

Compiler Construction

Lecture 21: Code Generation VI

(Translation Example & Static Data Structures)

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 Correction: update Function
- 2 Repetition: Procedures with Parameters
- 3 A Translation Example
- 4 Alternative Implementation of Static Links
- 5 Intermediate Code for Data Structures
- 6 Static Data Structures
- 7 Modifying the Abstract Machine
- 8 Translation of EPL with Static Data Structures into AM Programs

Correction: update Function

The symbol table is maintained by the function $\text{update}(D, \text{st}, a, l)$ which specifies the update of symbol table st according to declaration D (with respect to **next free code address a and** current level l):

Definition (update function)

$$\text{update} : Dcl \times Tab \times PC \times Lev \dashrightarrow Tab$$

is defined by

```
update( $D_C \ D_V \ D_P, \text{st}, a, l$ )
  := update( $D_P, \text{update}(D_V, \text{update}(D_C, \text{st}, a, l), a, l), a, l$ )
     if all identifiers in  $D_C \ D_V \ D_P$  different
update( $\varepsilon, \text{st}, a, l$ )
  := st
update(const  $I_1 := z_1, \dots, I_n := z_n; \text{st}, a, l$ )
  := st[ $I_1 \mapsto (\text{const}, z_1), \dots, I_n \mapsto (\text{const}, z_n)$ ]
update(var  $I_1, \dots, I_n; \text{st}, a, l$ )
  := st[ $I_1 \mapsto (\text{var}, l, 1), \dots, I_n \mapsto (\text{var}, l, n)$ ]
update(proc  $I_1; K_1; \dots; I_n; K_n; \text{st}, a, l$ )
  := st[ $I_1 \mapsto (\text{proc}, a_1, l, \text{size}(K_1)), \dots, I_n \mapsto (\text{proc}, a_n, l, \text{size}(K_n))$ ]
     with “fresh” addresses  $a_1, \dots, a_n$ 
     where  $\text{size}(D_C \text{ var } I_1, \dots, I_n; D_P C) := n$ 
```

Correction: update Function

The symbol table is maintained by the function $\text{update}(D, \text{st}, l)$ which specifies the update of symbol table st according to declaration D (with respect to current level l):

Definition (update function)

$$\text{update} : Dcl \times Tab \times Lev \dashrightarrow Tab$$

is defined by

$$\begin{aligned} & \text{update}(D_C \ D_V \ D_P, \text{st}, l) \\ & \quad := \text{update}(D_P, \text{update}(D_V, \text{update}(D_C, \text{st}, l), l), l) \\ & \quad \quad \text{if all identifiers in } D_C \ D_V \ D_P \text{ different} \\ & \text{update}(\varepsilon, \text{st}, l) \\ & \quad := \text{st} \\ & \text{update}(\text{const } I_1 := z_1, \dots, I_n := z_n; \text{st}, l) \\ & \quad := \text{st}[I_1 \mapsto (\text{const}, z_1), \dots, I_n \mapsto (\text{const}, z_n)] \\ & \text{update}(\text{var } I_1, \dots, I_n; \text{st}, l) \\ & \quad := \text{st}[I_1 \mapsto (\text{var}, l, 1), \dots, I_n \mapsto (\text{var}, l, n)] \\ & \text{update}(\text{proc } I_1; K_1; \dots; I_n; K_n; \text{st}, l) \\ & \quad := \text{st}[I_1 \mapsto (\text{proc}, a_1, l, \text{size}(K_1)), \dots, I_n \mapsto (\text{proc}, a_n, l, \text{size}(K_n))] \\ & \quad \quad \text{with “fresh” addresses } a_1, \dots, a_n \\ & \quad \quad \text{where } \text{size}(D_C \ \text{var } I_1, \dots, I_n; D_P C) := n \end{aligned}$$

- 1 Correction: update Function
- 2 Repetition: Procedures with Parameters
- 3 A Translation Example
- 4 Alternative Implementation of Static Links
- 5 Intermediate Code for Data Structures
- 6 Static Data Structures
- 7 Modifying the Abstract Machine
- 8 Translation of EPL with Static Data Structures into AM Programs

Extended Syntax of EPL

Definition (Extended syntax of EPL)

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

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

where $I_k, J_k, V_k \in Ide$

Definition (Extended abstract machine for EPL)

