

Compiler Construction

Lecture 17: Code Generation II (Semantics & Intermediate Code)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University
`noll@cs.rwth-aachen.de`

`http://www-i2.informatik.rwth-aachen.de/i2/cc08/`

Summer semester 2008

- 1 Repetition: The Example Programming Language EPL
- 2 Dynamic Semantics of EPL
- 3 Intermediate Code for EPL

Definition (Syntax of EPL)

The **syntax of EPL** is defined as follows:

$\mathbb{Z} :$ z (* z is an integer *)

$Ide :$ I (* I is an identifier *)

$AExp :$ $A ::= z \mid I \mid A_1 + A_2 \mid \dots$

$BExp :$ $B ::= A_1 < A_2 \mid \text{not } B \mid B_1 \text{ and } B_2 \mid B_1 \text{ or } B_2$

$Cmd :$ $C ::= I := A \mid C_1; C_2 \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid$
 $\text{while } B \text{ do } C \mid I()$

$Dcl :$ $D ::= D_C D_V D_P$
 $D_C ::= \varepsilon \mid \text{const } I_1 := z_1, \dots, I_n := z_n;$
 $D_V ::= \varepsilon \mid \text{var } I_1, \dots, I_n;$
 $D_P ::= \varepsilon \mid \text{proc } I_1; K_1; \dots; I_n; K_n;$

$Block :$ $K ::= D C$

$Pgm :$ $P ::= \text{in/out } I_1, \dots, I_n; K.$

- All identifiers in a declaration D have to be **different**.
- Every identifier occurring in the command C of a block D must be **declared**
 - in D or
 - in the declaration list of a surrounding block.
- **Multiple declarations** of an identifier in different blocks are possible. Each usage in a command C refers to the “**innermost**” **declaration**.
- **Static scoping**: the usage of an identifier in the body of a called procedure refers to its declaration environment (and not to its calling environment).

Example

```
in/out x;  
  const c = 10;  
  var y;  
  proc P;  
    var y, z;  
    proc Q;  
      var x, z;  
      [... z := 1; P() ...]  
      [... P() ... R() ...]  
    proc R;  
      [... P() ...]  
      [... x := 0; P() ...] .
```

Example

```
in/out x;  
const c = 10;  
var y;  
proc P;  
  var y, z;  
  proc Q;  
    var x, z;  
    [... z := 1; P() ...]  
  [... P() ... R() ...]  
proc R;  
  [... P() ...]  
[... x := 0; P() ...] .
```

- “Innermost” principle

Example

```
in/out x;  
const c = 10;  
var y;  
proc P;  
  var y, z;  
  proc Q;  
    var x, z;  
    [... z := 1; P() ...]  
  [... P() ... R() ...]  
proc R;  
  [... P() ...]  
[... x := 0; P() ...] .
```

- “Innermost” principle

Example

```
in/out x;  
const c = 10;  
var y;  
proc P;  
  var y, z;  
  proc Q;  
    var x, z;  
    [... z := 1; P() ...]  
  [... P() ... R() ...]  
proc R;  
  [... P() ...]  
[... x := 0; P() ...] .
```

- “Innermost” principle
- **Static scoping**: body of P can refer to **x, y, z**

Example

```
in/out x;  
const c = 10;  
var y;  
proc P;  
  var y, z;  
  proc Q;  
    var x, z;  
    [... z := 1; P() ...]  
    [... P() ... R() ...]  
  proc R;  
    [... P() ...]  
  [... x := 0; P() ...] .
```

- “Innermost” principle
- Static scoping: body of P can refer to x, y, z
- **Later declaration**: call of R in P followed by declaration (in Pascal: forward declarations for one-pass compilation)

- 1 Repetition: The Example Programming Language EPL
- 2 Dynamic Semantics of EPL
- 3 Intermediate Code for EPL

Example 17.1

```
in/out x, y;  
  x := 1;  
  while x * x < y do  
    x := x + 1.
```

- Without nested declarations:

Program state : Variables $\dashrightarrow$ Values

Meaning of Variables

- Without nested declarations:

