5. Imperative Languages I

Imperative Languages

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:

1. Critical for evaluation of phrases.

   Phrase meaning depends on store.

2. Communication between phrases.

   Phrases deposit values in store for use by other phrases; sequencing mechanism establishes communication order.

3. 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")*
    $\Rightarrow$ $\mathbf{C}[[\mathrm{C}]]s = \perp$.
  - *Followup commands will not evaluate*
    $\Rightarrow$ $\mathbf{C}[[\mathrm{C}]]$:$Store_\perp \rightarrow Store_\perp$ *is strict.*
  - *Command sequencing is store composition*
    $\Rightarrow$ $\mathbf{C}[[\mathrm{C}_1;\ \mathrm{C}_2]](s) = \mathbf{C}[[\mathrm{C}_2]](\mathbf{C}[[\mathrm{C}_1]]s)$
    $\Rightarrow$ $\mathbf{C}[[\mathrm{C}_1;\ \mathrm{C}_2]] = \mathbf{C}[[\mathrm{C}_1]] \circ \mathbf{C}[[\mathrm{C}_2]]$

(See 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 $\perp$).

- **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 (codomain 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.

  $\mathbf{P}$: *Program* $\longrightarrow$ *Nat* $\longrightarrow$ $Nat_\perp$

- Various results for various inputs.

  Natural number or $\perp$ *(non-termination)*

```
Z:=1;
if A=0 then diverge;
Z:=3.
```

$\Downarrow$ **P** (*two*)

Denotation

Simplification

P[[Z:=1; **if** A=0 **then diverge**; Z:=3.]](*two*)
= let s_1 = (*update* [[A]] *two newstore*)
  s' = **C**[[Z:=1; **if** A=0 **then diverge**; Z:=3]]s_1
  in *access* [[Z]] s'
= let s_1 = ([[[A]] $\mapsto$ *two*] *newstore*)
  s' = **C**[[Z:=1; **if** A=0 **then diverge**; Z:=3]]s_1
  in *access* [[Z]] s'

C[[Z:=1; **if** A=0 **then diverge**; Z:=3]]s_1
= ($\underline{\lambda}$s.**C**[[**if** A=0 **then diverge**; Z:=3]]
  (**C**[[Z:=1]]s))s_1
= **C**[[**if** A=0 **then diverge**; Z:=3]](**C**[[Z:=1]]s_1)

C[[Z:=1]]s_1
= ($\underline{\lambda}$s.*update* [[Z]] (**E**[[1]]s) s)s_1
= *update* [[Z]] (**E**[[1]]s_1) s_1
= *update* [[Z]] (**N**[[1]]) s_1
= *update* [[Z]] *one* s_1
= [[[Z]] $\mapsto$ *one*]s_1
= s_2

Simplification

$\mathbf{C}$[[**if** A=0 **then diverge**; Z:=3]]($\underline{\mathbf{C}[[\text{Z:=1}]]s_1}$)
= $\mathbf{C}$[[**if** A=0 **then diverge**; Z:=3]]s_2
= ($\underline{\lambda}s.\mathbf{C}$[[Z:=3]]($\mathbf{C}$[[**if** A=0 **then diverge**]]s)s_2
= ($\underline{\lambda}s.\mathbf{C}$[[Z:=3]](($\underline{\lambda}s.\mathbf{B}$[[A=0]]$s$ $\rightarrow$
  $\mathbf{C}$[[**diverge**]]s [] s) s))s_2
= $\mathbf{C}$[[Z:=3]](($\underline{\lambda}s.\mathbf{B}$[[A=0]]$s$ $\rightarrow$
  $\mathbf{C}$[[**diverge**]]s [] s)s_2)
= $\mathbf{C}$[[Z:=3]]($\underline{\mathbf{B}[[\text{A=0}]]s_2}$ $\rightarrow$ $\mathbf{C}$[[**diverge**]]s_2 [] s_2)

