Compiler Construction

Lecture 20: Code Generation V (Jumping Code & Procedure Parameters)

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

Translation of EPL into AM Programs

Translation mapping trans : $Pgm \dashrightarrow AM$ defined using

- symbol table:

$$Tab := \{\mathrm{st} \mid \mathrm{st} : Ide \dashrightarrow (\{\texttt{const}\} \times \mathbb{Z}) \cup (\{\texttt{var}\} \times Lev \times Off) \cup (\{\texttt{proc}\} \times PC \times Lev \times Size)\}$$

- update : $Dcl \times Tab \times Lev \dashrightarrow Tab$
  for extending the symbol table by declarations
- kt : $Block \times Tab \times PC \times Lev \dashrightarrow AM$ for translating blocks
- dt : $Dcl \times Tab \times Lev \dashrightarrow AM$
  for translating the bodies of procedure declarations
- ct : $Cmd \times Tab \times PC \times Lev \dashrightarrow AM$
  for translating commands
- bt : $BExp \times Tab \times PC \times Lev \dashrightarrow AM$
  for translating Boolean expressions
- at : $AExp \times Tab \times PC \times Lev \dashrightarrow AM$
  for translating arithmetic expressions

Correctness of the Translation

Theorem (Correctness of translation)

For every $P \in Pgm$, $n \in \mathbb{N}$, *and* $(z_1, \ldots, z_n), (z'_1, \ldots, z'_n) \in \mathbb{Z}^n$:

$$\begin{aligned} & \mathfrak{M}[\![P]\!](z_1, \ldots, z_n) = (z'_1, \ldots, z'_n) \\ \iff & [\![\text{trans}(P)]\!](1, \varepsilon, 0 : 0 : 0 : z_1 : \ldots : z_n) = (0, \varepsilon, 0 : 0 : 0 : z'_1 : \ldots : z'_n) \end{aligned}$$

Proof.

see M. Mohnen: *A Compiler Corectness Proof for the Static Link Technique by means of Evolving Algebras*, Fundamenta Informaticae 29(3), 1997, pp. 257–303 □

Outline

Boolean Expressions with Sequential Semantics

So far: Boolean expressions with strict semantics

$$b_1 \diamondsuit \bot = \bot$$

$$\bot \diamondsuit b_2 = \bot$$

Boolean Expressions with Sequential Semantics

So far: Boolean expressions with strict semantics

$$b_1 \diamondsuit \bot = \bot$$
$$\bot \diamondsuit b_2 = \bot$$

Now: Boolean expressions with sequential semantics

$$\mathsf{false} \wedge \bot = \mathsf{false}$$
$$\mathsf{true} \vee \bot = \mathsf{true}$$
$$\bot \diamondsuit b = \bot$$

Boolean Expressions with Sequential Semantics

So far: Boolean expressions with strict semantics

$$b_1 \diamondsuit \bot = \bot$$
$$\bot \diamondsuit b_2 = \bot$$

Now: Boolean expressions with sequential semantics

$$\mathsf{false} \wedge \bot = \mathsf{false}$$
$$\mathsf{true} \vee \bot = \mathsf{true}$$
$$\bot \diamondsuit b = \bot$$

Idea:

- employ jump rather than Boolean instructions ("jumping code")
- equip bt and ct with two additional address parameters:

  a_t: target address for true

  a_f: target address for false

Jumping Code for Boolean Expressions

Definition 20.1 (Jumping code for Boolean expressions)

The mapping

$$\text{sbt} : \mathit{BExp} \times \mathit{Tab} \times \mathit{PC}^3 \times \mathit{Lev} \dashrightarrow \mathit{AM}$$

is defined by

$$\begin{aligned}
\text{sbt}(A_1 \texttt{ < } A_2, \text{st}, a, a_t, a_f, l) &:= \text{at}(A_1, \text{st}, a, l) \\
&\quad \text{at}(A_2, \text{st}, a', l) \\
&\quad a'' : \texttt{LT;} \\
&\quad a'' + 1 : \texttt{JFALSE(}a_f\texttt{);} \\
&\quad a'' + 2 : \texttt{JMP(}a_t\texttt{);} \\
\text{sbt}(\texttt{not } B, \text{st}, a, a_t, a_f, l) &:= \text{sbt}(B, \text{st}, a, a_f, a_t, l) \\
\text{sbt}(B_1 \texttt{ and } B_2, \text{st}, a, a_t, a_f, l) &:= \text{sbt}(B_1, \text{st}, a, a', a_f, l) \\
&\quad \text{sbt}(B_2, \text{st}, a', a_t, a_f, l) \\
\text{sbt}(B_1 \texttt{ or } B_2, \text{st}, a, a_t, a_f, l) &:= \text{sbt}(B_1, \text{st}, a, a_t, a', l) \\
&\quad \text{sbt}(B_2, \text{st}, a', a_t, a_f, l)
\end{aligned}$$