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

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

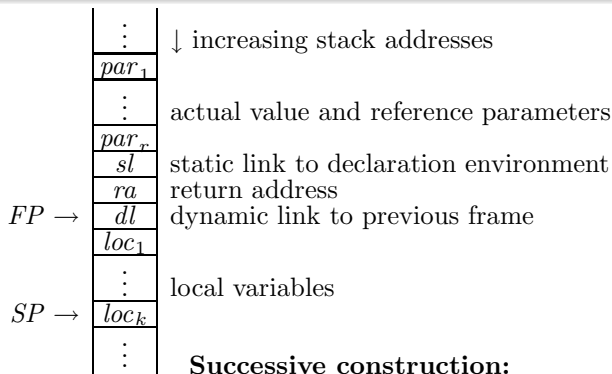
with

- 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

Goal: modification of translation mapping

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

taking into account

- Procedures with parameters
- Modified AM instruction set

Modifying the Symbol Table

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

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

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

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

update function: as before

(processing of procedure parameters by dt)

- 1 Correction: update Function
- 2 Repetition: Procedures with Parameters
- 3 A Translation Example
- 4 Alternative Implementation of Static Links
- 5 Intermediate Code for Data Structures
- 6 Static Data Structures
- 7 Modifying the Abstract Machine
- 8 Translation of EPL with Static Data Structures into AM Programs

Translation Example I

Example 21.1 (Factorial function)

$P = \text{in/out } x, y;$
 $\left. \begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array} \right\} C_F \left. \vphantom{\begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array}} \right\}^D \left. \vphantom{\begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array}} \right\}^K$
 $\left. \begin{array}{l} y := 1; \\ F(x, y); \end{array} \right\} C$

Translation Example I

Example 21.1 (Factorial function)

$P = \text{in/out } x, y;$
 $\left. \begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array} \right\} C_F \left. \right\} D \left. \right\} K$
 $\left. \begin{array}{l} y := 1; \\ F(x, y); \end{array} \right\} C$

$\text{trans}(P) = 1 : \text{PUSH FP}; \% \text{ static link}$
 $2 : \text{CALL } a_0;$
 $3 : \text{JMP } 0; \% \text{ STOP}$
 $\text{kt}(K, \text{st}_{I/0}, a_0, 1, 0)$

where

$\text{st}_{I/0} = [x \mapsto (\text{var}, 0, -4), y \mapsto (\text{var}, 0, -3)]$

Translation Example I

Example 21.1 (Factorial function)

$P = \text{in/out } x, y;$
 $\left. \begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array} \right\} C_F \left. \right\} D \left. \right\} K$
 $\left. \begin{array}{l} y := 1; \\ F(x, y); \end{array} \right\} C$

where st'
 $= \text{update}(D, st_{I/0}, 1)$
 $= st_{I/0}[F \mapsto (\text{proc}, a_1, 1, 0)]$

$\text{trans}(P) = 1 : \text{PUSH FP}; \% \text{ static link}$
 $2 : \text{CALL } a_0;$
 $3 : \text{JMP } 0; \% \text{ STOP}$
 $\text{kt}(K, st_{I/0}, a_0, 1, 0)$

where
 $st_{I/0} = [x \mapsto (\text{var}, 0, -4), y \mapsto (\text{var}, 0, -3)]$

$\text{kt}(K, st_{I/0}, a_0, 1, 0)$
 $= \text{dt}(D, st', 1)$
 $a_0 : \text{PUSH FP}; \% \text{ dynamic link}$
 $\text{LOAD FP, SP}; \% \text{ set FP}$
 $\text{ADD SP, } \underbrace{\text{size}(K)}_0; \% \text{ local vars}$
 $\text{ct}(C, st', a_0 + 3, 1)$
 $\text{LOAD SP, FP};$
 $\text{POP FP}; \% \text{ reset FP}$
 $\text{RET } 1; \% \text{ jump back}$

Translation Example I

Example 21.1 (Factorial function)

$$P = \text{in/out } x, y; \left. \begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array} \right\} C_F \left. \right\} D \left. \right\} K$$

$$\left. \begin{array}{l} y := 1; \\ F(x, y); \end{array} \right\} C$$

trans(P) = 1 : PUSH FP; % static link
 2 : CALL a_0 ;
 3 : JMP 0; % STOP
 kt($K, st_{I/0}, a_0, 1, 0$)

where
 $st_{I/0} = [x \mapsto (\text{var}, 0, -4), y \mapsto (\text{var}, 0, -3)]$

kt($K, st_{I/0}, a_0, 1, 0$)
 = dt($D, st', 1$)
 a_0 : PUSH FP; % dynamic link
 LOAD FP, SP; % set FP
 ADD SP, $\underbrace{\text{size}(K)}_0$; % local vars
 ct($C, st', a_0 + 3, 1$)
 LOAD SP, FP;
 POP FP; % reset FP
 RET 1; % jump back

where st'
 = update($D, st_{I/0}, 1$)
 = $st_{I/0}[F \mapsto (\text{proc}, a_1, 1, 0)]$

dt($D, st', 1$)
 = kt($C_F, st'', a_1, 2, 2$)
 = a_1 : PUSH FP; % dynamic link
 LOAD FP, SP; % set FP
 ADD SP, 0; % local vars
 ct($C_F, st'', a_1 + 3, 2$)
 LOAD SP, FP;
 POP FP; % reset FP
 RET 3; % jump back

where $st'' =$
 $st'[x \mapsto (\text{var}, 2, -4), y \mapsto (\text{rpar}, 2, -3)]$

