

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

- 1 Repetition: Translation of EPL into AM Programs
- 2 Boolean Expressions with Sequential Semantics
- 3 Adding Procedure Parameters
- 4 Translation of Procedure Parameters

Translation of EPL into AM Programs

Translation mapping $\text{trans} : Pgm \dashrightarrow AM$ defined using

- symbol table:

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

- $\text{update} : Dcl \times Tab \times Lev \dashrightarrow Tab$
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 $(z_1, \dots, z_n), (z'_1, \dots, z'_n) \in \mathbb{Z}^n$:

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

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 □

- 1 Repetition: Translation of EPL into AM Programs
- 2 Boolean Expressions with Sequential Semantics
- 3 Adding Procedure Parameters
- 4 Translation of Procedure Parameters

Boolean Expressions with Sequential Semantics

So far: Boolean expressions with **strict** semantics

$$b_1 \hat{\wedge} \perp = \perp$$

$$\perp \hat{\wedge} b_2 = \perp$$

Now: Boolean expressions with **sequential** semantics

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

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

$$\perp \hat{\wedge} b = \perp$$

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

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

is defined by

$$\begin{aligned} \text{sbt}(A_1 < A_2, \text{st}, a, a_t, a_f, l) &:= \text{at}(A_1, \text{st}, a, l) \\ &\quad \text{at}(A_2, \text{st}, a', l) \\ &\quad a'' : \text{LT}; \\ &\quad a'' + 1 : \text{JFALSE}(a_f); \\ &\quad a'' + 2 : \text{JMP}(a_t); \end{aligned}$$

$$\text{sbt}(\text{not } B, \text{st}, a, a_t, a_f, l) := \text{sbt}(B, \text{st}, a, a_f, a_t, l)$$

$$\begin{aligned} \text{sbt}(B_1 \text{ and } B_2, \text{st}, a, a_t, a_f, l) &:= \text{sbt}(B_1, \text{st}, a, a', a_f, l) \\ &\quad \text{sbt}(B_2, \text{st}, a', a_t, a_f, l) \end{aligned}$$

$$\begin{aligned} \text{sbt}(B_1 \text{ or } B_2, \text{st}, a, a_t, a_f, l) &:= \text{sbt}(B_1, \text{st}, a, a_t, a', l) \\ &\quad \text{sbt}(B_2, \text{st}, a', a_t, a_f, l) \end{aligned}$$

Definition 20.2 (Jumping code for commands)

The mapping

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

is defined by

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

(remaining cases analogously)

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

⇒ generally: longer code, but shorter executions

- 1 Repetition: Translation of EPL into AM Programs
- 2 Boolean Expressions with Sequential Semantics
- 3 Adding Procedure Parameters
- 4 Translation of Procedure Parameters

Extended Syntax of EPL

Definition 20.4 (Extended syntax of EPL)

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

$$\begin{array}{l} \vdots \\ \text{Cmd : } \quad C ::= I := A \mid C_1; C_2 \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid \\ \quad \quad \quad \text{while } B \text{ do } C \mid \underbrace{I(A_1, \dots, A_p; V_1, \dots, V_q)}_{\text{actual value and ref. parameters}} \\ \\ \text{Dcl : } \quad D ::= D_C D_V D_P \\ \quad \quad D_C ::= \varepsilon \mid \text{const } I_1 := z_1, \dots, I_n := z_n; \\ \quad \quad D_V ::= \varepsilon \mid \text{var } I_1, \dots, I_n; \\ \quad \quad D_P ::= \varepsilon \mid \underbrace{\text{proc } I(I_1, \dots, I_p; \text{var } J_1, \dots, J_q); K; \text{proc } \dots }_{\text{formal value and ref. parameters}} \\ \quad \quad \quad \vdots \end{array}$$

where $I_k, J_k, V_k \in Ide$

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)
- ⇒ assignments have **no effect on calling site**
- only **variables as formal reference parameters** (all different)
- implemented by **pointer** to corresponding actual address
- ⇒ assignments have effect on calling site (**return values**)

Definition 20.5 (Extended abstract machine for EPL)

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

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

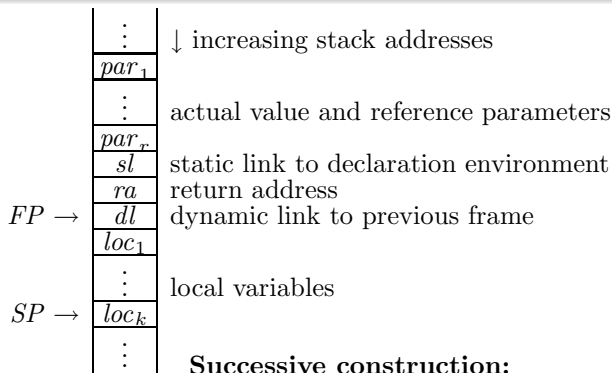
with

- 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** $RS := (\mathbb{N} \rightarrow \mathbb{Z})$.