$\mathbf{B}$[[A=0]]s_2
= (λ s.$\mathbf{E}$[[A]]s *equals* $\mathbf{E}$[[0]]s)s_2
= $\mathbf{E}$[[A]]s_2 *equals* $\mathbf{E}$[[0]]s_2
= $\underline{(\textit{access}\ [[\text{A}]]\ s_2)}$ *equals zero*

access [[A]] s_2
= s_2 [[A]]
= ([[[Z]] $\mapsto$ *one*][[[A]] $\mapsto$ *two*] *newstore*) [[A]]
= ([[[A]] $\mapsto$ *two*] *newstore*) [[A]]
= *two*

Simplification

$\underline{(\mathit{access}\ [\![\mathsf{A}]\!]\ s_2)}\ \mathit{equals\ zero}$
$= \mathit{two\ equals\ zero}$
$= \mathit{false}$

$\mathbf{C}[\![\mathsf{Z:=3}]\!](\underline{\mathbf{B}[\![\mathsf{A=0}]\!]s_2} \rightarrow \mathbf{C}[\![\mathbf{diverge}]\!]s_2\ []\ s_2)$
$= \mathbf{C}[\![\mathsf{Z:=3}]\!](\mathit{false} \rightarrow \mathbf{C}[\![\mathbf{diverge}]\!]s_2\ []\ s_2)$
$= \mathbf{C}[\![\mathsf{Z:=3}]\!]s_2$
$= (\underline{\lambda}s.\mathit{update}\ [\![\mathsf{Z}]\!]\ (\mathbf{E}[\![\mathsf{3}]\!]s)\ s)s_2$
$= \mathit{update}\ [\![\mathsf{Z}]\!]\ (\mathbf{E}[\![\mathsf{3}]\!]s_2)\ s_2$
$= \mathit{update}\ [\![\mathsf{Z}]\!]\ (\mathbf{N}[\![\mathsf{3}]\!])\ s_2$
$= \mathit{update}\ [\![\mathsf{Z}]\!]\ \mathit{three}\ s_2$
$= [\ [\![\mathsf{Z}]\!] \mapsto \mathit{three}\ ]s_2$

Simplification

let s_1 = ([[[A]] $\mapsto$ *two*] *newstore*)
s_2 = [[[Z]] $\mapsto$ *one*]s_1
s' = **C**[[Z:=1; **if** A=0 **then diverge**; Z:=3]]s_1
in *access* [[Z]] s'
= let s_1 = ([[[A]] $\mapsto$ *two*] *newstore*)
s_2 = [[[Z]] $\mapsto$ *one*]s_1
s' = [[[Z]] $\mapsto$ *three*]s_2
in *access* [[Z]] s'

access [[Z]] s'
= *access* [[Z]] [[[Z]] $\mapsto$ *three*]s_2
= [[[Z]] $\mapsto$ *three*]s_2 [[Z]]
= *three*

Program Denotation

Program plus input $\Rightarrow$ result.

```
Z:=1;
if A=0 then diverge;
Z:=3.
```

$$\Downarrow \textbf{P}\ (\mathit{two})$$

let s_1 = ([[[A]] $\mapsto$ *two*] *newstore*)
s_2 = [[[Z]] $\mapsto$ *one*]s_1
s_3 = ((*access* [[A]] s_2) *equals zero*) $\rightarrow \bot$ [] s_2
s' = [[[Z]] $\mapsto$ *three*]s_3
in *access* [[Z]] s'

$$\Vert$$

three

Intermediate form (only partial simplification) delivers more insight!