Translation Example I

Example 21.1 (Factorial function)

$$P = \text{in/out } x, y; \left. \begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array} \right\} C_F \left. \right\} D \left. \right\} K$$

$$\left. \begin{array}{l} y := 1; \\ F(x, y); \end{array} \right\} C$$

trans(P) = 1 : PUSH FP; % static link
 2 : CALL a_0 ;
 3 : JMP 0; % STOP
 kt($K, st_{I/0}, a_0, 1, 0$)

where
 $st_{I/0} = [x \mapsto (\text{var}, 0, -4), y \mapsto (\text{var}, 0, -3)]$

kt($K, st_{I/0}, a_0, 1, 0$)
 = dt($D, st', 1$)
 a_0 : PUSH FP; % dynamic link
 LOAD FP, SP; % set FP
 ADD SP, $\underbrace{\text{size}(K)}_0$; % local vars
 ct($C, st', a_0 + 3, 1$)
 LOAD SP, FP;
 POP FP; % reset FP
 RET 1; % jump back

where st'
 = update($D, st_{I/0}, 1$)
 = $st_{I/0}[F \mapsto (\text{proc}, a_1, 1, 0)]$
 dt($D, st', 1$)
 = kt($C_F, st'', a_1, 2, 2$)
 = a_1 : PUSH FP; % dynamic link
 LOAD FP, SP; % set FP
 ADD SP, 0; % local vars
 ct($C_F, st'', a_1 + 3, 2$)
 LOAD SP, FP;
 POP FP; % reset FP
 RET 3; % jump back

where $st'' =$
 $st''[x \mapsto (\text{var}, 2, -4), y \mapsto (\text{rpar}, 2, -3)]$
 ct($C_F, st'', a_1 + 3, 2$)
 = bt($x > 1, st'', a_1 + 3, 2$)
 JFALSE a_2 ;
 ct($y := y * x; F(x-1; y), st'', a', 2$)
 a_2 :

Translation Example I

Example 21.1 (Factorial function)

$$P = \text{in/out } x, y; \left. \begin{array}{l} \text{proc } F(x; \text{var } y); \\ \quad \text{if } x > 1 \text{ then} \\ \quad \quad y := y * x; \\ \quad \quad F(x-1; y); \end{array} \right\} C_F \left. \vphantom{\begin{array}{l} \text{proc } F(x; \text{var } y); \\ \text{if } x > 1 \text{ then} \\ \text{if } x > 1 \text{ then} \end{array}} \right\} D \left. \vphantom{\begin{array}{l} \text{proc } F(x; \text{var } y); \\ \text{if } x > 1 \text{ then} \\ \text{if } x > 1 \text{ then} \end{array}} \right\} K$$

$$\left. \begin{array}{l} y := 1; \\ F(x, y); \end{array} \right\} C$$

trans(P) = 1 : PUSH FP; % static link
 2 : CALL a_0 ;
 3 : JMP 0; % STOP
 kt($K, st_{I/0}, a_0, 1, 0$)

where
 $st_{I/0} = [x \mapsto (\text{var}, 0, -4), y \mapsto (\text{var}, 0, -3)]$

kt($K, st_{I/0}, a_0, 1, 0$)
 = dt($D, st', 1$)
 a_0 : PUSH FP; % dynamic link
 LOAD FP, SP; % set FP
 ADD SP, $\underbrace{\text{size}(K)}_0$; % local vars
 ct($C, st', a_0 + 3, 1$)
 LOAD SP, FP;
 POP FP; % reset FP
 RET 1; % jump back

where st'
 = update($D, st_{I/0}, 1$)
 = $st_{I/0}[F \mapsto (\text{proc}, a_1, 1, 0)]$
 dt($D, st', 1$)
 = kt($C_F, st'', a_1, 2, 2$)
 = a_1 : PUSH FP; % dynamic link
 LOAD FP, SP; % set FP
 ADD SP, 0; % local vars
 ct($C_F, st'', a_1 + 3, 2$)
 LOAD SP, FP;
 POP FP; % reset FP
 RET 3; % jump back

where $st'' =$
 $st''[x \mapsto (\text{var}, 2, -4), y \mapsto (\text{rpar}, 2, -3)]$
 ct($C_F, st'', a_1 + 3, 2$)
 = **bt**($x > 1, st'', a_1 + 3, 2$)
 JFALSE a_2 ;
 ct($y := y * x; F(x-1; y), st'', a', 2$)
 a_2 :
bt($x > 1, st'', a_1 + 3, 2$)
 = PUSH <FP-4>; % x
 PUSH 1;
 GT;