Jumping Code for Commands

Definition 20.2 (Jumping code for commands)

The mapping

$$\text{sct} : \mathit{Cmd} \times \mathit{Tab} \times \mathit{PC} \times \mathit{Lev} \dashrightarrow \mathit{AM}$$

is defined by

$$\begin{aligned}
\text{sct}(\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, \text{st}, a, l) := \; & \text{sbt}(B, \text{st}, a, a_t, a_f, l) \\
& \text{sct}(C_1, \text{st}, a_t, l) \\
& a_f - 1 : \texttt{JMP(}a'\texttt{);} \\
& \text{sct}(C_2, \text{st}, a_f, l) \\
& a' : \\
\text{sct}(\texttt{while } B \texttt{ do } C, \text{st}, a, l) := \; & \text{sbt}(B, \text{st}, a, a_t, a_f, l) \\
& \text{sct}(C, \text{st}, a_t, l) \\
& a_f - 1 : \texttt{JMP(}a\texttt{);} \\
& a_f :
\end{aligned}$$

(remaining cases analogously)

Example: Jumping Code

Example 20.3

Translation of `while not (x < 1) and (x < y) do` C:

Example: Jumping Code

Example 20.3

Translation of `while not (x < 1) and (x < y) do` C:

Strict:

```
1 : LOAD(x);
    LIT(1);
    LT;
    NOT;
    LOAD(x);
    LOAD(y);
    LT;
    AND;
    JFALSE(a);
    ct(C, ...)
    JMP(1);
a : ...
```

Example: Jumping Code

Example 20.3

Translation of `while not (x < 1) and (x < y) do` C:

Strict:

```
1 : LOAD(x);
    LIT(1);
    LT;
    NOT;
    LOAD(x);
    LOAD(y);
    LT;
    AND;
    JFALSE(a);
    ct(C, ...)
    JMP(1);
a : ...
```

Sequential:

```
 1 : LOAD(x);
     LIT(1);
     LT;
     JFALSE(6);
     JMP(a);
 6 : LOAD(x);
     LOAD(y);
     LT;
     JFALSE(a);
     JMP(11);
11 : sct(C, ...)
     JMP(1);
 a : ...
```

Example: Jumping Code

Example 20.3

Translation of `while not (x < 1) and (x < y) do` C:

Strict:

```
1 : LOAD(x);
    LIT(1);
    LT;
    NOT;
    LOAD(x);
    LOAD(y);
    LT;
    AND;
    JFALSE(a);
    ct(C, ...)
    JMP(1);
a : ...
```

Sequential:

```
 1 : LOAD(x);
     LIT(1);
     LT;
     JFALSE(6);
     JMP(a);
 6 : LOAD(x);
     LOAD(y);
     LT;
     JFALSE(a);
     JMP(11);
11 : sct(C, ...)
     JMP(1);
 a : ...
```

If `x` $= 0$:

9 instructions executed

Example: Jumping Code

Example 20.3

Translation of `while not (x < 1) and (x < y) do` C:

Strict:

```
1 : LOAD(x);
    LIT(1);
    LT;
    NOT;
    LOAD(x);
    LOAD(y);
    LT;
    AND;
    JFALSE(a);
    ct(C, ...)
    JMP(1);
a : ...
```

If `x` $= 0$:
9 instructions executed

Sequential:

```
1 : LOAD(x);
    LIT(1);
    LT;
    JFALSE(6);
    JMP(a);
6 : LOAD(x);
    LOAD(y);
    LT;
    JFALSE(a);
    JMP(11);
11 : sct(C, ...)
    JMP(1);
a : ...
```

If `x` $= 0$:
5 instructions executed

Example: Jumping Code

Example 20.3

Translation of `while not (x < 1) and (x < y) do` C:

Strict:

```
1 : LOAD(x);
    LIT(1);
    LT;
    NOT;
    LOAD(x);
    LOAD(y);
    LT;
    AND;
    JFALSE(a);
    ct(C, ...)
    JMP(1);
a : ...
```

If `x` $= 0$:
9 instructions executed

Sequential:

```
 1 : LOAD(x);
     LIT(1);
     LT;
     JFALSE(6);
     JMP(a);
 6 : LOAD(x);
     LOAD(y);
     LT;
     JFALSE(a);
     JMP(11);
11 : sct(C, ...)
     JMP(1);
 a : ...
```

If `x` $= 0$:
5 instructions executed

$\Longrightarrow$ generally: longer code, but shorter executions

Outline

Extended Syntax of EPL

Definition 20.4 (Extended syntax of EPL)

