1 \qquad E_2 \hookrightarrow s_2}{\begin{array}{c}\texttt{while } E_1 \texttt{ do } E_2 \\ \hookrightarrow \\ s_1.(\texttt{JOFR } |s_2 + 3|).s_2.\texttt{POP}. \\ (\texttt{GOTOR } -(|s_1| + 2 + |s_2|)).LDCB \texttt{ true}\end{array}}$$

Translation of Loops with Labels

$$\frac{E_1 \hookrightarrow s_1 \qquad E_2 \hookrightarrow s_2 \qquad \textit{fresh } L0, L1}{\begin{array}{c}\texttt{while } E_1 \texttt{ do } E_2 \\ \hookrightarrow \\ \texttt{LABEL } l_0 : s_1.(\texttt{JOF } l_1).s_2.\texttt{POP}. \\ (\texttt{GOTO } l_0.\texttt{LABEL } l_1 : \mathit{LDCB} \texttt{ true}\end{array}}$$