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
at(y*x, st'', a', 2)
= LOAD IR, <FP-3>; % adr(y)
  PUSH <IR>; % y
  PUSH <FP-4>; % x
  MULT;
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
at(y*x, st'', a', 2)
= LOAD IR, <FP-3>; % adr(y)
  PUSH <IR>; % y
  PUSH <FP-4>; % x
  MULT;
ct(F(x-1;y), st'', a'', 2)
= at(x-1, st'', a'', 2)
  PUSH <FP-3>; % adr(y)
  LOAD IR, <FP-2>;
  PUSH IR; % static link
  CALL a1;
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
at(y*x, st'', a', 2)
= LOAD IR, <FP-3>; % adr(y)
  PUSH <IR>; % y
  PUSH <FP-4>; % x
  MULT;
ct(F(x-1;y), st'', a'', 2)
= at(x-1, st'', a'', 2)
  PUSH <FP-3>; % adr(y)
  LOAD IR, <FP-2>;
  PUSH IR; % static link
  CALL a1;
```

```
at(x-1, st'', a'', 2)
= PUSH <FP-4>; % x
  PUSH 1;
  SUB;
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
at(y*x, st'', a', 2)
= LOAD IR, <FP-3>; % adr(y)
  PUSH <IR>; % y
  PUSH <FP-4>; % x
  MULT;
ct(F(x-1;y), st'', a'', 2)
= at(x-1, st'', a'', 2)
  PUSH <FP-3>; % adr(y)
  LOAD IR, <FP-2>;
  PUSH IR; % static link
  CALL a1;
```

```
at(x-1, st'', a'', 2)
= PUSH <FP-4>; % x
  PUSH 1;
  SUB;
ct(C, st', a0 + 3, 1)
= ct(y:=1, st', a0 + 3, 1)
  ct(F(x;y), st', a', 1)
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
at(y*x, st'', a', 2)
= LOAD IR, <FP-3>; % adr(y)
  PUSH <IR>; % y
  PUSH <FP-4>; % x
  MULT;
ct(F(x-1;y), st'', a'', 2)
= at(x-1, st'', a'', 2)
  PUSH <FP-3>; % adr(y)
  LOAD IR, <FP-2>;
  PUSH IR; % static link
  CALL a1;
```

```
at(x-1, st'', a'', 2)
= PUSH <FP-4>; % x
  PUSH 1;
  SUB;
ct(C, st', a0 + 3, 1)
= ct(y:=1, st', a0 + 3, 1)
  ct(F(x;y), st', a', 1)
ct(y:=1, st', a0 + 3, 1)
= PUSH 1;
  LOAD IR, <FP-2>; % adr(y)
  POP <IR-3>; % assign
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
at(y*x, st'', a', 2)
= LOAD IR, <FP-3>; % adr(y)
  PUSH <IR>; % y
  PUSH <FP-4>; % x
  MULT;
ct(F(x-1;y), st'', a'', 2)
= at(x-1, st'', a'', 2)
  PUSH <FP-3>; % adr(y)
  LOAD IR, <FP-2>;
  PUSH IR; % static link
  CALL a1;
```

```
at(x-1, st'', a'', 2)
= PUSH <FP-4>; % x
  PUSH 1;
  SUB;
ct(C, st', a0 + 3, 1)
= ct(y:=1, st', a0 + 3, 1)
  ct(F(x;y), st', a', 1)
ct(y:=1, st', a0 + 3, 1)
= PUSH 1;
  LOAD IR, <FP-2>; % adr(y)
  POP <IR-3>; % assign
ct(F(x;y), st', a', 1)
= at(x, st', a', 1)
  LOAD IR, <FP-2>;
  PUSH IR-3; % adr(y)
  PUSH FP; % static link
  CALL a1;
```

Example 21.1 (Factorial function; continued)