The extended syntax of EPL is defined as follows:

$$\vdots$$

$$Cmd: \quad C ::= I := A \mid C_1 ; C_2 \mid \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \mid \texttt{while } B \texttt{ do } C \mid I(\underbrace{A_1, \ldots, A_p ; V_1, \ldots, V_q}_{\text{actual value and ref. parameters}})$$

$$\begin{aligned} Dcl: \quad D &::= D_C\ D_V\ D_P \\ D_C &::= \varepsilon \mid \texttt{const } I_1 := z_1, \ldots, I_n := z_n ; \\ D_V &::= \varepsilon \mid \texttt{var } I_1, \ldots, I_n ; \\ D_P &::= \varepsilon \mid \\ &\quad \texttt{proc } I(\underbrace{I_1, \ldots, I_p ; \texttt{ var } J_1, \ldots, J_q}_{\text{formal value and ref. parameters}}) ; K ; \texttt{ proc} \ldots \end{aligned}$$

$$\vdots$$

where $I_k, J_k, V_k \in Ide$

Extended Semantics of EPL

Procedure declaration:

- all formal parameters different and disjoint from local variables
- use in procedure body like variables (read/write access)

Extended Semantics of EPL

Procedure declaration:

- all formal parameters different and disjoint from local variables
- use in procedure body like variables (read/write access)

Procedure call:

- formal value parameters implemented like local variables (new memory locations)

$\Longrightarrow$ assignments have no effect on calling site

- only variables as formal reference parameters (all different)
- implemented by pointer to corresponding actual address

$\Longrightarrow$ assignments have effect on calling site (return values)

The Extended Abstract Machine AM

Definition 20.5 (Extended abstract machine for EPL)

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

$$S := PC \times SP \times FP \times IR \times RS$$

with

- the program counter $PC := \mathbb{N}$,
- the stack pointer $SP := \mathbb{N}$,
- the frame pointer $FP := \mathbb{N}$,
- the index register $IR := \mathbb{N}$, and
- the runtime stack $RS := (\mathbb{N} \to \mathbb{Z})$.

The Extended Abstract Machine AM

Definition 20.5 (Extended abstract machine for EPL)

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

$$S := PC \times SP \times FP \times IR \times RS$$

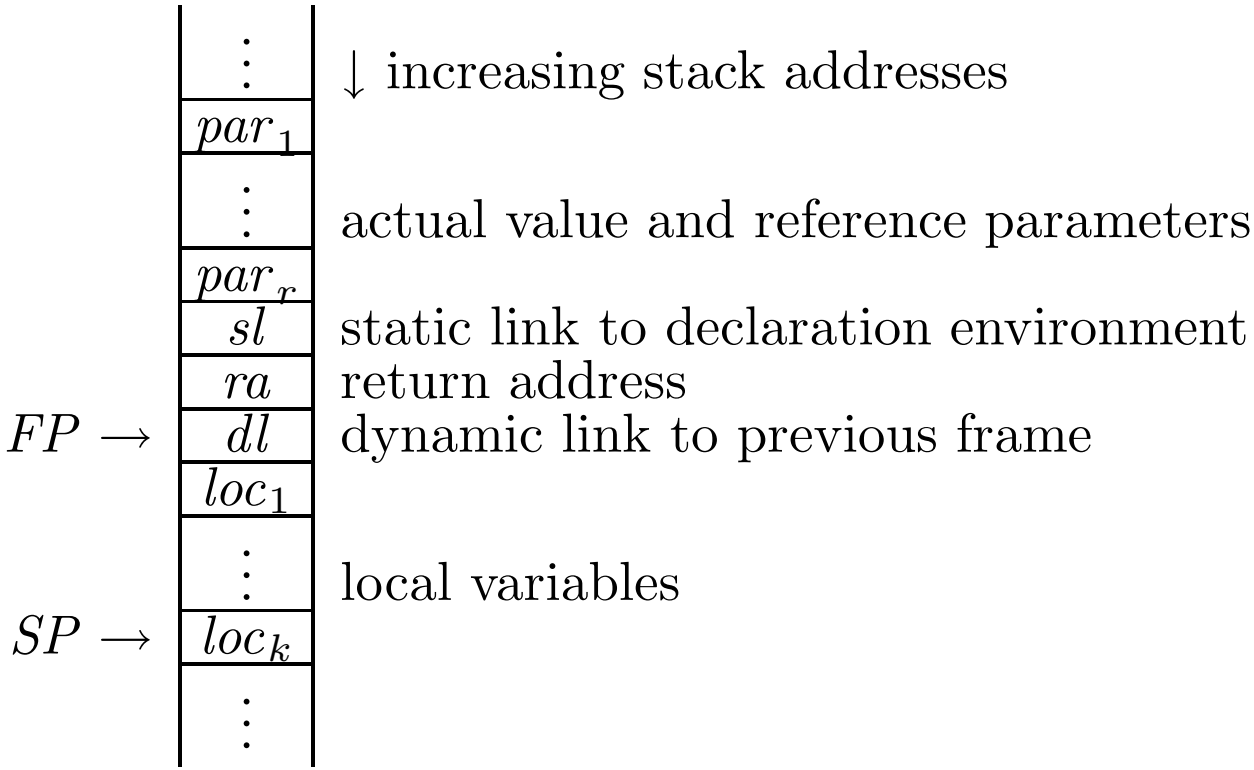
with

- the program counter $PC := \mathbb{N}$,
- the stack pointer $SP := \mathbb{N}$,
- the frame pointer $FP := \mathbb{N}$,
- the index register $IR := \mathbb{N}$, and
- the runtime stack $RS := (\mathbb{N} \to \mathbb{Z})$.