Program state : Variables $\rightarrow$ Values

- With nested declarations: consider

```
in/out x;  
var y;  
proc P;  
    var x;  
    y := 1;  
P(); x := x + y.
```

- value of I/O variable x **must not be overwritten** while calling P
 $\Rightarrow$ introduce intermediate level of **memory locations** and **environments**:

Environment : Variables $\rightarrow$ Locations
Program state : Locations $\rightarrow$ Values

Semantic Domains

- Memory addresses (**locations**) store integer values ($\Rightarrow$ **states**)
- **Variable** identifiers refer to locations
- **Constant** identifiers refer to integers
- **Procedure** identifiers refer to state transformations
- **Commands** transform states
- **Declarations** determine identifier references (**environments**)

Semantic Domains

- Memory addresses (**locations**) store integer values ($\Rightarrow$ **states**)
- **Variable** identifiers refer to locations
- **Constant** identifiers refer to integers
- **Procedure** identifiers refer to state transformations
- **Commands** transform states
- **Declarations** determine identifier references (**environments**)

Definition 17.2 (Semantic domains of EPL)

The **semantic domains of EPL** are given as follows:

$\mathbb{Z} := \{0, 1, -1, \dots\}$	integer numbers
$\mathbb{B} := \{\text{true}, \text{false}\}$	Booleans
$Loc := \{\alpha_1, \alpha_2, \dots\}$	locations
$Stt := \{\sigma \mid \sigma : Loc \dashrightarrow \mathbb{Z}\}$	states
$Trn := \{\tau \mid \tau : Stt \dashrightarrow Stt\}$	state transformations
$Env := \{\rho \mid \rho : Ide \dashrightarrow \mathbb{Z} \cup Loc \cup Trn\}$	environments

Semantics of Arithmetic Expressions I

The semantics of an arithmetic expression is its integer value (or undefined).

Definition 17.3 (Semantics of arithmetic expressions)

$$\mathfrak{A} : AExp \times Env \times Stt \dashrightarrow \mathbb{Z}$$

is given by

$$\begin{aligned}\mathfrak{A}[[z]] \rho \sigma &:= z \\ \mathfrak{A}[[I]] \rho \sigma &:= \begin{cases} z & \text{if } \rho(I) = z \in \mathbb{Z} \\ \sigma(\alpha) & \text{if } \rho(I) = \alpha \in Loc \end{cases} \\ \mathfrak{A}[[A_1 + A_2]] \rho \sigma &:= \mathfrak{A}[[A_1]] \rho \sigma + \mathfrak{A}[[A_2]] \rho \sigma\end{aligned}$$

Semantics of Arithmetic Expressions I

The semantics of an arithmetic expression is its integer value (or undefined).

Definition 17.3 (Semantics of arithmetic expressions)

$$\mathfrak{A} : AExp \times Env \times Stt \dashrightarrow \mathbb{Z}$$

is given by

$$\begin{aligned}\mathfrak{A}[[z]] \rho \sigma &:= z \\ \mathfrak{A}[[I]] \rho \sigma &:= \begin{cases} z & \text{if } \rho(I) = z \in \mathbb{Z} \\ \sigma(\alpha) & \text{if } \rho(I) = \alpha \in Loc \end{cases} \\ \mathfrak{A}[[A_1 + A_2]] \rho \sigma &:= \mathfrak{A}[[A_1]] \rho \sigma + \mathfrak{A}[[A_2]] \rho \sigma\end{aligned}$$

Remark: $\mathfrak{A}[[I]] \rho \sigma$ is undefined (notation: $\mathfrak{A}[[I]] \rho \sigma = \perp$) if I is a procedure identifier, i.e., $\rho(I) \in Trn$.

