

09—The Language imPL

CS 4215: Programming Language Implementation

Martin Henz

March 16, 2012

Generated on Thursday 15 March, 2012, 22:56

Introduction

- simPL, rePL: 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 natural when presented using random-access memory
- assignment allows to change the value stored in the memory location associated with an identifier
- imPL0 allows assignment
- imPL1 adds assignment to record properties
- many interesting variants...

- 1 imPL0
 - Syntax
 - Examples
 - Denotational Semantics
- 2 imPL1
- 3 Pass-by-reference, Pass-by-copy
- 4 Imperative Programming and Exception Handling
- 5 A Virtual Machine for imPL

Syntax of imPL0

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

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

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

Example

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

Another Example

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

Yet Another Example

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


Denotational Semantics: Then How?

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

Denotational Semantics: Idea

- 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	Int + Bool + Fun + {error}
SV	Int + Bool + Fun
DV	Loc
Fun	DV * ... * DV * Store \rightsquigarrow (EV, Store)
Store	Loc \rightsquigarrow SV
Env	Id \rightsquigarrow 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

$$\cdot \rightsquigarrow \cdot : \mathbf{imPL0} \rightarrow \mathbf{EV}$$

$$\emptyset_{\mathbf{Store}} \mid \emptyset_{\mathbf{Env}} \Vdash E \rightsquigarrow (v, \Sigma)$$

$$E \rightsquigarrow v$$

$$\cdot \mid \cdot \Vdash \cdot \rightsquigarrow \cdot : \mathbf{Store} * \mathbf{Env} * \mathbf{imPL0} \rightarrow \mathbf{EV} * \mathbf{Store}$$

Let Expressions

$$\Sigma'[l_1 \leftarrow v_1] \mid \Delta[x_1 \leftarrow l_1] \Vdash E \rightsquigarrow (v, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash \text{let } x_1 = E_1 \text{ in } E \text{ end} \rightsquigarrow (v, \Sigma'')$$

if $\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma')$, and

l_1 is a new location,

which means $\Sigma'(l_1)$ is undefined

Identifiers

$$\Sigma \mid \Delta \Vdash x \rightsquigarrow (\Sigma(\Delta(x)), \Sigma)$$

Assignment

$$\Sigma \mid \Delta \Vdash E \rightsquigarrow (v, \Sigma')$$

$$\Sigma \mid \Delta \Vdash x := E \rightsquigarrow (v, \Sigma'[\Delta(x) \leftarrow v])$$

Example

$\emptyset\text{Store}[l \leftarrow 1] \mid \emptyset\text{Env}[a \leftarrow l] \Vdash$

$a := 2 \rightsquigarrow (2, \emptyset\text{Store}[l \leftarrow 1][l \leftarrow 2])$

The resulting store $\emptyset\text{Store}[l \leftarrow 1][l \leftarrow 2]$ is of course the same as $\emptyset\text{Store}[l \leftarrow 2]$.

The original binding of l to 1 is overwritten by the new value 2.

Function Definition

$$\frac{}{\Sigma \mid \Delta \Vdash \text{fun } x \rightarrow E \text{ end} \rightsquigarrow (f, \Sigma)}$$

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

Function Application

$$\Sigma \mid \Delta \Vdash E_1 \mapsto (f, \Sigma') \quad \Sigma' \mid \Delta \Vdash E_2 \mapsto (v_2, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash (E_1 E_2) \mapsto f(l, \Sigma''[l \leftarrow v_2])$$

where l is a new location in Σ'' .

Sequence

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma') \quad \Sigma' \mid \Delta \Vdash E_2 \rightsquigarrow (v_2, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash E_1 ; E_2 \rightsquigarrow (v_2, \Sigma'')$$

While Loop

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma')$$

if $v_1 = \text{false}$

$$\Sigma \mid \Delta \Vdash \text{while } E_1 \text{ do } E_2 \text{ end} \rightsquigarrow (\text{true}, \Sigma')$$

While Loop

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma')$$

$$\Sigma \mid \Delta \Vdash \text{while } E_1 \text{ do } E_2 \text{ end} \rightsquigarrow (v, \Sigma''')$$

if $v_1 = \text{true}$,

where $\Sigma' \mid \Delta \Vdash E_2 \rightsquigarrow (v_2, \Sigma'')$ and

$\Sigma'' \mid \Delta \Vdash \text{while } E_1 \text{ do } E_2 \text{ end} \rightsquigarrow (v, \Sigma''')$

- 1 imPL0
- 2 imPL1
 - Syntax
 - Denotational Semantics
 - Example
- 3 Pass-by-reference, Pass-by-copy
- 4 Imperative Programming and Exception Handling
- 5 A Virtual Machine for imPL

Record Property Assignment

$$\frac{E_1 \quad E_2}{E_1.q := E_2}$$

Semantic Domains

Domain name	Definition
EV	Int + Bool + Fun + Rec + $\{\perp\}$
SV	Int + Bool + Fun + Rec
DV	Loc
Rec	Id \rightsquigarrow Loc

Record Construction

$$\Sigma^{(0)} \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma^{(1)}) \quad \Sigma^{(n-1)} \mid \Delta \Vdash E_n \rightsquigarrow (v_n, \Sigma^{(n)})$$

$$\Sigma^{(0)} \mid \Delta \Vdash [q_1 : E_1, \dots, q_n : E_n] \rightsquigarrow (f, \Sigma')$$

where l_1, \dots, l_n are new
locations in $\Sigma^{(n)}$,
 $\Sigma' = \Sigma^{(n)}[l_1 \leftarrow v_1] \cdots [l_n \leftarrow v_n]$,
 and $f(q_i) = l_i$ for $1 \leq i \leq n$.