```
ct(y:=y*x; F(x-1;y), st'', a', 2)
= ct(y:=y*x, st'', a', 2)
  ct(F(x-1;y), st'', a'', 2)
ct(y:=y*x, st'', a', 2)
= at(y*x, st'', a', 2)
  LOAD IR, <FP-3>; % adr(y)
  POP <IR>; % assign
at(y*x, st'', a', 2)
= LOAD IR, <FP-3>; % adr(y)
  PUSH <IR>; % y
  PUSH <FP-4>; % x
  MULT;
ct(F(x-1;y), st'', a'', 2)
= at(x-1, st'', a'', 2)
  PUSH <FP-3>; % adr(y)
  LOAD IR, <FP-2>;
  PUSH IR; % static link
  CALL a1;
```

```
at(x-1, st'', a'', 2)
= PUSH <FP-4>; % x
  PUSH 1;
  SUB;
ct(C, st', a0 + 3, 1)
= ct(y:=1, st', a0 + 3, 1)
  ct(F(x;y), st', a', 1)
ct(y:=1, st', a0 + 3, 1)
= PUSH 1;
  LOAD IR, <FP-2>; % adr(y)
  POP <IR-3>; % assign
ct(F(x;y), st', a', 1)
= at(x, st', a', 1)
  LOAD IR, <FP-2>;
  PUSH IR-3; % adr(y)
  PUSH FP; % static link
  CALL a1;
at(x, st', a', 1)
= LOAD IR, <FP-2>;
  PUSH <IR-4>
```

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```
1 : PUSH FP;
2 : CALL 27;
3 : JMP 0;
4 : PUSH FP;           % entry F
5 : LOAD FP,SP;
6 : ADD SP,0;
7 : PUSH <FP-4>;       % x>1
8 : PUSH 1;
9 : GT;
10 : JFALSE 24;
11 : LOAD IR,<FP-3>;    % y:=y*x
12 : PUSH <IR>;
13 : PUSH <FP-4>;
14 : MULT;
15 : LOAD IR,<FP-3>;
16 : POP <IR>;
17 : PUSH <FP-4>;       % F(x-1;y)
18 : PUSH 1;
19 : SUB;
20 : PUSH <FP-3>;
21 : LOAD IR,<FP-2>;
22 : PUSH IR;
23 : CALL 4;
24 : LOAD SP,FP;        % exit F
25 : POP FP;
26 : RET 3;
27 : PUSH FP;           % entry MAIN
28 : LOAD FP,SP;
29 : ADD SP,0;
30 : PUSH 1;            % y:=1
31 : LOAD IR,<FP-2>;
32 : POP <IR-3>;
33 : LOAD IR,<FP-2>;     % F(x;y)
34 : PUSH <IR-4>;
35 : LOAD IR,<FP-2>;
36 : PUSH IR-3;
37 : PUSH FP;
38 : CALL 4;
39 : LOAD SP,FP;        % exit MAIN
40 : POP FP;
41 : RET 1;
```

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```
1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;
```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

PC	RS
1	<u>1: 0: 0: 0: 0</u>

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```
1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;
```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

PC	RS
1	1: 0: 0: 0: 0
2	1: 0: 0: 0: 0 : 5

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```
1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;         % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;
```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1: 0: 0: 0: 0</u>
2	1: 0: 0: 0: 0: 5
27	<u>1: 0: 0: 0: 0: 5: 3</u>

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1:0:0:0:0</u>
2	1:0:0:0:0:5
27	1:0:0:0:0:5:3
28	<u>1:0:0:0:0</u> :5:3:5

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1:0:0:0:0</u>
2	1:0:0:0:0:5
27	1:0:0:0:0:5:3
28	1:0:0:0:0:5:3:5
29	<u>1:0:0:0:0:5:3:5</u>

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;           % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;
    
```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , **FP**, **IR**):

<i>PC</i>	<i>RS</i>
1	<u>1: 0: 0: 0: 0</u>
2	1: 0: 0: 0: 0: 5
27	1: 0: 0: 0: 0: 5: 3
28	1: 0: 0: 0: 0: 5: 3: 5
29	1: 0: 0: 0: 0: 5: 3: 5
30	<u>1: 0: 0: 0: 0: 5: 3: 5</u>

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1:0:0:0:0</u>
2	<u>1:0:0:0:0</u> :5
27	<u>1:0:0:0:0</u> :5:3
28	<u>1:0:0:0:0</u> :5:3:5
29	<u>1:0:0:0:0</u> :5:3:5
30	<u>1:0:0:0:0</u> :5:3:5
31	<u>1:0:0:0:0</u> :5:3:5:1

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;
31: LOAD IR,<FP-2>;    % y:=1
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1:0:0:0:0</u>
2	<u>1:0:0:0:0</u> :5
27	<u>1:0:0:0:0</u> :5:3
28	<u>1:0:0:0:0</u> :5:3:5
29	<u>1:0:0:0:0</u> :5:3:5
30	<u>1:0:0:0:0</u> :5:3:5
31	<u>1:0:0:0:0</u> :5:3:5:1
32	<u>1:0:0:0:0</u> :5:3:5:1