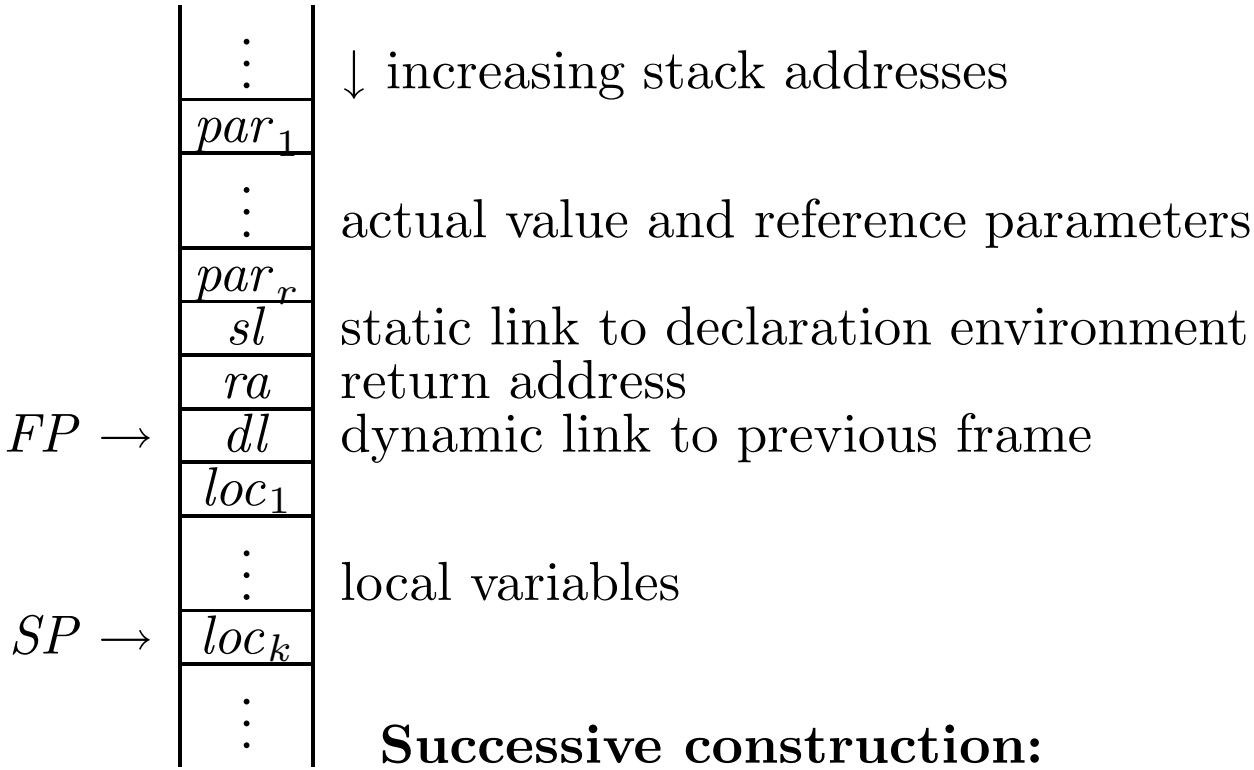
Characteristics:

- Closer to "real" machine (less powerful instructions)
- Runtime stack RS combines data stack DS and procedure stack PS
- Absolute addressing of stack entries
- Stack pointer SP points to top of stack
- Frame pointer FP points to (dynamic link field of) topmost frame
- Index register IR implements dereferencing of static links

Structure of Frames



Structure of Frames



Successive construction:

- by calling procedure:
  1. Computation of actual parameters par_i
  2. Computation of static link sl using index register IR
  3. Jump to called procedure, setting return address ra
- by called procedure ("entry code"):
  4. Store pointer to previous frame as dynamic link dl
  5. Allocate memory for local variables loc_j