Simplification II

P[[Z:=1; **if** A=0 **then diverge**; Z:=3.]](*zero*)
= let s_3 = [[[A]] $\mapsto$ *zero*]*newstore*
  s' = **C**[[Z:=1; **if** A=0 **then diverge**; Z:=3]]s_3
  in *access* [[Z]] s'
= let s_3 = [[[A]] $\mapsto$ *zero*]*newstore*
  s_4 = [[[Z]] $\mapsto$ *one*]s_3
  s' = **C**[[**if** A=0 **then diverge**; Z:=3]]s_4
  in *access* [[Z]] s'
= let s_3 = [[[A]] $\mapsto$ *zero*]*newstore*
  s_4 = [[[Z]] $\mapsto$ *one*]s_3
  s_5 = $\underline{\textbf{C}[[\textbf{if A=0 then diverge}]]s_4}$
  s' = **C**[[Z:=3]]s_5
  in *access* [[Z]] s'

C[[**if** A=0 **then diverge**]]s_4
= **B**[[A=0]]s_4 $\rightarrow$ **C**[[**diverge**]]s_4 [] s_4
= *true* $\rightarrow$ **C**[[**diverge**]]s_4 [] s_4
= **C**[[**diverge**]]s_4
= $(\underline{\lambda}s.\bot)s_4$
= $\bot$

Simplification II

let s_3 = [[[A]] $\mapsto$ *zero*]*newstore*
s_4 = [[[Z]] $\mapsto$ *one*]s_3
s_5 = $\underline{\mathbf{C}[[\textbf{if A=0 then diverge}]]s_4}$
s' = $\mathbf{C}$[[Z:=3]]s_5
in *access* [[Z]] s'
= let s_3 = [[[A]] $\mapsto$ *zero*]*newstore*
s_4 = [[[Z]] $\mapsto$ *one*]s_3
s_5 = $\bot$
s' = $\mathbf{C}$[[Z:=3]]s_5
in *access* [[Z]] s'
= let s' = $\mathbf{C}$[[Z:=3]]$\bot$
in *access* [[Z]] s'
= let s' = $(\underline{\lambda}s.$*update* [[Z]] ($\mathbf{E}$[[3]]s))$\bot$
in *access* [[Z]] s'
= let s' = $\bot$
in *access* [[Z]] s'
= *access* [[Z]] $\bot$
= $\bot$

Program Equivalence

$$\mathbf{C}[\![\text{X:=0; Y:=X+1}]\!] \stackrel{?}{=} \mathbf{C}[\![\text{Y:=1;X:=0}]\!]$$

- $\mathbf{C}$: $\mathit{Store}_\perp \to \mathit{Store}_\perp$
- Show $\mathbf{C}[\![\mathrm{C}_1]\!]s = \mathbf{C}[\![\mathrm{C}_2]\!]s$ for every s.
- $\mathbf{C}[\![\mathrm{C}_1]\!]\perp = \perp = \mathbf{C}[\![\mathrm{C}_2]\!]\perp$.

Assume proper store s.

$\mathbf{C}$[[X:=0; Y:=X+1]]s
= $\mathbf{C}$[[Y:=X+1]]($\mathbf{C}$[[X:=0]]s)
= $\mathbf{C}$[[Y:=X+1]]([[[X]] $\mapsto$ *zero*]s)
= *update* [[Y]] ($\mathbf{E}$[[X+1]]([[[X]] $\mapsto$ *zero*]s))
  [[[X]] $\mapsto$ *zero*]s
= *update* [[Y]] *one* [[[X]] $\mapsto$ *zero*]s
= [[[Y]] $\mapsto$ *one*][[[X]] $\mapsto$ *zero*]s
= s_1

Program Equivalence

$\mathbf{C}$[[Y:=1; X:=0]]s
$= \mathbf{C}$[[X:=0]]($\mathbf{C}$[[Y:=1]]s)
$=$ $\mathbf{C}$[[X:=0]]([[[Y]] $\mapsto$ *one*]s)
$=$ [[[X]] $\mapsto$ *zero*][[[Y]] $\mapsto$ *one*]$s = s_2$

- $s_1 \stackrel{?}{=} s_2$.
- s_1, s_2: *Id* $\to$ *Nat*
- Show $s_1(id) = s_2(id)$ for every *id*.