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1:0:0:0:0</u>
2	<u>1:0:0:0:0</u> :5
27	<u>1:0:0:0:0</u> :5:3
28	<u>1:0:0:0:0</u> :5:3:5
29	<u>1:0:0:0:0</u> :5:3:5
30	<u>1:0:0:0:0</u> :5:3:5
31	<u>1:0:0:0:0</u> :5:3:5:1
32	<u>1:0:0:0:0</u> :5:3:5:1
33	<u>1:1:0:0:0</u> :5:3:5

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1: 0: 0: 0: 0
2	1: 0: 0: 0: 0 : 5
27	1: 0: 0: 0: 0 : 5: 3
28	1: 0: 0: 0: 0 : 5: 3: 5
29	1: 0: 0: 0: 0 : 5: 3: 5
30	1: 0: 0: 0: 0 : 5: 3: 5
31	1: 0: 0: 0: 0 : 5: 3: 5: 1
32	1: 0: 0: 0: 0 : 5: 3: 5: 1
33	1: 1: 0: 0: 0 : 5: 3: 5
34	1: 1: 0: 0: 0 : 5: 3: 5

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1: 0: 0: 0: 0
2	1: 0: 0: 0: 0 : 5
27	1: 0: 0: 0: 0 : 5: 3
28	1: 0: 0: 0: 0 : 5: 3: 5
29	1: 0: 0: 0: 0 : 5: 3: 5
30	1: 0: 0: 0: 0 : 5: 3: 5
31	1: 0: 0: 0: 0 : 5: 3: 5 : 1
32	1: 0: 0: 0: 0 : 5: 3: 5 : 1
33	1: 1: 0: 0: 0 : 5: 3: 5
34	1: 1: 0: 0: 0 : 5: 3: 5
35	1: 1: 0: 0: 0 : 5: 3: 5 : 1

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;            % y=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1:0:0:0:0</u> :0
2	<u>1:0:0:0:0</u> :0:5
27	<u>1:0:0:0:0</u> :0:5:3
28	<u>1:0:0:0:0</u> :0:5:3:5
29	<u>1:0:0:0:0</u> :0:5:3:5
30	<u>1:0:0:0:0</u> :0:5:3:5
31	<u>1:0:0:0:0</u> :0:5:3:5:1
32	<u>1:0:0:0:0</u> :0:5:3:5:1
33	<u>1:1:0:0:0</u> :0:5:3:5
34	<u>1:1:0:0:0</u> :0:5:3:5
35	<u>1:1:0:0:0</u> :0:5:3:5:1
36	<u>1:1:0:0:0</u> :0:5:3:5:1

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;          % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;      % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;   % y=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;      % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;       % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;          % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;           % y=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;       % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	<u>1:0:0:0:0</u> :0
2	<u>1:0:0:0:0</u> :0:5
27	<u>1:0:0:0:0</u> :0:5:3
28	<u>1:0:0:0:0</u> :0:5:3:5
29	<u>1:0:0:0:0</u> :0:5:3:5
30	<u>1:0:0:0:0</u> :0:5:3:5
31	<u>1:0:0:0:0</u> :0:5:3:5:1
32	<u>1:0:0:0:0</u> :0:5:3:5:1
33	<u>1:1:0:0:0</u> :0:5:3:5
34	<u>1:1:0:0:0</u> :0:5:3:5
35	<u>1:1:0:0:0</u> :0:5:3:5:1
36	<u>1:1:0:0:0</u> :0:5:3:5:1
37	<u>1:1:0:0:0</u> :0:5:3:5:1:2

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;       % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;           % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;       % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0
2	1:0:0:0:0:5
27	1:0:0:0:0:5:3
28	1:0:0:0:0:5:3:5
29	1:0:0:0:0:5:3:5
30	1:0:0:0:0:5:3:5
31	1:0:0:0:0:5:3:5:1
32	1:0:0:0:0:5:3:5:1
33	1:1:0:0:0:5:3:5
34	1:1:0:0:0:5:3:5
35	1:1:0:0:0:5:3:5:1
36	1:1:0:0:0:5:3:5:1
37	1:1:0:0:0:5:3:5:1:2
38	1:1:0:0:0:5:3:5:1:2:8

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , **FP**, **IR**):

