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

Outline

Syntax of EPL

Definition (Syntax of EPL)

The syntax of EPL is defined as follows:

$$
\begin{array}{lll}
\mathbb{Z}: & z & (* \; z \text{ is an integer } *) \\
Ide: & I & (* \; I \text{ is an identifier } *) \\
AExp: & A ::= z \mid I \mid A_1 \texttt{ + } A_2 \mid \ldots & \\
BExp: & B ::= A_1 \texttt{ < } A_2 \mid \texttt{not } B \mid B_1 \texttt{ and } B_2 \mid B_1 \texttt{ or } B_2 & \\
Cmd: & C ::= I \texttt{ := } A \mid C_1 \texttt{;} C_2 \mid \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \mid & \\
 & \quad \texttt{while } B \texttt{ do } C \mid I \texttt{()} & \\
Dcl: & D ::= D_C \; D_V \; D_P & \\
 & D_C ::= \varepsilon \mid \texttt{const } I_1 \texttt{ := } z_1 \texttt{,} \ldots \texttt{,} I_n \texttt{ := } z_n \texttt{;} & \\
 & D_V ::= \varepsilon \mid \texttt{var } I_1 \texttt{,} \ldots \texttt{,} I_n \texttt{;} & \\
 & D_P ::= \varepsilon \mid \texttt{proc } I_1 \texttt{;} K_1 \texttt{;} \ldots \texttt{;} I_n \texttt{;} K_n \texttt{;} & \\
Block: & K ::= D \; C & \\
Pgm: & P ::= \texttt{in/out } I_1 \texttt{,} \ldots \texttt{,} I_n \texttt{;} K \texttt{.} &
\end{array}
$$

Static Semantics of EPL I

- All identifiers in a declaration D have to be different.
- Every identifier occurring in the command C of a block $D\ C$ 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).

Static Semantics of EPL II

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)

Outline

Running Example

Example 17.1

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

Meaning of Variables

- **Without nested declarations:**

$$\text{Program state} : \text{Variables} \dashrightarrow \text{Values}$$

- **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`

$\Longrightarrow$ introduce intermediate level of memory locations and environments:

$$\text{Environment} : \text{Variables} \dashrightarrow \text{Locations}$$
$$\text{Program state} : \text{Locations} \dashrightarrow \text{Values}$$

Semantic Domains

- Memory addresses (locations) store integer values ($\Longrightarrow$ 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:

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

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

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

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

Semantics of Arithmetic Expressions II

Example 17.4

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

$$\begin{aligned}
\mathfrak{A}[\![\mathtt{x + 1}]\!]\,\rho\,\sigma &= \mathfrak{A}[\![\mathtt{x}]\!]\,\rho\,\sigma + \mathfrak{A}[\![\mathtt{1}]\!]\,\rho\,\sigma \\
&= \sigma(\alpha_1) + 1 \\
&= 1 + 1 \\
&= 2
\end{aligned}$$

Semantics of Boolean Expressions I

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

Definition 17.5 (Semantics of Boolean expressions)

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

is given by

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

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:
  $a \Diamond \bot = \bot$
  $\bot \Diamond b = \bot$

  Sequential:
  $\mathsf{false} \wedge \bot = \mathsf{false}$
  $\mathsf{true} \vee \bot = \mathsf{true}$
  $\bot \Diamond b = \bot$

  Non-strict:
  $\mathsf{false} \wedge \bot = \bot \wedge \mathsf{false} = \mathsf{false}$
  $\mathsf{true} \vee \bot = \bot \vee \mathsf{true} = \mathsf{true}$

Semantics of Boolean Expressions II

Example 17.6

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

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

Semantics of Commands I

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

Definition 17.7 (Semantics of commands)

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

is given by

$$\mathfrak{C}[\![I \texttt{ := } A]\!]\ \rho\ \sigma := \sigma[\alpha \mapsto z]$$
$$\text{if } \rho(I) = \alpha \in \mathit{Loc} \text{ and } \mathfrak{A}[\![A]\!]\ \rho\ \sigma = z \in \mathbb{Z}$$
$$\text{where } \sigma[\alpha \mapsto z](\beta) := \begin{cases} z & \text{if } \beta = \alpha \\ \sigma(\beta) & \text{otherwise} \end{cases}$$
$$\mathfrak{C}[\![C_1 \,;\! C_2]\!]\ \rho\ \sigma := \mathfrak{C}[\![C_2]\!]\ \rho\ (\mathfrak{C}[\![C_1]\!]\ \rho\ \sigma)$$
$$\mathfrak{C}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!]\ \rho\ \sigma := \begin{cases} \mathfrak{C}[\![C_1]\!]\ \rho\ \sigma & \text{if } \mathfrak{B}[\![B]\!]\ \rho\ \sigma \\ \mathfrak{C}[\![C_2]\!]\ \rho\ \sigma & \text{if } \neg\mathfrak{B}[\![B]\!]\ \rho\ \sigma \end{cases}$$
$$\mathfrak{C}[\![\underbrace{\texttt{while } B \texttt{ do } C}_{C'}]\!]\ \rho\ \sigma := \begin{cases} \mathfrak{C}[\![C']\!]\ \rho\ (\mathfrak{C}[\![C]\!]\ \rho\ \sigma) & \text{if } \mathfrak{B}[\![B]\!]\ \rho\ \sigma \\ \sigma & \text{if } \neg\mathfrak{B}[\![B]\!]\ \rho\ \sigma \end{cases}$$
$$\mathfrak{C}[\![I\,()]\!]\ \rho\ \sigma := \tau(\sigma) \quad \text{if } \rho(I) = \tau \in \mathit{Trn}$$

Example 17.8

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

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

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 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

Definition 17.9 (Semantics of programs)

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

is given by

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

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

Example 17.10

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

$$
\begin{aligned}
\mathfrak{M}[\![P]\!](0,4) &= (\sigma(\alpha_1), \sigma(\alpha_2)) \quad \text{where} \\
\sigma &= \mathfrak{K}[\![\texttt{x := 1; while x * x < y do x := x + 1}]\!] \\
&\quad \rho_\emptyset[\texttt{x} \mapsto \alpha_1, \texttt{y} \mapsto \alpha_2]\, \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4] \\
&= \mathfrak{C}[\![\texttt{x := 1; while x * x < y do x := x + 1}]\!] \\
&\quad \rho_\emptyset[\texttt{x} \mapsto \alpha_1, \texttt{y} \mapsto \alpha_2]\, \sigma_\emptyset[\alpha_1 \mapsto 0, \alpha_2 \mapsto 4] \\
\text{(Def. 17.7)} &= \mathfrak{C}[\![\texttt{while x * x < y do x := x + 1}]\!] \\
&\quad \rho_\emptyset[\texttt{x} \mapsto \alpha_1, \texttt{y} \mapsto \alpha_2]\, \sigma_\emptyset[\alpha_1 \mapsto 1, \alpha_2 \mapsto 4] \\
\text{(Ex. 17.8)} &= \sigma_\emptyset[\alpha_1 \mapsto 2, \alpha_2 \mapsto 4]
\end{aligned}
$$

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

Outline

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 $d = d.r : \ldots : d.1 \in DS$, and
- a procedure stack $p = p.1 : \ldots : p.t \in PS$.

AM Instructions

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

$$[\![O]\!] : S \dashrightarrow S$$

is defined as follows:

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