Example 17.4

Let $\rho(\mathbf{x}) = \alpha_1$ and $\sigma(\alpha_1) = 1$. Then

$$\mathcal{A}[\mathbf{x} + 1] \rho \sigma$$

Example 17.4

Let $\rho(\mathbf{x}) = \alpha_1$ and $\sigma(\alpha_1) = 1$. Then

$$\mathfrak{A}[\mathbf{x} + 1] \rho \sigma = \mathfrak{A}[\mathbf{x}] \rho \sigma + \mathfrak{A}[1] \rho \sigma$$

Example 17.4

Let $\rho(\mathbf{x}) = \alpha_1$ and $\sigma(\alpha_1) = 1$. Then

$$\begin{aligned}\mathfrak{A}[\mathbf{x} + 1] \rho \sigma &= \mathfrak{A}[\mathbf{x}] \rho \sigma + \mathfrak{A}[1] \rho \sigma \\ &= \sigma(\alpha_1) + 1\end{aligned}$$

Example 17.4

Let $\rho(\mathbf{x}) = \alpha_1$ and $\sigma(\alpha_1) = 1$. Then

$$\begin{aligned}\mathfrak{A}[\mathbf{x} + 1] \rho \sigma &= \mathfrak{A}[\mathbf{x}] \rho \sigma + \mathfrak{A}[1] \rho \sigma \\ &= \sigma(\alpha_1) + 1 \\ &= 1 + 1\end{aligned}$$

Example 17.4

Let $\rho(\mathbf{x}) = \alpha_1$ and $\sigma(\alpha_1) = 1$. Then

$$\begin{aligned}\mathcal{A}[\mathbf{x} + 1] \rho \sigma &= \mathcal{A}[\mathbf{x}] \rho \sigma + \mathcal{A}[1] \rho \sigma \\ &= \sigma(\alpha_1) + 1 \\ &= 1 + 1 \\ &= 2\end{aligned}$$

Semantics of Boolean Expressions I

The semantics of a Boolean expression is its truth value (or undefined).

Definition 17.5 (Semantics of Boolean expressions)

$$\mathfrak{B} : BExp \times Env \times Stt \dashrightarrow \mathbb{B}$$

is given by

$$\begin{aligned}\mathfrak{B}[A_1 < A_2] \rho \sigma &:= \mathfrak{A}[A_1] \rho \sigma < \mathfrak{A}[A_2] \rho \sigma \\ \mathfrak{B}[\text{not } B] \rho \sigma &:= \neg \mathfrak{B}[B] \rho \sigma \\ \mathfrak{B}[B_1 \text{ and } B_2] \rho \sigma &:= \mathfrak{B}[B_1] \rho \sigma \wedge \mathfrak{B}[B_2] \rho \sigma \\ \mathfrak{B}[B_1 \text{ or } B_2] \rho \sigma &:= \mathfrak{B}[B_1] \rho \sigma \vee \mathfrak{B}[B_2] \rho \sigma\end{aligned}$$

Semantics of Boolean Expressions I

The semantics of a Boolean expression is its truth value (or undefined).

Definition 17.5 (Semantics of Boolean expressions)

$$\mathfrak{B} : BExp \times Env \times Stt \dashrightarrow \mathbb{B}$$

is given by

$$\begin{aligned}\mathfrak{B}[A_1 < A_2] \rho \sigma &:= \mathfrak{A}[A_1] \rho \sigma < \mathfrak{A}[A_2] \rho \sigma \\ \mathfrak{B}[\text{not } B] \rho \sigma &:= \neg \mathfrak{B}[B] \rho \sigma \\ \mathfrak{B}[B_1 \text{ and } B_2] \rho \sigma &:= \mathfrak{B}[B_1] \rho \sigma \wedge \mathfrak{B}[B_2] \rho \sigma \\ \mathfrak{B}[B_1 \text{ or } B_2] \rho \sigma &:= \mathfrak{B}[B_1] \rho \sigma \vee \mathfrak{B}[B_2] \rho \sigma\end{aligned}$$

Remarks:

- $\mathfrak{B}[B] \rho \sigma$ is undefined only if the value of an arithmetic subexpression of B is undefined.
- Possible interpretations of binary operations:

Strict:

$$a \diamond \perp = \perp$$

$$\perp \diamond b = \perp$$

Sequential:

$$\text{false} \wedge \perp = \text{false}$$

$$\text{true} \vee \perp = \text{true}$$

$$\perp \diamond b = \perp$$

Non-strict:

$$\text{false} \wedge \perp = \perp \wedge \text{false} = \text{false}$$

$$\text{true} \vee \perp = \perp \vee \text{true} = \text{true}$$

Example 17.6

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\mathfrak{B}[\![x * x < y]\!] \rho \sigma$$

Example 17.6

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\mathfrak{B}[\![x * x < y]\!] \rho \sigma = \mathfrak{A}[\![x * x]\!] \rho \sigma < \mathfrak{A}[\![y]\!] \rho \sigma$$

Example 17.6

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\begin{aligned}\mathcal{B}[\![x * x < y]\!] \rho \sigma &= \mathcal{A}[\![x * x]\!] \rho \sigma < \mathcal{A}[\![y]\!] \rho \sigma \\ &= \mathcal{A}[\![x]\!] \rho \sigma \cdot \mathcal{A}[\![x]\!] \rho \sigma < \sigma(\alpha_2)\end{aligned}$$

Example 17.6

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\begin{aligned}\mathcal{B}[\![x * x < y]\!] \rho \sigma &= \mathcal{A}[\![x * x]\!] \rho \sigma < \mathcal{A}[\![y]\!] \rho \sigma \\ &= \mathcal{A}[\![x]\!] \rho \sigma \cdot \mathcal{A}[\![x]\!] \rho \sigma < \sigma(\alpha_2) \\ &= \sigma(\alpha_1) \cdot \sigma(\alpha_1) < 4\end{aligned}$$

Example 17.6

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\begin{aligned}\mathcal{B}[\![x * x < y]\!] \rho \sigma &= \mathcal{A}[\![x * x]\!] \rho \sigma < \mathcal{A}[\![y]\!] \rho \sigma \\ &= \mathcal{A}[\![x]\!] \rho \sigma \cdot \mathcal{A}[\![x]\!] \rho \sigma < \sigma(\alpha_2) \\ &= \sigma(\alpha_1) \cdot \sigma(\alpha_1) < 4 \\ &= 1 \cdot 1 < 4\end{aligned}$$

Example 17.6

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\begin{aligned}\mathcal{B}[\![x * x < y]\!] \rho \sigma &= \mathcal{A}[\![x * x]\!] \rho \sigma < \mathcal{A}[\![y]\!] \rho \sigma \\ &= \mathcal{A}[\![x]\!] \rho \sigma \cdot \mathcal{A}[\![x]\!] \rho \sigma < \sigma(\alpha_2) \\ &= \sigma(\alpha_1) \cdot \sigma(\alpha_1) < 4 \\ &= 1 \cdot 1 < 4 \\ &= \text{true}\end{aligned}$$

Semantics of Commands I

Commands modify the values of variables, i.e., transform states.

Definition 17.7 (Semantics of commands)

$$\mathfrak{C} : \text{Cmd} \times \text{Env} \times \text{Stt} \dashrightarrow \text{Stt}$$

is given by

$$\begin{aligned} \mathfrak{C}[I := A] \rho \sigma &:= \sigma[\alpha \mapsto z] \\ &\quad \text{if } \rho(I) = \alpha \in \text{Loc} \text{ and } \mathfrak{A}[A] \rho \sigma = z \in \mathbb{Z} \\ &\quad \text{where } \sigma[\alpha \mapsto z](\beta) := \begin{cases} z & \text{if } \beta = \alpha \\ \sigma(\beta) & \text{otherwise} \end{cases} \end{aligned}$$

$$\mathfrak{C}[C_1; C_2] \rho \sigma := \mathfrak{C}[C_2] \rho (\mathfrak{C}[C_1] \rho \sigma)$$

$$\mathfrak{C}[\text{if } B \text{ then } C_1 \text{ else } C_2] \rho \sigma := \begin{cases} \mathfrak{C}[C_1] \rho \sigma & \text{if } \mathfrak{B}[B] \rho \sigma \\ \mathfrak{C}[C_2] \rho \sigma & \text{if } \neg \mathfrak{B}[B] \rho \sigma \end{cases}$$

$$\mathfrak{C}[\underbrace{\text{while } B \text{ do } C}_{C'}] \rho \sigma := \begin{cases} \mathfrak{C}[C'] \rho (\mathfrak{C}[C] \rho \sigma) & \text{if } \mathfrak{B}[B] \rho \sigma \\ \sigma & \text{if } \neg \mathfrak{B}[B] \rho \sigma \end{cases}$$

$$\mathfrak{C}[I()] \rho \sigma := \tau(\sigma) \quad \text{if } \rho(I) = \tau \in \text{Trn}$$