<i>PC</i>	<i>RS</i>
1	1: 0: 0: 0: 0
2	1: 0: 0: 0: 0 : 5
27	1: 0: 0: 0: 0 : 5: 3
28	1: 0: 0: 0: 0 : 5: 3: 5
29	1: 0: 0: 0: 0 : 5: 3: 5
30	1: 0: 0: 0: 0 : 5: 3: 5
31	1: 0: 0: 0: 0 : 5: 3: 5 : 1
32	1: 0: 0: 0: 0 : 5: 3: 5 : 1
33	1: 1: 0: 0: 0 : 5: 3: 5
34	1: 1: 0: 0: 0 : 5: 3: 5
35	1: 1: 0: 0: 0 : 5: 3: 5 : 1
36	1: 1: 0: 0: 0 : 5: 3: 5 : 1
37	1: 1: 0: 0: 0 : 5: 3: 5 : 1: 2
38	1: 1: 0: 0: 0 : 5: 3: 5 : 1: 2: 8
4	1: 1: 0: 0: 0 : 5: 3: 5 : 1: 2: 8: 39

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

PC	RS
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;
31: LOAD IR,<FP-2>;    % y:=1
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1: 0: 0: 0: 0: 0
2	1: 0: 0: 0: 0: 0 : 5
27	1: 0: 0: 0: 0: 0 : 5: 3
28	1: 0: 0: 0: 0: 0 : 5: 3: 5
29	1: 0: 0: 0: 0: 0 : 5: 3: 5
30	1: 0: 0: 0: 0: 0 : 5: 3: 5
31	1: 0: 0: 0: 0: 0 : 5: 3: 5: 1
32	1: 0: 0: 0: 0: 0 : 5: 3: 5: 1
33	1: 1: 0: 0: 0: 0 : 5: 3: 5
34	1: 1: 0: 0: 0: 0 : 5: 3: 5
35	1: 1: 0: 0: 0: 0 : 5: 3: 5: 1
36	1: 1: 0: 0: 0: 0 : 5: 3: 5: 1
37	1: 1: 0: 0: 0: 0 : 5: 3: 5: 1: 2
38	1: 1: 0: 0: 0: 0 : 5: 3: 5: 1: 2: 8
4	1: 1: 0: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39
5	1: 1: 0: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
6	1: 1: 0: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;      % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;
31: LOAD IR,<FP-2>;    % y:=1
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>	
1	1:0:0:0:0	0
2	1:0:0:0:0	0:5
27	1:0:0:0:0	0:5:3
28	1:0:0:0:0	0:5:3:5
29	1:0:0:0:0	0:5:3:5
30	1:0:0:0:0	0:5:3:5
31	1:0:0:0:0	0:5:3:5:1
32	1:0:0:0:0	0:5:3:5:1
33	1:1:0:0:0	0:5:3:5
34	1:1:0:0:0	0:5:3:5
35	1:1:0:0:0	0:5:3:5:1
36	1:1:0:0:0	0:5:3:5:1
37	1:1:0:0:0	0:5:3:5:1:2
38	1:1:0:0:0	0:5:3:5:1:2:8
4	1:1:0:0:0	0:5:3:5:1:2:8:39
5	1:1:0:0:0	0:5:3:5:1:2:8:39:8
6	1:1:0:0:0	0:5:3:5:1:2:8:39:8
7	1:1:0:0:0	0:5:3:5:1:2:8:39:8

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1: 0: 0: 0: 0
2	1: 0: 0: 0: 0 : 5
27	1: 0: 0: 0: 0 : 5: 3
28	1: 0: 0: 0: 0 : 5: 3: 5
29	1: 0: 0: 0: 0 : 5: 3: 5
30	1: 0: 0: 0: 0 : 5: 3: 5
31	1: 0: 0: 0: 0 : 5: 3: 5: 1
32	1: 0: 0: 0: 0 : 5: 3: 5: 1
33	1: 1: 0: 0: 0 : 5: 3: 5
34	1: 1: 0: 0: 0 : 5: 3: 5
35	1: 1: 0: 0: 0 : 5: 3: 5: 1
36	1: 1: 0: 0: 0 : 5: 3: 5: 1
37	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2
38	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8
4	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39
5	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
6	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
7	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
8	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8: 1

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1:1

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1:1
10	1:1:0:0:0:0:5:3:5:1:2:8:39:8:0

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;          % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;      % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;   % y=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;      % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;      % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;          % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;           % y=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1:1
10	1:1:0:0:0:0:5:3:5:1:2:8:39:8:0
24	1:1:0:0:0:0:5:3:5:1:2:8:39:8

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1:1
10	1:1:0:0:0:0:5:3:5:1:2:8:39:8:0
24	1:1:0:0:0:0:5:3:5:1:2:8:39:8
25	1:1:0:0:0:0:5:3:5:1:2:8:39:8