Property Access

$$\Sigma \mid \Delta \Vdash E \rightsquigarrow (f, \Sigma')$$

$$\Sigma \mid \Delta \Vdash E.q \rightsquigarrow (\Sigma'(f(q)), \Sigma')$$

Property Assignment

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (f, \Sigma') \quad \Sigma' \mid \Delta \Vdash E_2 \rightsquigarrow (v, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash E_1.q := E_2 \rightsquigarrow (v, \Sigma''[f(q) \leftarrow v])$$

Java: Passing Objects to Methods

```
void f(MyClass myobject) {  
    myobject.myfield = 1;  
    myobject = new MyClass();  
    myobject.myfield = 2;  
}
```

imPL: Passing Records to Functions

```
let a = [Myfield: 0]
in
  (fun b ->
    b.Myfield := 1;
    b := [Myfield: 2];
    b.Myfield := 3
  end
  a);
a.Myfield
end
```

- 1 imPL0
- 2 imPL1
- 3 Pass-by-reference, Pass-by-copy**
 - Pass-by-reference
 - Pass-by-copy
- 4 Imperative Programming and Exception Handling
- 5 A Virtual Machine for imPL

Pass-by-Reference

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (f, \Sigma')$$

$$\Sigma \mid \Delta \Vdash (E_1 \ x) \rightsquigarrow f(\Delta(x), \Sigma')$$

Pass-by-Copy

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (f, \Sigma') \quad \Sigma' \mid \Delta \Vdash E_2 \rightsquigarrow (v_2, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash (E_1 \ E_2) \rightsquigarrow f(l, \Sigma''')$$

if $v_2 \in \mathbf{Rec}$,

where l, l'_1, \dots, l'_n are new locations in Σ'' ,

$\{q_1, \dots, q_n\} = \text{dom}(v_2)$,

$\Sigma''' = \Sigma'' [l \leftarrow \{(q_1, l'_1), \dots, (q_n, l'_n)\}]$

$[l'_1 \leftarrow \Sigma''(v_2(q_1))] \dots [l'_n \leftarrow \Sigma''(v_2(q_n))]$

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

Division by Zero

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma') \quad \Sigma' \mid \Delta \Vdash E_2 \rightsquigarrow (0, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash E_1/E_2 \rightsquigarrow (e, \Sigma'')$$

if $v_1 \notin \mathbf{Exc}$

and where $e = [\text{DivisionByZero:true}]$,

and $e \in \mathbf{Exc}$

Execution of try-expressions

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v, \Sigma')$$

if $v \notin \mathbf{Exc}$

$$\Sigma \mid \Delta \Vdash \text{try } E_1 \text{ catch } x \text{ with } E_2 \text{ end} \rightsquigarrow (v, \Sigma')$$

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma') \quad \Sigma'[l \leftarrow v_1] \mid \Delta[x \leftarrow l] \Vdash E_2 \rightsquigarrow (v_2, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash \text{try } E_1 \text{ catch } x \text{ with } E_2 \text{ end} \rightsquigarrow (v_2, \Sigma'')$$

if $v_1 \in \mathbf{Exc}$ and l new location

Rolling back the State

$$\Sigma \mid \Delta \Vdash E_1 \rightsquigarrow (v_1, \Sigma') \quad \Sigma[l \leftarrow v_1] \mid \Delta[x \leftarrow l] \Vdash E_2 \rightsquigarrow (v_2, \Sigma'')$$

$$\Sigma \mid \Delta \Vdash \text{try } E_1 \text{ catch } x \text{ with } E_2 \text{ end} \rightsquigarrow (v_2, \Sigma'')$$

if $v_1 \in \mathbf{Exc}$ and l new location

- 1 imPL0
- 2 imPL1
- 3 Pass-by-reference, Pass-by-copy
- 4 Imperative Programming and Exception Handling
- 5 A Virtual Machine for imPL
 - Idea
 - Assignment
 - Sequences
 - Loops

Idea

Reuse heap for implementing imperative constructs

Translation of Assignment

$$E \hookrightarrow s$$

$$x := E \hookrightarrow s.\text{ASSIGNS } x$$

Execution of Assignment

$$s(pc) = \text{ASSIGNS } x$$

$$(os, pc, e, rs, h) \Rightarrow_s (os, pc + 1, e, rs, \text{update}(e, x, v, h'))$$

where $(v, h') = \text{pop}(os, h)$

Translation of Sequences

$$E_1 \hookrightarrow s_1 \quad E_2 \hookrightarrow s_2$$

$$E_1 ; E_2 \hookrightarrow s_1.\text{POP}.s_2$$

Implementation of Sequences

$$s(pc) = \text{POP}$$

$$\frac{}{(os, pc, e, rs, h) \Rightarrow_s (os, pc + 1, e, rs, h')} \text{ where } (v, h') = \text{pop}(os, h)$$

Translation of Loops

$$E_1 \hookrightarrow s_1 \quad E_2 \hookrightarrow s_2$$

$$\begin{aligned} &\text{while } E_1 \text{ do } E_2 \\ &\quad \hookrightarrow \\ &\quad s_1.(JOFR \ |s_2 + 3|).s_2.POP. \\ &\quad (GOTOR \ - (|s_1| + 2 + |s_2|)).LDCB \ \text{true} \end{aligned}$$

Overview of Next Two Lectures

- Object-oriented programming languages
 - defining an object system
 - runtime support
 - typing
- Concurrent languages