

References, assignments, and side-effects

· Outline

- Reference model
- Typing
- Program store
- Axiomatic semantics



Expressions

Expressions

$e ::=$	•	base values
	x	variables
	$e_1 e_2$	application
	$\lambda x: t. e$	abstraction
	ref e	reference
	! e	dereference
	$e_1 := e_2$	assignment
	l	location
*	$\Lambda X. e$	type abstraction
*	$e[t]$	type application



Types and values

Values

$v ::=$	•	base values
	l	locations
	$\lambda x: t. e$	abstraction
*	$\Lambda X. e$	type abstraction

Types

$t ::=$	$Unit$	base type
	$t_1 \rightarrow t_2$	function space
	t Ref	reference cell
*	X	type variable
*	$\forall X. t$	polymorphism



Stores

Stores
 $s ::= []$ empty store
 $::= s[l := v]$ update

- $s[l := v]$: update a value in the store. The new store is the same as s except for location l , which contains value v .
- $s(l)$: fetch the value at location l
- Operation:

$$s[l_1 := v](l_2) = \begin{cases} v & \text{if } l_2 = l_1 \\ s(l_2) & \text{if } l_2 \neq l_1 \end{cases}$$



Basic beta-reduction

$((\lambda x: t.e_1) e_2, s) \rightarrow_{\beta} (e_1[e_2/x], s)$

$$\frac{(e_1, s) \rightarrow (e'_1, s')}{(e_1 e_2, s) \rightarrow (e'_1 e_2, s')} \text{ app1}$$

$$\frac{(e_2, s) \rightarrow (e'_2, s')}{(v_1 e_2, s) \rightarrow (v_1 e'_2, s')} \text{ app2}$$



Typing sequents

A type judgment is a sequent $\Gamma, \Sigma \vdash e : t$

- Γ is the usual type context $x_1: t_1, \dots, x_n: t_n$
- Σ is a store type $l_1: t_1, \dots, l_m: t_m$
- e is a program
- t is a type



Reference type judgments

$$\frac{}{\Gamma, \Sigma, l : t, \Sigma_2 \vdash l : t} \text{location}$$

$$\frac{\Gamma, \Sigma \vdash e : t}{\Gamma, \Sigma \vdash \text{ref } e : t \text{ Ref}} \text{ref}$$

$$\frac{\Gamma, \Sigma \vdash e : t \text{ Ref}}{\Gamma, \Sigma \vdash !e : t} \text{deref}$$

$$\frac{\Gamma, \Sigma \vdash e_1 : t \text{ Ref} \quad \Gamma, \Sigma \vdash e_2 : t}{\Gamma, \Sigma \vdash (e_1 := e_2) : \text{Unit}} \text{assign}$$



Subject-reduction statement

Preservation

If

$$\begin{aligned} &\Gamma, \Sigma \vdash e : t \\ &\Gamma, \Sigma \vdash s \\ &(e, s) \rightarrow (e', s') \end{aligned}$$

Then, there is *some* Σ' for which

$$\begin{aligned} &\Gamma, \Sigma' \vdash e' : t \\ &\Gamma, \Sigma' \vdash s' \end{aligned}$$



Program verification

- Type systems provide *partial* correctness
- What if we want to *verify* a program?
- Other questions?
 - What if we want to show two programs are equal (for optimization)
 - What if we want to synthesize a program from a description?



Axiomatic semantics

- One of the earliest program semantics
- Design for verification of *imperative* programs
- Also called Hoare logic (after Tony Hoare)
- A proposition in Hoare logic (called a Hoare triple)
 - $\{P\} e \{Q\}$
 - P, Q are predicates on the program state
 - " e " is a program
 - Intuitive interpretation: if P holds before e is executed, and e is executed, then Q hold afterward



Some axioms

- We won't use refs; we'll use the more conventional syntax where variables are mutable
- "Skip" (the program that does nothing)
 - $\{P\} \text{skip} \{Q\} \text{ if } P \Rightarrow Q$
- Assignment
 - $\{x = 1\} x := x + 7 \{x = 8\}$
- Looping
 - $\{P\} \text{while false do skip} \{P\}$



"Weakest-precondition" semantics

- Usually, what we care about is the state of a program when it terminates
 - The "*weakest precondition*" computes the start state for a given final state

Definition $wp(S, R)$: the set of *all* states such that executing S in any one of them is guaranteed to terminate in a state satisfying R .