1. *id* = [[X]]: s_1[[X]] = ([[[Y]] $\mapsto$ *one*][[[X]] $\mapsto$ *zero*]s)[[X]] = ([[X]] $\mapsto$ *zero*]s)[[X]] = *zero* = ([[[X]] $\mapsto$ *zero*][[[Y]] $\mapsto$ *one*]s)[[X]] = s_2[[X]].
2. *id* = [[Y]]: s_1[[Y]] = ([[[Y]] $\mapsto$ *one*][[[X]] $\mapsto$ *zero*]s)[[Y]] = *one* = ([[[Y]] $\mapsto$ *one*]s)[[Y]] = ([[[X]] $\mapsto$ *zero*][[[Y]] $\mapsto$ *one*]s)[[Y]] = s_2[[Y]].
3. *id* = [[I]]: s_1[[I]] = ([[[Y]] $\mapsto$ *one*][[[X]] $\mapsto$ *zero*]s)[[I]] = ([[[X]] $\mapsto$ *zero*]s)[[I]] = s[[I]] = ([[[Y]] $\mapsto$ *one*]s)[[I]] = ([[[X]] $\mapsto$ *zero*][[[Y]] $\mapsto$ *one*]s)[[I]] = s_2[[I]].

Programs Are Functions

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

```
Z:=1;
if A=0 then diverge;
Z:=3.
```

$\Downarrow$ **P**

λn.let s_1 = ([[[A]] $\mapsto$ n] *newstore*)
s_2 = [[[Z]] $\mapsto$ *one*]s_1
s_3 = ((*access* [[A]] s_2) *equals zero*) $\rightarrow$ $\perp$ [] s_2
s' = [[[Z]] $\mapsto$ *three*]s_3
in *access* [[Z]] s'

Study denotation without sample input!

Programs are Functions

$$\|$$

$$\lambda n.\text{let } s_1 = (\ [\ [[\text{A}]] \mapsto n]\ \textit{newstore})$$
$$\quad\quad s_2 = [\ [[\text{Z}]] \mapsto \textit{one}\ ]s_1$$
$$\quad\quad s_3 = (n\ \textit{equals zero}) \rightarrow \bot\ []\ s_2$$
$$\quad\quad s' = [\ [[\text{Z}]] \mapsto \textit{three}\ ]s_3$$
$$\text{in } \textit{access}\ [[\text{Z}]]\ s'$$

$$\|$$

$$\lambda n.\text{let } s_1 = (\ [\ [[\text{A}]] \mapsto n]\ \textit{newstore})$$
$$\quad\quad s_2 = [\ [[\text{Z}]] \mapsto \textit{one}\ ]s_1$$
$$\quad\quad s_3 = (n\ \textit{equals zero}) \rightarrow \bot\ []\ s_2$$
$$\quad\quad s' = (n\ \textit{equals zero}) \rightarrow \bot\ []\ [\ [[\text{Z}]] \mapsto \textit{three}\ ]s_3$$
$$\text{in } \textit{access}\ [[\text{Z}]]\ s'$$

$$\|$$

$$\lambda n.\text{let } s_1 = (\ [\ [[\text{A}]] \mapsto n]\ \textit{newstore})$$
$$\quad\quad s_2 = [\ [[\text{Z}]] \mapsto \textit{one}\ ]s_1$$
$$\text{in } (n\ \textit{equals zero}) \rightarrow \bot$$
$$\quad\quad []\ \textit{access}\ [[\text{Z}]]\ [\ [[\text{Z}]] \mapsto \textit{three}\ ]s_2$$

$$\|$$

$$\lambda n.(n\ \textit{equals zero}) \rightarrow \bot\ []\ \textit{three}$$

Program Denotation

```
Z:=1;
if A=0 then diverge;
Z:=3.
```

$\Downarrow$ **P**

$$\lambda n.(n \textit{ equals zero}) \longrightarrow \bot \; [] \; \textit{three}$$

- 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.