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 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*
- $\mathbf{C}$: 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 (codomain 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.

  P: Program $\rightarrow$ *Nat* $\rightarrow$ $Nat_{\perp}$
- 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
$= (\underline{\lambda}s.\mathbf{C}[[\text{Z:=3}]](\mathbf{C}[[\textbf{if A=0 then diverge}]]s)s_2$
$= (\underline{\lambda}s.\mathbf{C}[[\text{Z:=3}]]((\underline{\lambda}s.\mathbf{B}[[\text{A=0}]]s \rightarrow$
$\quad \mathbf{C}[[\textbf{diverge}]]s\ []\ s)\ s))s_2$
$= \mathbf{C}[[\text{Z:=3}]]((\underline{\lambda}s.\mathbf{B}[[\text{A=0}]]s \rightarrow$
$\quad \mathbf{C}[[\textbf{diverge}]]s\ []\ s)s_2)$
$= \mathbf{C}[[\text{Z:=3}]](\underline{\mathbf{B}[[\text{A=0}]]s_2} \rightarrow \mathbf{C}[[\textbf{diverge}]]s_2\ []\ s_2)$

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

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

Simplification

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

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

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.
```

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

let s_1 = ([[[A]] $\mapsto$ *two*] *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'

$\Vert$

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 = **C**[[**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 = $\underline{\textbf{C}[[\textbf{if A=0 then diverge}]]s_4}$
s' = $\textbf{C}[[\text{Z:=3}]]s_5$
in *access* [[Z]] s'
= let s_3 = [[[A]] $\mapsto$ *zero*]*newstore*
s_4 = [[[Z]] $\mapsto$ *one*]s_3
s_5 = $\bot$
s' = $\textbf{C}[[\text{Z:=3}]]s_5$
in *access* [[Z]] s'
= let s' = $\textbf{C}[[\text{Z:=3}]]\bot$
in *access* [[Z]] s'
= let s' = $(\underline{\lambda}s.\mathit{update}\ [[\text{Z}]]\ (\textbf{E}[[3]]s))\bot$
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 $\mathbf{C}[\![\text{C}_1]\!]s = \mathbf{C}[\![\text{C}_2]\!]s$ for every s.
- $\mathbf{C}[\![\text{C}_1]\!]\perp = \perp = \mathbf{C}[\![\text{C}_2]\!]\perp$.

Assume proper store s.

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

Program Equivalence

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

- $s_1 \stackrel{?}{=} s_2$.
- s_1, s_2: $\mathit{Id} \to \mathit{Nat}$
- Show $s_1(\mathit{id}) = s_2(\mathit{id})$ for every id.
  1. $\underline{\mathit{id} = [\![\text{X}]\!]}$: $s_1[\![\text{X}]\!] = ([\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ][\ [\![\text{X}]\!] \mapsto \mathit{zero}\ ]s)[\![\text{X}]\!] = ([\![\text{X}]\!] \mapsto \mathit{zero}\ ]s)[\![\text{X}]\!] = \mathit{zero} = ([\ [\![\text{X}]\!] \mapsto \mathit{zero}\ ][\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ]s)[\![\text{X}]\!] = s_2[\![\text{X}]\!]$.
  2. $\underline{\mathit{id} = [\![\text{Y}]\!]}$: $s_1[\![\text{Y}]\!] = ([\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ][\ [\![\text{X}]\!] \mapsto \mathit{zero}\ ]s)[\![\text{Y}]\!] = \mathit{one} = ([\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ]s)[\![\text{Y}]\!] = ([\ [\![\text{X}]\!] \mapsto \mathit{zero}\ ][\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ]s)[\![\text{Y}]\!] = s_2[\![\text{Y}]\!]$.
  3. $\underline{\mathit{id} = [\![\text{I}]\!]}$: $s_1[\![\text{I}]\!] = ([\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ][\ [\![\text{X}]\!] \mapsto \mathit{zero}\ ]s)[\![\text{I}]\!] = ([\ [\![\text{X}]\!] \mapsto \mathit{zero}\ ]s)[\![\text{I}]\!] = s[\![\text{I}]\!] = ([\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ]s)[\![\text{I}]\!] = ([\ [\![\text{X}]\!] \mapsto \mathit{zero}\ ][\ [\![\text{Y}]\!] \mapsto \mathit{one}\ ]s)[\![\text{I}]\!] = s_2[\![\text{I}]\!]$.

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 $s_1 = (\ [\ [[\mathsf{A}]] \mapsto n]$ *newstore*$)$
$\quad s_2 = [\ [[\mathsf{Z}]] \mapsto$ *one* $]s_1$
$\quad s_3 = ((\mathit{access}\ [[\mathsf{A}]]\ s_2)\ \mathit{equals\ zero}) \to \bot\ []\ s_2$
$\quad s' = [\ [[\mathsf{Z}]] \mapsto$ *three* $]s_3$
in *access* $[[\mathsf{Z}]]\ s'$

Study denotation without sample input!

Programs are Functions

$$\|$$

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

$$\|$$

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

$$\|$$

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

$$\|$$

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

Program Denotation

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

$\Downarrow$ **P**

$\lambda n.(n$ *equals zero*$) \rightarrow \perp$ [] *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.