New AM Instructions

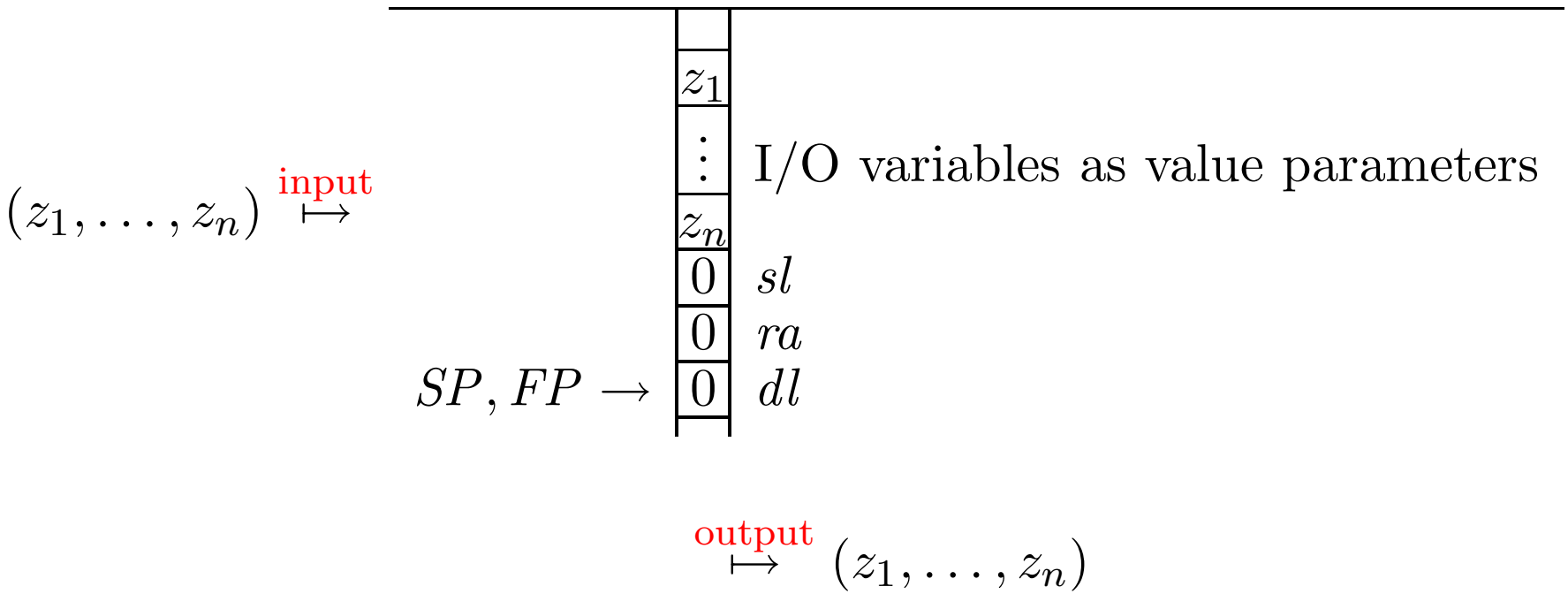
Definition 20.6 (New AM instructions)

The procedure instructions (`CALL(`ca`,`dif`,`loc`)`, `RET`) and transfer instructions (`LOAD(`dif`,`off`)`, `STORE(`dif`,`off`)`) are replaced by the following instructions with the respective semantics $[\![O]\!] : S \dashrightarrow S$:

$$
\begin{aligned}
[\![\texttt{CALL } ca]\!](a, sp, fp, ir, p) &:= (ca, sp+1, fp, ir, p[sp+1 \mapsto a+1]) \\
[\![\texttt{RET } k]\!](a, sp, fp, ir, p) &:= (p(sp), sp-k-1, fp, ir, p) \\
[\![\texttt{PUSH } z]\!](a, sp, fp, ir, p) &:= (a+1, sp+1, fp, ir, p[sp+1 \mapsto z]) \\
[\![\texttt{PUSH FP}]\!](a, sp, fp, ir, p) &:= (a+1, sp+1, fp, ir, p[sp+1 \mapsto fp]) \\
[\![\texttt{PUSH <FP>}]\!](a, sp, fp, ir, p) &:= (a+1, sp+1, fp, ir, p[sp+1 \mapsto p(fp)]) \\
[\![\texttt{PUSH <IR-2>}]\!](a, sp, fp, ir, p) &:= (a+1, sp+1, fp, ir, p[sp+1 \mapsto p(ir-2)]) \\
[\![\texttt{POP FP}]\!](a, sp, fp, ir, p) &:= (a+1, sp-1, p(sp), ir, p) \\
[\![\texttt{POP <}n\texttt{>}]\!](a, sp, fp, ir, p) &:= (a+1, sp-1, fp, ir, p[n \mapsto p(sp)]) \\
[\![\texttt{ADD SP,}n]\!](a, sp, fp, ir, p) &:= (a+1, sp+n, fp, ir, p) \\
[\![\texttt{LOAD FP,SP}]\!](a, sp, fp, ir, p) &:= (a+1, sp, sp, ir, p) \\
[\![\texttt{LOAD SP,FP}]\!](a, sp, fp, ir, p) &:= (a+1, fp, fp, ir, p) \\
[\![\texttt{LOAD IR,<}n\texttt{>}]\!](a, sp, fp, ir, p) &:= (a+1, sp, fp, p(n), p)
\end{aligned}
$$

