Section 5: Imperative Languages

Languages that utilise stores are called **imperative** languages. The store is a data structure that exists independently of any program in the language.

Essential Features:

- Sequential execution,
- Implicit data structure (the ``store'')
  - Existence indpendent of any program,
  - Not mentioned in language syntax,
  - Phrases may access it and update it.
- Relationship between store and programs:
  - Critical for evaluation of phrases. Phrase meaning depends on store.
  - Communication between phrases. Phrases deposit values in store for use by other phrases; sequencing mechanism establishes communication order.
  - Inherently ``large'' argument. Only one copy existing during execution.

A Language with Assignment

- Declaration-free Pascal subset.
- Program = sequence of commands
- C: Command $\rightarrow$ Store$_\perp$ $\rightarrow$ Store $_\perp$
  - A command produces a new store from its store argument.
  - Command might not terminate (``loop'') **C**[| C |]s = $\perp$
  - Follow up commands C ‘, will not evaluate
    C[| C ‘|] Store$_\perp$ $\rightarrow$ Store $_\perp$ is strict.
  - Command sequencing is store composition
    C[| C_1; C_2 |] = **C**[| C_2 |] ° **C**[| C_1 |]

(See Photocopy from Schmidt, Figures 5.1 and 5.2)

Valuation Functions

P: Program $\rightarrow$ Nat $\rightarrow$ Nat$_\perp$
Program maps input number to an answer number; non-termination is possible (co-domain includes ?).
C: Command $\rightarrow$ Store $_\perp$ $\rightarrow$ Store $_\perp$
Command maps store into a new store;
predecessor command may not have terminated (domain includes $\perp$) and command may not terminate (co-domain includes $\perp$).
E: Expression $\rightarrow$ Store $\rightarrow$ Nat
Expression maps store into natural number.
B: Bool-exp $\rightarrow$ Store $\rightarrow$ Tr
Boolean expression maps store into truth value.
N: Numeral $\rightarrow$ Nat Numeral yields a natural value.

Program Denotation

- How to understand a program?
- A possibility is to compute its denotation with a particular input argument. P: Program $\rightarrow$ Nat $\rightarrow$ Nat $_\perp$
- Various results for various inputs. Natural number or $\perp$ (non-termination)

P [| Z:=1; if A=0 then diverge; Z:=3 **|]** (two)

Simplification

P[| Z:=1; if A=0 then diverge; Z:=3. **|]**(two)

= let s = (**update [| A |] two newstore**) in
let s’ = C[| Z:=1; if A=0 then diverge; Z:=3 |] **s** in
access [| Z |] s'

= < By Simplification>

let s1 = (**[[| A |] $\rightarrow$ two] newstore**)
let s' = C[| Z:=1; if A=0 then diverge; Z:=3 |]**s1** in
access [| Z |] s'

C[| **Z:=1**; if A=0 then diverge; Z:=3 |]s1
= (**λs**.C[| if A=0 then diverge; Z:=3 |] **(C[| Z:=1|]s)) s1**

= <as the store s1 may be bound to s)
C[| if A=0 then diverge; Z:=3 |](C[| Z:=1 |]**s1**)

Now work on C[| Z:=1 |]s1
C[| **Z:=1** |]s1
= (λ s.**update** [| Z |] (E[| 1|]s) s)s1
= update [| Z |] (E[| 1|]s1) **s1**
= update [| Z |] (**N[|1|]**) s1
= update [| Z |] **one** s1
= **[[| Z |] → one]s1** which we call s2

C[| if A=0 then diverge; Z:=3 |](C[| Z:=1 |]s1)
< from previous slide>
= C[| if A=0 then diverge; Z:=3|]**s2**
<from ; rule>
= (λ s.C[| Z:=3 |] **(C[| if A=0 then diverge |]s)**s2
<from if rule>
= (λ s.C[| Z:=3 |] **((λ s.B[| A=0|]s → C[| diverge |]s [] s) s)**)s2

= C[| Z:=3|]((**λ s**.B[| A=0 |]s → C[| diverge |]**s [] s**)s2)

= C[| Z:=3|](B[| A=0 |]s2 → C[| diverge |]**s2 [] s2**)

B[| A=0 |]s2

= (**λ s**.E[| A|]s equals E[| 0 |]**s**)s2

= E[| A|]**s2** equals E[| 0 |]**s2**

= (access [| A |] s2) equals zero

access [| A |] s2
= s2 [| A |]
= ([[| Z |] → one][[| A |] → two] newstore) [| A |]
= ([[| A |] → two] newstore) [| A |]
= two

(access [| A |] s2) equals zero
= two equals zero
= false

C[| Z:=3|](B[| A=0 |]s2 → C[| diverge |]s2 [] s2)
= C[| Z:=3 |](false → C[| diverge |]s2 [] s2)
= C[| Z:=3 |]s2
= (λ s.update [| Z |] (E[| 3|]s) s)s2
= update [| Z |] (E[| 3|]s2) s2
= update [| Z |] (N[| 3 |]) s2
= update [| Z |] three s2
= [[| Z |] → three]s2

Denotation of the Entire Program

let s1 = ([[| A |] → two] newstore)
s2 = [[| Z|] → one]s1
s' = C[| Z:=1; if A=0 then diverge; Z:=3 |]s1
in access [| Z|] s'
= let s1 = ([[| A|] → two] newstore)
s2 = [[| Z |] → one]s1
s' = [[| Z |] → three]s2
in access [| Z |] s'

- access [| Z |] s'
  = access [| Z |] [[| Z |] → three]s2
  = [[| Z |] → three]s2 [| Z |]
  = three