Example 17.8

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho \sigma$$

Example 17.8

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\begin{aligned} & \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho \sigma \\ & \quad (\text{Ex. 17.6: } \mathcal{B}[x * x < y] \rho \sigma = \text{true}) \\ & = \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho (\mathcal{C}[x := x + 1] \rho \sigma) \end{aligned}$$

Example 17.8

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\begin{aligned} & \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho \sigma \\ & \quad (\text{Ex. 17.6: } \mathcal{B}[x * x < y] \rho \sigma = \text{true}) \\ &= \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho (\mathcal{C}[x := x + 1] \rho \sigma) \\ & \quad (\text{Ex. 17.4: } \mathcal{A}[x + 1] \rho \sigma = 2) \\ &= \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho (\sigma[\alpha_1 \mapsto 2]) \end{aligned}$$

Example 17.8

Let $\rho(x) = \alpha_1$, $\rho(y) = \alpha_2$, $\sigma(\alpha_1) = 1$, and $\sigma(\alpha_2) = 4$. Then

$$\begin{aligned} & \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho \sigma \\ & \quad (\text{Ex. 17.6: } \mathcal{B}[x * x < y] \rho \sigma = \text{true}) \\ &= \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho (\mathcal{C}[x := x + 1] \rho \sigma) \\ & \quad (\text{Ex. 17.4: } \mathcal{A}[x + 1] \rho \sigma = 2) \\ &= \mathcal{C}[\text{while } x * x < y \text{ do } x := x + 1] \rho (\sigma[\alpha_1 \mapsto 2]) \\ & \quad (\mathcal{B}[x * x < y] \rho \sigma[\alpha_1 \mapsto 2] = \text{false}) \\ &= \sigma[\alpha_1 \mapsto 2] \end{aligned}$$

Semantics of Declarations/Blocks (informally)

Declarations update environments:

$$\mathfrak{D} : Dcl \times Env \times Stt \dashrightarrow Env \times Stt$$

Constant declarations: bind identifiers to values

Variable declarations: bind identifiers to free locations

Procedure declarations: bind identifiers to state transformations
(which are determined with respect to the **declaration environment** of the procedure (**static scoping**))

Semantics of Declarations/Blocks (informally)

Declarations update environments:

$$\mathfrak{D} : Dcl \times Env \times Stt \dashrightarrow Env \times Stt$$

Constant declarations: bind identifiers to values

Variable declarations: bind identifiers to free locations

Procedure declarations: bind identifiers to state transformations
(which are determined with respect to the **declaration environment** of the procedure (**static scoping**))

Semantics of a block $K = D \ C$:

$$\mathfrak{K} : Block \times Env \times Stt \dashrightarrow Stt$$

- 1 Extension of the current environment according to the declarations in D
- 2 Execution of command C in the extended environment
- 3 “Release” of memory addresses allocated by D

To “run” a program, execute the main block in

- the **environment** which is determined by the I/O variables and
- the **state** which is given by the input values

Semantics of Programs I

To “run” a program, execute the main block in

- the **environment** which is determined by the I/O variables and
- the **state** which is given by the input values

Definition 17.9 (Semantics of programs)

$$\mathfrak{M} : Pgm \times \mathbb{Z}^* \dashrightarrow \mathbb{Z}^*$$

is given by

$$\mathfrak{M}[\text{in/out } I_1, \dots, I_n; K.](z_1, \dots, z_n) := (\sigma(\alpha_1), \dots, \sigma(\alpha_n))$$

where $\sigma := \mathfrak{K}[K] \rho_\emptyset[I_1 \mapsto \alpha_1, \dots, I_n \mapsto \alpha_n] \sigma_\emptyset[\alpha_1 \mapsto z_1, \dots, \alpha_n \mapsto z_n]$
and $\rho_\emptyset(I) := \sigma_\emptyset(\alpha) := \perp$ for every $I \in Ide$, $\alpha \in Loc$.