Processing Input and Output

For $P = \texttt{in/out}\ I_1, \ldots, I_n; K\,. \in Pgm$:



Outline

Translation of Extended EPL into AM Programs

Goal: modification of translation mapping

$$\text{trans} : Pgm \dashrightarrow AM$$

taking into account

- Procedures with parameters
- Modified AM instruction set

Modifying the Symbol Table

$$
\begin{aligned}
Tab := \{\mathrm{st} \mid \mathrm{st} : Ide \dashrightarrow\ & (\{\texttt{const}\} \times \mathbb{Z}) \\
& \cup (\{\texttt{var}\} \times Lev \times Off) \\
& \quad \% \text{ local variables and value parameters} \\
& \cup (\{\texttt{proc}\} \times PC \times Lev \times Size)\} \\
& \cup (\{\texttt{rpar}\} \times Lev \times Off)\} \\
& \quad \% \text{ reference parameters}
\end{aligned}
$$

Position of $FP \implies$ negative offsets possible $\implies Off := \mathbb{Z}$

Modifying the Symbol Table

$$
\begin{aligned}
Tab := \{\mathrm{st} \mid \mathrm{st} : Ide \dashrightarrow\ & (\{\texttt{const}\} \times \mathbb{Z}) \\
& \cup (\{\texttt{var}\} \times Lev \times Off) \\
& \quad \% \text{ local variables and value parameters} \\
& \cup (\{\texttt{proc}\} \times PC \times Lev \times Size)\} \\
& \cup (\{\texttt{rpar}\} \times Lev \times Off)\} \\
& \quad \% \text{ reference parameters}
\end{aligned}
$$

Position of $FP \implies$ negative offsets possible $\implies Off := \mathbb{Z}$

Initial symbol table for $P = \texttt{in/out}\ I_1, \ldots, I_n; K\,. \in Pgm$:

$$\mathrm{st}_{\texttt{I/O}}(I_j) := (\texttt{var}, 0, j - n - 3) \qquad \text{for every } j \in [n]$$

Modifying the Symbol Table

$$\begin{aligned} Tab := \{\mathrm{st} \mid \mathrm{st} : Ide \dashrightarrow\ & (\{\texttt{const}\} \times \mathbb{Z}) \\ & \cup (\{\texttt{var}\} \times Lev \times Off) \\ & \quad \% \text{ local variables and value parameters} \\ & \cup (\{\texttt{proc}\} \times PC \times Lev \times Size)\} \\ & \cup (\{\texttt{rpar}\} \times Lev \times Off)\} \\ & \quad \% \text{ reference parameters} \end{aligned}$$

Position of $FP \implies$ negative offsets possible $\implies Off := \mathbb{Z}$

Initial symbol table for $P = \texttt{in/out}\ I_1, \ldots, I_n; K\,. \in Pgm$:

$$\mathrm{st}_{\texttt{I/O}}(I_j) := (\texttt{var}, 0, j - n - 3) \qquad \text{for every } j \in [n]$$

update function: as before
(processing of procedure parameters by dt)

Translation of Programs

Translation of `in/out` $I_1, \ldots, I_n; D\ C$.:

1. Create (initial part of) `MAIN` frame for executing C
2. Stop program execution after return

Translation of Programs

Translation of `in/out` $I_1, \ldots, I_n; D\ C$.:

1. Create (initial part of) `MAIN` frame for executing C
2. Stop program execution after return

Definition 20.7 (Translation of programs)

The mapping

$$\text{trans} : Pgm \dashrightarrow AM$$

is defined by

$$\begin{aligned}\text{trans}(\texttt{in/out}\ I_1, \ldots, I_n; K.) := \ & 1 : \texttt{PUSH FP;} && \% \text{ create static link} \\ & 2 : \texttt{CALL } a\texttt{;} && \% \text{ create return address} \\ & 3 : \texttt{JMP 0;} && \% \text{ STOP} \\ & \text{kt}(K, \text{st}_{\texttt{I/O}}, a, 1, \underbrace{0}_{\text{no. of parameters}}) &&\end{aligned}$$

Translation of Blocks

Definition 20.8 (Translation of blocks)

