

Imperative Languages I

Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at

<http://www.risc.uni-linz.ac.at/people/schreine>

Imperative Languages

Essential Features:

- Sequential execution,
- Implicit data structure (the “store”)
 - Existence independent 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*
- $C: \text{Command} \rightarrow \text{Store}_\perp \rightarrow \text{Store}_\perp$
 - A command produces a new store from its store argument.
 - Command might not terminate (“loop”)
 - $C[[C]]s = \perp$.
 - Followup commands will not evaluate
 - $C[[C]]: \text{Store}_\perp \rightarrow \text{Store}_\perp$ is strict.
 - Command sequencing is store composition
 - $C[[C_1; C_2]](s) = C[[C_2]](C[[C_1]]s)$
 - $C[[C_1; C_2]] = C[[C_1]] \circ C[[C_2]]$

(See Schmidt, Figures 5.1 and 5.2)

Valuation Functions

- **P**: $\text{Program} \rightarrow \text{Nat} \rightarrow \text{Nat}_{\perp}$

Program maps input number to an answer number; non-termination is possible (codomain includes \perp).

- **C**: $\text{Command} \rightarrow \text{Store}_{\perp} \rightarrow \text{Store}_{\perp}$

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**: $\text{Expression} \rightarrow \text{Store} \rightarrow \text{Nat}$

Expression maps store into natural number.

- **B**: $\text{Bool-exp} \rightarrow \text{Store} \rightarrow \text{Tr}$

Boolean expression maps store into truth value.

- **N**: $\text{Numeral} \rightarrow \text{Nat}$

Numeral yields a natural value.

Program Denotation

- How to understand a program?
- A possibility is to compute its denotation with a particular input argument.

$P: \text{Program} \rightarrow \text{Nat} \rightarrow \text{Nat}_\perp$

- 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

$$\begin{aligned}
& \mathbf{P}[[Z:=1; \text{if } A=0 \text{ then diverge; } Z:=3.]](\text{two}) \\
&= \text{let } s_1 = (\text{update } [[A]] \text{ two newstore}) \\
&\quad s' = \mathbf{C}[[Z:=1; \text{if } A=0 \text{ then diverge; } Z:=3]]_{s_1} \\
&\quad \text{in access } [[Z]] s' \\
&= \text{let } s_1 = ([[[A]] \mapsto \text{two}] \text{ newstore}) \\
&\quad s' = \underline{\mathbf{C}[[Z:=1; \text{if } A=0 \text{ then diverge; } Z:=3]]}_{s_1} \\
&\quad \text{in access } [[Z]] s'
\end{aligned}$$

$$\begin{aligned}
& \mathbf{C}[[Z:=1; \text{if } A=0 \text{ then diverge; } Z:=3]]_{s_1} \\
&= (\lambda s. \mathbf{C}[[\text{if } A=0 \text{ then diverge; } Z:=3]]) \\
&\quad (\mathbf{C}[[Z:=1]]_s)_{s_1} \\
&= \mathbf{C}[[\text{if } A=0 \text{ then diverge; } Z:=3]](\underline{\mathbf{C}[[Z:=1]]}_{s_1})
\end{aligned}$$

$$\begin{aligned}
& \mathbf{C}[[Z:=1]]_{s_1} \\
&= (\lambda s. \text{update } [[Z]] (\mathbf{E}[[1]]_s) s)_{s_1} \\
&= \text{update } [[Z]] (\mathbf{E}[[1]]_{s_1}) s_1 \\
&= \text{update } [[Z]] (\mathbf{N}[[1]]) s_1 \\
&= \text{update } [[Z]] \text{ one } s_1 \\
&= [[[Z]] \mapsto \text{one}]_{s_1} \\
&= s_2
\end{aligned}$$