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



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
 - 6 Allocate memory for local variables loc_j

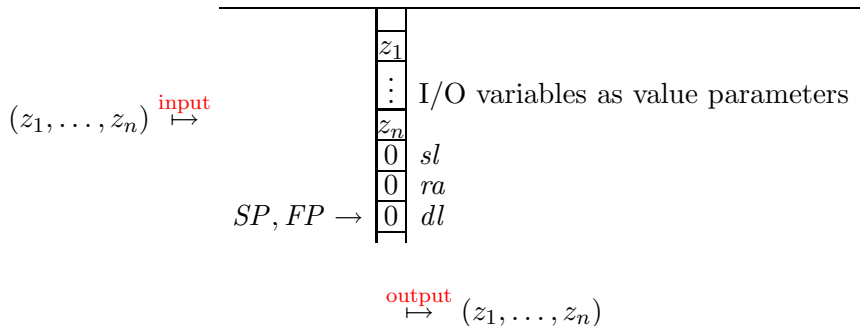
Definition 20.6 (New AM instructions)

The **procedure instructions** ($\text{CALL}(ca, dif, loc)$, RET) and **transfer instructions** ($\text{LOAD}(dif, off)$, $\text{STORE}(dif, off)$) are replaced by the following instructions with the respective semantics $\llbracket O \rrbracket : S \dashrightarrow S$:

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

Processing Input and Output

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



- 1 Repetition: Translation of EPL into AM Programs
- 2 Boolean Expressions with Sequential Semantics
- 3 Adding Procedure Parameters
- 4 Translation of Procedure Parameters

Goal: modification of **translation mapping**

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

taking into account

- Procedures with parameters
- Modified AM instruction set

Modifying the Symbol Table

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

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

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

$$st_{I/O}(I_j) := (\text{var}, 0, j - n - 3) \quad \text{for every } j \in [n]$$

update function: as before

(processing of procedure parameters by dt)

Translation of Programs

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

- 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

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

Definition 20.8 (Translation of blocks)

The mapping

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

is defined by

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

Definition 20.9 (Translation of declarations)

The mapping

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

is defined by

$$\text{dt}(D_C \ D_V \ D_P, \text{st}, l) := \text{dt}(D_P, \text{st}, l)$$

$$\text{dt}(\varepsilon, \text{st}, l) := \varepsilon$$

$$\begin{aligned} \text{dt}(\text{proc } I(I_1, \dots, I_p; \text{var } J_1, \dots, J_q); K; D'_P, \text{st}, l) \\ := \text{kt}(K, \text{st}', a_I, l + 1, p + q) \\ \text{dt}(D'_P, \text{st}, l) \end{aligned}$$

where $\text{st}(I) = (\text{proc}, a_I, \dots, \dots)$

and $\text{st}' := \text{st}[I_1 \mapsto (\text{var}, l + 1, -p - q - 2),$

$$\begin{aligned} & \vdots \\ I_p & \mapsto (\text{var}, l + 1, -q - 3), \\ J_1 & \mapsto (\text{rpar}, l + 1, -q - 2), \\ & \vdots \\ J_q & \mapsto (\text{rpar}, l + 1, -3)] \end{aligned}$$

The mapping

$$ct : Cmd \times Tab \times PC \times Lev \dashrightarrow AM$$

needs to be adapted only for

- assignments and
- procedure calls;

ct is not modified otherwise.

Translation of Assignments

Definition 20.10 (Translation of assignments)

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

Definition 20.11 (Translation of procedure calls)

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

where

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

$$\text{ref}(V, l) := \begin{cases} \begin{cases} \text{PUSH FP} + \text{off}; & \text{if } \text{st}(V) = (\text{var}, lev, \text{off}) \text{ and } l = lev \\ \text{LOAD IR}, \langle \text{FP} - 2 \rangle; & \text{if } \text{st}(V) = (\text{var}, lev, \text{off}) \\ \text{LOAD IR}, \langle \text{IR} - 2 \rangle; & \text{and } l - lev = k + 1 > 0 \\ \dots \\ \text{LOAD IR}, \langle \text{IR} - 2 \rangle; \end{cases} \Bigg\}^k \\ \begin{cases} \text{PUSH IR} + \text{off}; \\ \text{PUSH } \langle \text{FP} + \text{off} \rangle; & \text{if } \text{st}(V) = (\text{rpar}, lev, \text{off}) \text{ and } l = lev \\ \text{LOAD IR}, \langle \text{FP} - 2 \rangle; & \text{if } \text{st}(V) = (\text{rpar}, lev, \text{off}) \\ \text{LOAD IR}, \langle \text{IR} - 2 \rangle; & \text{and } l - lev = k + 1 > 0 \\ \dots \\ \text{LOAD IR}, \langle \text{IR} - 2 \rangle; \end{cases} \Bigg\}^k \\ \text{PUSH } \langle \text{IR} + \text{off} \rangle; \end{cases}$$

$$\text{slink}(l) := \begin{cases} \begin{cases} \text{PUSH FP}; & \text{if } \text{st}(I) = (\text{proc}, ca, lev, loc) \text{ and } l = lev \\ \text{LOAD IR}, \langle \text{FP} - 2 \rangle; & \text{if } \text{st}(I) = (\text{proc}, ca, lev, loc) \\ \text{LOAD IR}, \langle \text{IR} - 2 \rangle; & \text{and } l - lev = k + 1 > 0 \\ \dots \\ \text{LOAD IR}, \langle \text{IR} - 2 \rangle; \end{cases} \Bigg\}^k \\ \text{PUSH IR}; \end{cases}$$

Translation of Arithmetic Expressions

Definition 20.12 (Translation of arithmetic expressions)

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

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