The mapping

$$\mathrm{kt} : \mathit{Block} \times \mathit{Tab} \times \mathit{PC} \times \mathit{Lev} \times \mathbb{N} \dashrightarrow \mathit{AM}$$

is defined by

$$\begin{array}{l}
\mathrm{kt}(D\ C, \mathrm{st}, a, l, r) \\
\quad := \quad \mathrm{dt}(D, \mathrm{update}(D, \mathrm{st}, l), l) \\
\qquad\qquad a : \texttt{PUSH FP;} \qquad\qquad \% \text{ entry code:} \\
\qquad\qquad a+1 : \texttt{LOAD FP,SP;} \qquad\qquad \% \text{ create dynamic link and} \\
\qquad\qquad a+2 : \texttt{ADD SP,}\mathrm{size}(D\ C)\texttt{;} \qquad\qquad \% \text{ allocate local variables} \\
\qquad\qquad \mathrm{ct}(C, \mathrm{update}(D, \mathrm{st}, l), a+3, l) \\
\qquad\qquad a' : \texttt{LOAD SP,FP;} \qquad\qquad \% \text{ exit code:} \\
\qquad\qquad a'+1 : \texttt{POP FP;} \qquad\qquad \% \text{ reset frame pointer and} \\
\qquad\qquad a'+2 : \texttt{RET } r+1\texttt{;} \qquad\qquad \% \text{ jump back}
\end{array}$$

Translation of Declarations

Definition 20.9 (Translation of declarations)

The mapping

$$\mathrm{dt} : \mathit{Dcl} \times \mathit{Tab} \times \mathit{Lev} \dashrightarrow \mathit{AM}$$

is defined by

$$\begin{aligned}
\mathrm{dt}(D_C\ D_V\ D_P, \mathrm{st}, l) &:= \mathrm{dt}(D_P, \mathrm{st}, l) \\
\mathrm{dt}(\varepsilon, \mathrm{st}, l) &:= \varepsilon \\
\mathrm{dt}(\texttt{proc}\ I\texttt{(}I_1\texttt{,}\ldots\texttt{,}I_p\texttt{;var}\ J_1\texttt{,}\ldots\texttt{,}J_q\texttt{);}K\texttt{;}D'_P, \mathrm{st}, l) & \\
:= \mathrm{kt}(K, \mathrm{st}', a_I, l+1, p+q) & \\
\mathrm{dt}(D'_P, \mathrm{st}, l) & \\
\text{where } \mathrm{st}(I) = (\texttt{proc}, a_I, \ldots, \ldots) & \\
\text{and } \mathrm{st}' := \mathrm{st}[I_1 \mapsto (\texttt{var}, l+1, -p-q-2), & \\
\vdots & \\
I_p \mapsto (\texttt{var}, l+1, -q-3), & \\
J_1 \mapsto (\texttt{rpar}, l+1, -q-2), & \\
\vdots & \\
J_q \mapsto (\texttt{rpar}, l+1, -3)] &
\end{aligned}$$

Translation of Commands

The mapping

$$\text{ct} : Cmd \times Tab \times PC \times Lev \dashrightarrow AM$$

needs to be adapted only for

- assignments and
- procedure calls;

ct is not modified otherwise.

Translation of Assignments

Definition 20.10 (Translation of assignments)

$$
\mathrm{ct}(I := A, \mathrm{st}, a, l) := \begin{cases}
\mathrm{at}(A, \mathrm{st}, a, l) & \text{if } \mathrm{st}(I) = (\texttt{var}, lev, off) \\
\texttt{POP <FP+}off\texttt{>;} & \quad \text{and } l = lev \\
\\
\mathrm{at}(A, \mathrm{st}, a, l) & \text{if } \mathrm{st}(I) = (\texttt{var}, lev, off) \\
\texttt{LOAD IR,<FP-2>;} & \quad \text{and } l - lev = k + 1 > 0 \\
\left.\begin{array}{l}\texttt{LOAD IR,<IR-2>;} \\ \ldots \\ \texttt{LOAD IR,<IR-2>;}\end{array}\right\} k \text{ times} & \\
\texttt{POP <IR+}off\texttt{>;} & \\
\\
\mathrm{at}(A, \mathrm{st}, a, l) & \text{if } \mathrm{st}(I) = (\texttt{rpar}, lev, off) \\
\texttt{LOAD IR,<FP+}off\texttt{>;} & \quad \text{and } l = lev \\
\texttt{POP <IR>;} & \\
\\
\mathrm{at}(A, \mathrm{st}, a, l) & \text{if } \mathrm{st}(I) = (\texttt{rpar}, lev, off) \\
\texttt{LOAD IR,<FP-2>;} & \quad \text{and } l - lev = k + 1 > 0 \\
\left.\begin{array}{l}\texttt{LOAD IR,<IR-2>;} \\ \ldots \\ \texttt{LOAD IR,<IR-2>;}\end{array}\right\} k \text{ times} & \\
\texttt{LOAD IR,<IR+}off\texttt{>;} & \\
\texttt{POP <IR>;} &
\end{cases}
$$