Simplification

$$\begin{aligned}
& \mathbf{C}[[\text{if } A=0 \text{ then diverge; } Z:=3]](\underline{\mathbf{C}}[[Z:=1]]_{s_1}) \\
&= \mathbf{C}[[\text{if } A=0 \text{ then diverge; } Z:=3]]_{s_2} \\
&= (\lambda s. \mathbf{C}[[Z:=3]])(\mathbf{C}[[\text{if } A=0 \text{ then diverge}]]_s)_{s_2} \\
&= (\lambda s. \mathbf{C}[[Z:=3]])((\lambda s. \mathbf{B}[[A=0]]_s \rightarrow \\
&\quad \mathbf{C}[[\text{diverge}]]_s [] s) s)_{s_2} \\
&= \mathbf{C}[[Z:=3]]((\lambda s. \mathbf{B}[[A=0]]_s \rightarrow \\
&\quad \mathbf{C}[[\text{diverge}]]_s [] s)_{s_2}) \\
&= \mathbf{C}[[Z:=3]](\underline{\mathbf{B}}[[A=0]]_{s_2} \rightarrow \mathbf{C}[[\text{diverge}]]_{s_2} [] s_2)
\end{aligned}$$

$$\begin{aligned}
& \mathbf{B}[[A=0]]_{s_2} \\
&= (\lambda s. \mathbf{E}[[A]]_s \text{ equals } \mathbf{E}[[0]]_s)_{s_2} \\
&= \mathbf{E}[[A]]_{s_2} \text{ equals } \mathbf{E}[[0]]_{s_2} \\
&= (\underline{\text{access } [[A]]}_{s_2}) \text{ equals zero}
\end{aligned}$$

$$\begin{aligned}
& \text{access } [[A]]_{s_2} \\
&= s_2 [[A]] \\
&= ([[[Z]] \mapsto \text{one}] [[[A]] \mapsto \text{two}] \text{newstore}) [[A]] \\
&= ([[[A]] \mapsto \text{two}] \text{newstore}) [[A]] \\
&= \text{two}
\end{aligned}$$

Simplification

(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

= (λs.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

Simplification

$$\begin{aligned}
 &\text{let } s_1 = ([[A] \mapsto \text{two}] \text{newstore}) \\
 &\quad s_2 = [[Z] \mapsto \text{one}]_{s_1} \\
 &\quad s' = \underline{\mathbf{C}[Z:=1; \text{if } A=0 \text{ then diverge; } Z:=3]}_{s_1} \\
 &\quad \text{in } \underline{\text{access } [Z]} \ s' \\
 = &\text{let } s_1 = ([[A] \mapsto \text{two}] \text{newstore}) \\
 &\quad s_2 = [[Z] \mapsto \text{one}]_{s_1} \\
 &\quad s' = [[Z] \mapsto \text{three}]_{s_2} \\
 &\quad \text{in } \underline{\text{access } [Z]} \ s'
 \end{aligned}$$

$$\begin{aligned}
 &\text{access } [Z] \ s' \\
 = &\text{access } [Z] \ [[Z] \mapsto \text{three}]_{s_2} \\
 = &[[Z] \mapsto \text{three}]_{s_2} \ [Z] \\
 = &\text{three}
 \end{aligned}$$

Program Denotation

Program plus input \rightarrow result.

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

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

```
let  $s_1 = ( [ [A] \mapsto \textit{two} ] \textit{newstore} )$ 
     $s_2 = [ [Z] \mapsto \textit{one} ] s_1$ 
     $s_3 = ((\textit{access } [A] s_2) \textit{ equals zero}) \rightarrow \perp [] s_2$ 
     $s' = [ [Z] \mapsto \textit{three} ] s_3$ 
in  $\textit{access } [Z] s'$ 
```

\parallel

three

*Intermediate form (only partial simplification)
delivers more insight!*

Simplification II

$$\begin{aligned}
& \mathbf{P}[[Z:=1; \text{if } A=0 \text{ then diverge}; Z:=3.]](\text{zero}) \\
&= \text{let } s_3 = [[[A]] \mapsto \text{zero}] \text{newstore} \\
&\quad s' = \mathbf{C}[[Z:=1; \text{if } A=0 \text{ then diverge}; Z:=3]]_{s_3} \\
&\quad \text{in access } [[Z]] s' \\
&= \text{let } s_3 = [[[A]] \mapsto \text{zero}] \text{newstore} \\
&\quad s_4 = [[[Z]] \mapsto \text{one}]_{s_3} \\
&\quad s' = \mathbf{C}[[\text{if } A=0 \text{ then diverge}; Z:=3]]_{s_4} \\
&\quad \text{in access } [[Z]] s' \\
&= \text{let } s_3 = [[[A]] \mapsto \text{zero}] \text{newstore} \\
&\quad s_4 = [[[Z]] \mapsto \text{one}]_{s_3} \\
&\quad s_5 = \underline{\mathbf{C}[[\text{if } A=0 \text{ then diverge}]]}_{s_4} \\
&\quad s' = \mathbf{C}[[Z:=3]]_{s_5} \\
&\quad \text{in access } [[Z]] s'
\end{aligned}$$