Program Denotation when input is zero

P[| Z:=1; if A=0 then diverge; Z:=3 |](zero)

= let s3 = [[| A |] → zero]newstore
s' = C[| Z:=1; if A=0 then diverge; Z:=3 |]s3
in access [| Z |] s'

= let s3 = [[| A |] → zero]newstore
s 4 = [[| Z |] → one]s3
s' = C[| if A=0 then diverge; Z:=3 |]s 4
in access [| Z |] s'

= let s 3 = [[| A|] → zero]newstore
s 4 = [[| Z |] → one]s3
s 5 = C[| if A=0 then diverge |]s4
s' = C[| Z:=3 |]s5
in access [| Z |] s'

C[| if A=0 then diverge |]s 4
= B[| A=0 |]s 4 → C[| diverge |]s 4 [] s 4
= true → C[| diverge |]s 4 [] s 4
= C[| diverge |]s 4
= (λs.⊥)s 4
= ⊥

let s 3 = [[| A |] → zero]newstore
s 4 = [[| Z |] → one]s 3
s 5 = C[| if A=0 then diverge |]s 4
s' = C[| Z:=3 |]s 5
in access [| Z |] s'

= let s 3 = [[| A |] → zero]newstore
s 4 = [[| Z |] → one]s 3
s 5 = ⊥
s' = C[| Z:=3 |]s 5 **in access [| Z |] s'**

=

let s' = C[| Z:=3 |] ⊥
in access [| Z |] s'

= let s' = (λ s.update [| Z |] (E[| 3|]s)) ⊥
in access [| Z |] s'

= let s' = ⊥
in access [| Z |] s'

= access [| Z |] ⊥

= ⊥

Program Equivalence

Is C[| X:=0; Y:=X+1 |] equivalent to C[| Y:=1;X:=0 |]
C: Command → $Store_\perp$ → $Store_\perp$
Show C[| C 1 |]s = C[| C 2 |]s for every s.

- C[| C 1 |] ⊥ = ⊥ = C[| C 2 |] ⊥.
- Assume proper store s.
- C[| X:=0; Y:=X+1|]s
  = C[| Y:=X+1|] (**C[| X:=0 |]s)**
  = C[| Y:=X+1|] (**[[| X|] → zero**]s)
  = update [| Y|] (E[| X+1 |]([[| X |] → zero]s)) [[| X |] → zero]s
  = update [| Y |] **one** [[| X |] → zero]s
  = [[| Y |] → one] [[| X |] → zero]s
  Call this result s1

Program Equivalence

C[| Y:=1; X:=0 |]s
= C[| X:=0 |] **(C[| Y:=1 |]s)**
= C[| X:=0 |] **([[| Y |] → one]s)**
= [**[| X|] → zero**][[| Y|] → one]s **Call this result s2**

Is s1 equivalent to s2 ???
The two values are defined stores. Are they the same store?
We cannot simplify s1 into s2 using the simplification rules.
The stores are functions : Id → Nat
To show that the two stores are the same, we must show that each produces the same number answer from the identifier argument.
There are three cases to consider:

Case 1: The argument is [| X|]

s1 [| X |]
= ([[| Y |] → one][[| X |] → zero]s)[| X |]
= ([| X |] → zero]s)[| X |]
= zero

s2 [| X |]
= ([[| X |] → zero][[| Y |] → one]s)[| X |]
= zero

Case 2: The argument is [| Y |]

s1[| Y |]
= ([[| Y |] → one][[| X |] → zero]s)[| Y |]
= one

s2 [| Y |]
= ([[| X |] → zero][[| Y |] → one]s)[| Y |]
= ([[| Y |] → one]s)[| Y|]
= one

Case 3: The argument is [| I |] (some other identifier)

s1 [| I |]
= ([[| Y |] → one][[| X |] → zero]s)[| I |]
= ([[| X |] → zero]s)[| I |]
= s[| I |]
= ([[| Y |] → one]s)[| I |]
= ([[| X |] → zero][[| Y |] → one]s)[| I |]
= s2 [| I |].

Programs Are Functions

Simplifications were operational-like.
Answer computed from program and input.

P [|Z:=1; if A=0 then diverge; Z:=3 |]

= λ n.let s = ([[| A |] → n] newstore) in
let s' = C[| Z:=1; if A=0 then diverge; Z:=3 |] s
in access[|Z|]s'
= λ n.let s = ([[| A |] → n] newstore) in
let s'= (λ s.(λ s.C[| Z:=3 |] (C[|if A =0 then diverge|] s))s)(C[[Z:=1|] s)
in access[|Z|]s'

= λ n.let s = ([[| A |] → n] newstore) in
let s'= (λ s.(λ s.[[| Z|] → n]s)
((λ s.(access[|A|] s) equals zero → (λ s.⊥) s[] s)s))
((λ s. [[|Z |] → one]s) s)
in access[|Z|]s'

= λ n.let s = ([[| A |] → n] newstore) in
let s'= (let s'1 = .[[| Z|] → one]s) in
let s'2 = (access[|A|] s'1) equals zero → (λ s.⊥) s'1[] s'1
in [[|Z |] → three]s'2)
in access[|Z|]s'

Program Denotation

P [| Z:=1; if A=0 then diverge; Z:=3 |]
= λ n.(n equals zero) → ⊥ [] three

- Denotation extracts essence of a program.
- Store disappears
  - temporary data structure; not contained in the input/output relation of a program.
- Transformation resembles compilation.
- Simplification resembles code optimization.