Translation of Procedure Calls

Definition 20.11 (Translation of procedure calls)

$$\mathrm{ct}(I(A_1,\ldots,A_p;V_1,\ldots,V_q),\mathrm{st},a,l) :=$$
$$\quad \mathrm{at}(A_1,\mathrm{st},a,l)\ldots\mathrm{at}(A_p,\mathrm{st},a,l)\ \mathrm{ref}(V_1,l)\ldots\mathrm{ref}(V_q,l)\ \mathrm{slink}(l)\ \texttt{CALL}\ ca\texttt{;}$$

where

$$\mathrm{st}(I) = (\texttt{proc}, ca, lev, loc)$$

$$\mathrm{ref}(V,l) := \begin{cases} \texttt{PUSH FP+}off\texttt{;} & \text{if } \mathrm{st}(V) = (\texttt{var}, lev, off) \text{ and } l = lev \\ \texttt{LOAD IR,<FP-2>;} & \\ \left.\begin{array}{l}\texttt{LOAD IR,<IR-2>;}\\ \quad\ldots\\ \texttt{LOAD IR,<IR-2>;}\end{array}\right\} k & \text{if } \mathrm{st}(V) = (\texttt{var}, lev, off) \\ & \quad \text{and } l - lev = k+1 > 0 \\ \texttt{PUSH IR+}off\texttt{;} & \\ \texttt{PUSH <FP+}off\texttt{>;} & \text{if } \mathrm{st}(V) = (\texttt{rpar}, lev, off) \text{ and } l = lev \\ \texttt{LOAD IR,<FP-2>;} & \\ \left.\begin{array}{l}\texttt{LOAD IR,<IR-2>;}\\ \quad\ldots\\ \texttt{LOAD IR,<IR-2>;}\end{array}\right\} k & \text{if } \mathrm{st}(V) = (\texttt{rpar}, lev, off) \\ & \quad \text{and } l - lev = k+1 > 0 \\ \texttt{PUSH <IR+}off\texttt{>;} & \end{cases}$$

$$\mathrm{slink}(l) := \begin{cases} \texttt{PUSH FP;} & \text{if } \mathrm{st}(I) = (\texttt{proc}, ca, lev, loc) \text{ and } l = lev \\ \texttt{LOAD IR,<FP-2>;} & \\ \left.\begin{array}{l}\texttt{LOAD IR,<IR-2>;}\\ \quad\ldots\\ \texttt{LOAD IR,<IR-2>;}\end{array}\right\} k & \text{if } \mathrm{st}(I) = (\texttt{proc}, ca, lev, loc) \\ & \quad \text{and } l - lev = k+1 > 0 \\ \texttt{PUSH IR;} & \end{cases}$$

Translation of Arithmetic Expressions

Definition 20.12 (Translation of arithmetic expressions)

In $\mathrm{at} : \mathit{AExp} \times \mathit{Tab} \times \mathit{PC} \times \mathit{Lev} \dashrightarrow \mathit{AM}$, only the handling of identifiers need to be adapted:

$$\mathrm{at}(I, \mathrm{st}, a, l) := \begin{cases} \texttt{PUSH } z\texttt{;} & \text{if } \mathrm{st}(I) = (\texttt{const}, z) \\ \\ \texttt{PUSH <FP+}\mathit{off}\texttt{>;} & \text{if } \mathrm{st}(I) = (\texttt{var}, \mathit{lev}, \mathit{off}) \text{ and } l = \mathit{lev} \\ \\ \texttt{LOAD IR,<FP-2>;} & \text{if } \mathrm{st}(I) = (\texttt{var}, \mathit{lev}, \mathit{off}) \\ \left.\begin{array}{l}\texttt{LOAD IR,<IR-2>;} \\ \quad\ldots \\ \texttt{LOAD IR,<IR-2>;}\end{array}\right\} k & \quad \text{and } l - \mathit{lev} = k + 1 > 0 \\ \texttt{PUSH <IR+}\mathit{off}\texttt{>;} & \\ \\ \texttt{LOAD IR,<FP+}\mathit{off}\texttt{>;} & \text{if } \mathrm{st}(I) = (\texttt{rpar}, \mathit{lev}, \mathit{off}) \text{ and } l = \mathit{lev} \\ \texttt{PUSH <IR>;} & \\ \\ \texttt{LOAD IR,<FP-2>;} & \text{if } \mathrm{st}(I) = (\texttt{rpar}, \mathit{lev}, \mathit{off}) \\ \left.\begin{array}{l}\texttt{LOAD IR,<IR-2>;} \\ \quad\ldots \\ \texttt{LOAD IR,<IR-2>;}\end{array}\right\} k & \quad \text{and } l - \mathit{lev} = k + 1 > 0 \\ \texttt{LOAD IR,<IR+}\mathit{off}\texttt{>;} & \\ \texttt{PUSH <IR>;} & \end{cases}$$