$$\begin{aligned}
& \mathbf{C}[[\text{if } A=0 \text{ then diverge}]]_{s_4} \\
&= \mathbf{B}[[A=0]]_{s_4} \rightarrow \mathbf{C}[[\text{diverge}]]_{s_4} [] s_4 \\
&= \text{true} \rightarrow \mathbf{C}[[\text{diverge}]]_{s_4} [] s_4 \\
&= \mathbf{C}[[\text{diverge}]]_{s_4} \\
&= (\underline{\lambda}s. \perp)_{s_4} \\
&= \perp
\end{aligned}$$

Simplification II

$$\begin{aligned}
 & \text{let } s_3 = [\llbracket A \rrbracket \mapsto \text{zero}] \text{newstore} \\
 & \quad s_4 = [\llbracket Z \rrbracket \mapsto \text{one}] s_3 \\
 & \quad s_5 = \mathbf{C}[\llbracket \text{if } A=0 \text{ then diverge} \rrbracket] s_4 \\
 & \quad s' = \mathbf{C}[\llbracket Z:=3 \rrbracket] s_5 \\
 & \quad \text{in access } \llbracket Z \rrbracket s' \\
 = & \text{let } s_3 = [\llbracket A \rrbracket \mapsto \text{zero}] \text{newstore} \\
 & \quad s_4 = [\llbracket Z \rrbracket \mapsto \text{one}] s_3 \\
 & \quad s_5 = \perp \\
 & \quad s' = \mathbf{C}[\llbracket Z:=3 \rrbracket] s_5 \\
 & \quad \text{in access } \llbracket Z \rrbracket s' \\
 = & \text{let } s' = \mathbf{C}[\llbracket Z:=3 \rrbracket] \perp \\
 & \quad \text{in access } \llbracket Z \rrbracket s' \\
 = & \text{let } s' = (\lambda s. \text{update } \llbracket Z \rrbracket (\mathbf{E}[\llbracket 3 \rrbracket] s)) \perp \\
 & \quad \text{inaccess } \llbracket Z \rrbracket s' \\
 = & \text{let } s' = \perp \\
 & \quad \text{inaccess } \llbracket Z \rrbracket s' \\
 = & \text{access } \llbracket Z \rrbracket \perp \\
 = & \perp
 \end{aligned}$$

Program Equivalence

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

- $\mathbf{C}: \text{Store}_{\perp} \rightarrow \text{Store}_{\perp}$
- Show $\mathbf{C}[[C_1]]_s = \mathbf{C}[[C_2]]_s$ for every s .
- $\mathbf{C}[[C_1]]_{\perp} = \perp = \mathbf{C}[[C_2]]_{\perp}$.

Assume proper store s .

$$\begin{aligned} & \mathbf{C}[[X:=0; Y:=X+1]]_s \\ &= \mathbf{C}[[Y:=X+1]](\mathbf{C}[[X:=0]]_s) \\ &= \mathbf{C}[[Y:=X+1]]([\![X]\!] \mapsto \text{zero}]_s) \\ &= \text{update } [\![Y]\!] (\mathbf{E}[[X+1]]([\![X]\!] \mapsto \text{zero}]_s)) \\ & \quad [\![X]\!] \mapsto \text{zero}]_s \\ &= \text{update } [\![Y]\!] \text{ one } [\![X]\!] \mapsto \text{zero}]_s \\ &= [\![Y]\!] \mapsto \text{one } [\![X]\!] \mapsto \text{zero}]_s \\ &= s_1 \end{aligned}$$

Program Equivalence

$$\begin{aligned}
 & \mathbf{C}[[Y:=1; X:=0]]s \\
 &= \mathbf{C}[[X:=0]](\mathbf{C}[[Y:=1]]s) \\
 &= \mathbf{C}[[X:=0]]([\![Y]\!] \mapsto \mathit{one}]s) \\
 &= [\![X]\!] \mapsto \mathit{zero}] [\![Y]\!] \mapsto \mathit{one}]s = s_2
 \end{aligned}$$

- $s_1 \stackrel{?}{=} s_2$.