Example 17.10

For $P =$ in/out $x, y;$ we obtain:
 $x := 1;$
 while $x * x < y$ do
 $x := x + 1.$

Example 17.10

For $P =$ in/out $x, y;$ we obtain:

```
  x := 1;  
  while x * x < y do  
    x := x + 1.
```

$$\mathfrak{M}\llbracket P \rrbracket(0,4) = (\sigma(\alpha_1), \sigma(\alpha_2))$$

Example 17.10

For $P =$ in/out $x, y;$ we obtain:

```
x := 1;  
while x * x < y do  
  x := x + 1.
```

$\mathfrak{M}[P](0, 4) = (\sigma(\alpha_1), \sigma(\alpha_2))$ where

$\sigma = \mathfrak{K}[x := 1; \text{while } x * x < y \text{ do } x := x + 1]$
 $\rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]$

Example 17.10

For $P =$ in/out $x, y;$ we obtain:

```
x := 1;  
while x * x < y do  
  x := x + 1.
```

$\mathfrak{M}\llbracket P \rrbracket(0,4) = (\sigma(\alpha_1), \sigma(\alpha_2))$ where

$$\begin{aligned}\sigma &= \mathfrak{K}\llbracket x := 1; \text{ while } x * x < y \text{ do } x := x + 1 \rrbracket \\ &\quad \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4] \\ &= \mathfrak{C}\llbracket x := 1; \text{ while } x * x < y \text{ do } x := x + 1 \rrbracket \\ &\quad \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]\end{aligned}$$

Example 17.10

For $P =$ in/out $x, y;$ we obtain:

```
x := 1;  
while x * x < y do  
  x := x + 1.
```

$$\mathfrak{M}[P](0,4) = (\sigma(\alpha_1), \sigma(\alpha_2)) \quad \text{where}$$

$$\sigma = \mathfrak{K}[\![x := 1; \text{while } x * x < y \text{ do } x := x + 1]\!] \\ \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]$$

$$= \mathfrak{C}[\![x := 1; \text{while } x * x < y \text{ do } x := x + 1]\!] \\ \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]$$

$$\text{(Def. 17.7)} = \mathfrak{C}[\![\text{while } x * x < y \text{ do } x := x + 1]\!] \\ \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 1, \alpha_2 \mapsto 4]$$

Example 17.10

For $P =$ in/out $x, y;$ we obtain:

```
x := 1;  
while x * x < y do  
  x := x + 1.
```

$$\mathfrak{M}[P](0,4) = (\sigma(\alpha_1), \sigma(\alpha_2)) \text{ where}$$

$$\sigma = \mathfrak{K}[\![x := 1; \text{while } x * x < y \text{ do } x := x + 1]\!] \\ \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]$$

$$= \mathfrak{C}[\![x := 1; \text{while } x * x < y \text{ do } x := x + 1]\!] \\ \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]$$

$$\text{(Def. 17.7)} = \mathfrak{C}[\![\text{while } x * x < y \text{ do } x := x + 1]\!] \\ \rho_\emptyset[x \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_\emptyset[\alpha_1 \mapsto 1, \alpha_2 \mapsto 4]$$

$$\text{(Ex. 17.8)} = \sigma_\emptyset[\alpha_1 \mapsto 2, \alpha_2 \mapsto 4]$$

Example 17.10

For $P =$ `in/out x, y;` we obtain:

```
x := 1;
while x * x < y do
  x := x + 1.
```