Weakest precondition

$$wp(i := i + 1, i \leq 1) = i \leq 0$$

$$wp(i := j, i \leq 1) = j \leq 1$$

$$\{Q\}S\{R\} \equiv Q \Rightarrow wp(S, R)$$



Weakest precondition logical laws

- $wp(S, \text{false}) = \text{false}$
- $wp(S, Q) \wedge wp(S, R) = wp(S, Q \wedge R)$
- $wp(S, Q) \vee wp(S, R) \Rightarrow wp(S, Q \vee R)$
- if $Q \Rightarrow R$ then $wp(S, Q) \Rightarrow wp(S, R)$



Basic commands

- $wp(\text{skip}, R) = R$
- $wp(\text{abort}, R) = \text{false}$
- $wp(S_1; S_2, R) = wp(S_1, wp(S_2, R))$



Assignment

- Assignment: $x \leftarrow e$
- We assume e is functional
- Examples:
 - $wp(x \leftarrow 5, x = 5) = \mathbf{true}$
 - $wp(x \leftarrow x - 1, x < 0) = x < -1$
 - $wp(x \leftarrow x * x, x^4 = 32) = x^8 = 32$



Assignment

- $wp(x \leftarrow e, R) = R[e/x]$
- Examples:
 - $wp(x \leftarrow 5, x = 5) = (5 = 5) = \mathbf{true}$
 - $wp(x \leftarrow x - 1, x < 0) = (x - 1 < 0) = x < -1$
 - $wp(x \leftarrow x * x, x^4 = 32) = (x^{2^4} = 32) = x^8 = 32$



Swap

$$Q \Rightarrow wp(S, R) \equiv \{Q\}S\{R\}$$

$$\begin{aligned} &\{y = X \wedge x = Y\} \\ &t \leftarrow x; \\ &\{y = X \wedge t = Y\} \\ &x \leftarrow y; \\ &\{x = X \wedge t = Y\} \\ &y \leftarrow t; \\ &\{x = X \wedge y = Y\} \end{aligned}$$


Conditionals

Conditional: **if** B **then** e_1 **else** e_2

{???

if B **then** e_1 **else** e_2

{ R }

$$\begin{aligned} & wp(\mathbf{if\ } B \mathbf{\ then\ } e_1 \mathbf{\ else\ } e_2, R) \\ = & B \Rightarrow wp(e_1, R) \wedge \neg B \Rightarrow wp(e_2, R) \end{aligned}$$



Generalized conditional

$$\begin{aligned} S &= \mathbf{if} && B_1 \rightarrow e_1 \\ &| && B_2 \rightarrow e_2 \\ &| && \vdots \\ &| && B_n \rightarrow e_n \\ &\mathbf{fi} \end{aligned}$$

$$wp(S, R) = (\exists i. B_i) \wedge (\forall i. B_i \Rightarrow wp(e_i, R))$$



Loops

- This is the final, most important part
- We'll use the generalized "do loop"

$$\begin{aligned} DO &= \mathbf{do} && B_1 \rightarrow e_1 \\ &| && B_2 \rightarrow e_2 \\ &| && \vdots \\ &| && B_n \rightarrow e_n \\ &\mathbf{od} \end{aligned}$$


WP for loops

Suppose we want R to be true when the loop S terminates. Let $BB = \exists i.B_i$, to get equivalent form:

```
do BB → if B1 → e1
           |
           | Bn → en
           fi
od
do BB → IF od
```



WP for loops

- After 0 steps: $H_0(R) = \neg BB \wedge R$
- After k steps: $H_k(R) = H_0(R) \vee wp(IF, H_{k-1}(R))$
- $wp(DO, R) = \exists k.H_k(R)$



Loop reasoning

- The “wp” expression is almost useless

Summation:

```
i, s ← 1, b[0];
do i < 11 → i, s ← i + 1, s + b[i] od
{R : s = ∑k=010 b[k]}
```



Loop invariant

Summation:

```
 $i, s \leftarrow 1, b[0];$   
do  $i < 11 \rightarrow i, s \leftarrow i + 1, s + b[i]$  od  
{ $R : s = \sum_{k=0}^{10} b[k]$ }
```

Invariant:

$$P \equiv 1 \leq i \leq 11 \wedge s = \sum_{k=0}^{i-1} b[k]$$



Loop invariant

$$P \equiv 1 \leq i \leq 11 \wedge s = \sum_{k=0}^{i-1} b[k]$$

```
{true}
```

```
 $i, s \leftarrow 1, b[0];$   
{ $P$ }
```

```
do  $i < 11 \rightarrow \{i < 11 \wedge P\} i, s \leftarrow i + 1, s + b[i]$  { $P$ } od
```

```
{ $i \geq 11 \wedge P$ }
```

```
{ $R$ }
```



Multiplication

```
{ $b \geq 0$ }
```

```
 $x, y, z \leftarrow a, b, 0;$ 
```

```
do  $y > 0 \wedge \text{even}(x) \rightarrow y, x \leftarrow y/2, x + x$ 
```

```
  |  $\text{odd}(y) \rightarrow y, z \leftarrow y - 1, z + x$ 
```

```
od
```

```
{ $R : z = a * b$ }
```



Introduce an invariant

$P \equiv y \geq 0 \wedge z + x * y = a * b$

$\{b \geq 0\}$

$x, y, z \leftarrow a, b, 0;$

$\{P\}$

do $y > 0 \wedge \text{even}(x) \rightarrow \{P \wedge y > 0 \wedge \text{even}(y)\} y, x \leftarrow y/2, x + x \{P\}$
| $\text{odd}(y) \rightarrow \{P \wedge \text{odd}(y)\} y, z \leftarrow y - 1, z + x \{P\}$

od

$\{P \wedge y \leq 0 \wedge \neg \text{odd}(y)\}$

$\{P \wedge y = 0\}$

$\{R : z = a * b\}$