- $s_1, s_2: Id \rightarrow Nat$

- Show $s_1(id) = s_2(id)$ for every id .

1. $id = \llbracket X \rrbracket$: $s_1[\llbracket X \rrbracket] = ([\![Y]\!] \mapsto \mathit{one}] [\![X]\!] \mapsto \mathit{zero}]s)[\llbracket X \rrbracket]$
 $= ([\![X]\!] \mapsto \mathit{zero}]s)[\llbracket X \rrbracket] = \mathit{zero} = ([\![X]\!] \mapsto \mathit{zero}] [\![Y]\!] \mapsto \mathit{one}]s)[\llbracket X \rrbracket] = s_2[\llbracket X \rrbracket]$.

2. $id = \llbracket Y \rrbracket$: $s_1[\llbracket Y \rrbracket] = ([\![Y]\!] \mapsto \mathit{one}] [\![X]\!] \mapsto \mathit{zero}]s)[\llbracket Y \rrbracket]$
 $= \mathit{one} = ([\![Y]\!] \mapsto \mathit{one}]s)[\llbracket Y \rrbracket] = ([\![X]\!] \mapsto \mathit{zero}] [\![Y]\!] \mapsto \mathit{one}]s)[\llbracket Y \rrbracket] = s_2[\llbracket Y \rrbracket]$.

3. $id = \llbracket I \rrbracket$: $s_1[\llbracket I \rrbracket] = ([\![Y]\!] \mapsto \mathit{one}] [\![X]\!] \mapsto \mathit{zero}]s)[\llbracket I \rrbracket]$
 $= ([\![X]\!] \mapsto \mathit{zero}]s)[\llbracket I \rrbracket] = s[\llbracket I \rrbracket] = ([\![Y]\!] \mapsto \mathit{one}]s)[\llbracket I \rrbracket]$
 $= ([\![X]\!] \mapsto \mathit{zero}] [\![Y]\!] \mapsto \mathit{one}]s)[\llbracket I \rrbracket] = s_2[\llbracket I \rrbracket]$.

Programs Are Functions

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

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

↓ **P**

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

Study denotation without sample input!

Programs are Functions

$$\begin{array}{l} \parallel \\ \lambda n. \text{let } s_1 = ([[A] \mapsto n] \text{newstore}) \\ \quad s_2 = [[Z] \mapsto \text{one}] s_1 \\ \quad s_3 = (n \text{ equals zero}) \rightarrow \perp [] s_2 \\ \quad s' = [[Z] \mapsto \text{three}] s_3 \\ \text{in access } [Z] s' \end{array}$$

$$\begin{array}{l} \parallel \\ \lambda n. \text{let } s_1 = ([[A] \mapsto n] \text{newstore}) \\ \quad s_2 = [[Z] \mapsto \text{one}] s_1 \\ \quad s_3 = (n \text{ equals zero}) \rightarrow \perp [] s_2 \\ \quad s' = (n \text{ equals zero}) \rightarrow \perp [] [[Z] \mapsto \text{three}] s_3 \\ \text{in access } [Z] s' \end{array}$$

$$\begin{array}{l} \parallel \\ \lambda n. \text{let } s_1 = ([[A] \mapsto n] \text{newstore}) \\ \quad s_2 = [[Z] \mapsto \text{one}] s_1 \\ \text{in } (n \text{ equals zero}) \rightarrow \perp \\ \quad [] \text{access } [Z] [[Z] \mapsto \text{three}] s_2 \end{array}$$

$$\parallel \\ \lambda n. (n \text{ equals zero}) \rightarrow \perp [] \text{three}$$

Program Denotation

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

⇓ **P**

```
 $\lambda n.(n \text{ equals zero}) \rightarrow \perp [] \text{ three}$ 
```

- 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 optimization.