$$\mathfrak{M}[P](0,4) = (\sigma(\alpha_1), \sigma(\alpha_2)) \text{ where}$$

$$\sigma = \mathfrak{K}[\text{x} := 1; \text{while } x * x < y \text{ do } x := x + 1] \\ \rho_{\emptyset}[\text{x} \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_{\emptyset}[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]$$

$$= \mathfrak{C}[\text{x} := 1; \text{while } x * x < y \text{ do } x := x + 1] \\ \rho_{\emptyset}[\text{x} \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_{\emptyset}[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4]$$

$$\text{(Def. 17.7)} = \mathfrak{C}[\text{while } x * x < y \text{ do } x := x + 1] \\ \rho_{\emptyset}[\text{x} \mapsto \alpha_1, y \mapsto \alpha_2] \sigma_{\emptyset}[\alpha_1 \mapsto 1, \alpha_2 \mapsto 4]$$

$$\text{(Ex. 17.8)} = \sigma_{\emptyset}[\alpha_1 \mapsto 2, \alpha_2 \mapsto 4]$$

$$\implies \mathfrak{M}[P](0,4) = (2,4)$$

- 1 Repetition: The Example Programming Language EPL
- 2 Dynamic Semantics of EPL
- 3 Intermediate Code for EPL

Definition 17.11 (Abstract machine for EPL)

The **abstract machine for EPL (AM)** is defined by the **state space**

$$S := PC \times DS \times PS$$

with

- the **program counter** $PC := \mathbb{N}$,
- the **data stack** $DS := \mathbb{Z}^*$ (top of stack to the right), and
- the **procedure stack** (or: **runtime stack**) $PS := \mathbb{Z}^*$ (top of stack to the left).

Thus a state $s = (l, d, p) \in S$ is given by

- a program label $l \in PC$,
- a data stack $d = d.r : \dots : d.1 \in DS$, and
- a procedure stack $p = p.1 : \dots : p.t \in PS$.

Definition 17.12 (AM instructions)

The set of **AM instructions** is divided into

arithmetic instructions: `ADD`, `MULT`, ...

Boolean instructions: `NOT`, `AND`, `OR`, `LT`, ...

jumping instructions: `JMP(ca)`, `JFALSE(ca)` ($ca \in PC$)

procedure instructions: `CALL(ca, dif, loc)` ($ca \in PC$, $dif, loc \in \mathbb{N}$), `RET`

transfer instructions: `LOAD(dif, off)`, `STORE(dif, off)` ($dif, off \in \mathbb{N}$),
`LIT(z)` ($z \in \mathbb{Z}$)

Definition 17.13 (Semantics of AM instructions (1st part))

The semantics of an AM instruction O

$$\llbracket O \rrbracket : S \dashrightarrow S$$

is defined as follows:

$$\begin{aligned}\llbracket \text{ADD} \rrbracket(l, d : z_1 : z_2, p) &:= (l + 1, d : z_1 + z_2, p) \\ \llbracket \text{NOT} \rrbracket(l, d : b, p) &:= (l + 1, d : \neg b, p) && \text{if } b \in \{0, 1\} \\ \llbracket \text{AND} \rrbracket(l, d : b_1 : b_2, p) &:= (l + 1, d : b_1 \wedge b_2, p) && \text{if } b_1, b_2 \in \{0, 1\} \\ \llbracket \text{OR} \rrbracket(l, d : b_1 : b_2, p) &:= (l + 1, d : b_1 \vee b_2, p) && \text{if } b_1, b_2 \in \{0, 1\} \\ \llbracket \text{LT} \rrbracket(l, d : z_1 : z_2, p) &:= \begin{cases} (l + 1, d : 1, p) & \text{if } z_1 < z_2 \\ (l + 1, d : 0, p) & \text{if } z_1 \geq z_2 \end{cases} \\ \llbracket \text{JMP}(ca) \rrbracket(l, d, p) &:= (ca, d, p) \\ \llbracket \text{JFALSE}(ca) \rrbracket(l, d : b, p) &:= \begin{cases} (ca, d, p) & \text{if } b = 0 \\ (l + 1, d, p) & \text{if } b = 1 \end{cases}\end{aligned}$$