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;          % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;      % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;   % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;      % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;       % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;          % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;           % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;       % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1:1
10	1:1:0:0:0:0:5:3:5:1:2:8:39:8:0
24	1:1:0:0:0:0:5:3:5:1:2:8:39:8
25	1:1:0:0:0:0:5:3:5:1:2:8:39:8
26	1:1:0:0:0:0:5:3:5:1:2:8:39

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;          % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;      % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;   % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;      % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;       % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;          % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;           % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;    % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;       % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1:1
10	1:1:0:0:0:0:5:3:5:1:2:8:39:8:0
24	1:1:0:0:0:0:5:3:5:1:2:8:39:8
25	1:1:0:0:0:0:5:3:5:1:2:8:39:8
26	1:1:0:0:0:0:5:3:5:1:2:8:39
39	1:1:0:0:0:0:5:3:5

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;          % entry F
5: LOAD FP, SP;
6: ADD SP, 0;
7: PUSH <FP-4>;      % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR, <FP-3>;  % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR, <FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;      % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR, <FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP, FP;      % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;          % entry MAIN
28: LOAD FP, SP;
29: ADD SP, 0;
30: PUSH 1;           % y:=1
31: LOAD IR, <FP-2>;
32: POP <IR-3>;
33: LOAD IR, <FP-2>;  % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR, <FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP, FP;      % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1: 0: 0: 0: 0
2	1: 0: 0: 0: 0 : 5
27	1: 0: 0: 0: 0 : 5: 3
28	1: 0: 0: 0: 0 : 5: 3: 5
29	1: 0: 0: 0: 0 : 5: 3: 5
30	1: 0: 0: 0: 0 : 5: 3: 5
31	1: 0: 0: 0: 0 : 5: 3: 5: 1
32	1: 0: 0: 0: 0 : 5: 3: 5: 1
33	1: 1: 0: 0: 0 : 5: 3: 5
34	1: 1: 0: 0: 0 : 5: 3: 5
35	1: 1: 0: 0: 0 : 5: 3: 5: 1
36	1: 1: 0: 0: 0 : 5: 3: 5: 1
37	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2
38	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8
4	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39
5	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
6	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
7	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
8	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8: 1
9	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8: 1: 1
10	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8: 0
24	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
25	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
26	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39
39	1: 1: 0: 0: 0 : 5: 3: 5
40	1: 1: 0: 0: 0 : 5: 3: 5

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0:0
2	1:0:0:0:0:0:5
27	1:0:0:0:0:0:5:3
28	1:0:0:0:0:0:5:3:5
29	1:0:0:0:0:0:5:3:5
30	1:0:0:0:0:0:5:3:5
31	1:0:0:0:0:0:5:3:5:1
32	1:0:0:0:0:0:5:3:5:1
33	1:1:0:0:0:0:5:3:5
34	1:1:0:0:0:0:5:3:5
35	1:1:0:0:0:0:5:3:5:1
36	1:1:0:0:0:0:5:3:5:1
37	1:1:0:0:0:0:5:3:5:1:2
38	1:1:0:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:0:5:3:5:1:2:8:39:8:1:1
10	1:1:0:0:0:0:5:3:5:1:2:8:39:8:0
24	1:1:0:0:0:0:5:3:5:1:2:8:39:8
25	1:1:0:0:0:0:5:3:5:1:2:8:39:8
26	1:1:0:0:0:0:5:3:5:1:2:8:39
39	1:1:0:0:0:0:5:3:5
40	1:1:0:0:0:0:5:3:5
41	1:1:0:0:0:0:5:3

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP,SP;
6: ADD SP,0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR,<FP-3>;    % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR,<FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR,<FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP,FP;        % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP,SP;
29: ADD SP,0;
30: PUSH 1;             % y:=1
31: LOAD IR,<FP-2>;
32: POP <IR-3>;
33: LOAD IR,<FP-2>;     % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR,<FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP,FP;        % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1:0:0:0:0
2	1:0:0:0:0:5
27	1:0:0:0:0:5:3
28	1:0:0:0:0:5:3:5
29	1:0:0:0:0:5:3:5
30	1:0:0:0:0:5:3:5
31	1:0:0:0:0:5:3:5:1
32	1:0:0:0:0:5:3:5:1
33	1:1:0:0:0:5:3:5
34	1:1:0:0:0:5:3:5
35	1:1:0:0:0:5:3:5:1
36	1:1:0:0:0:5:3:5:1
37	1:1:0:0:0:5:3:5:1:2
38	1:1:0:0:0:5:3:5:1:2:8
4	1:1:0:0:0:5:3:5:1:2:8:39
5	1:1:0:0:0:5:3:5:1:2:8:39:8
6	1:1:0:0:0:5:3:5:1:2:8:39:8
7	1:1:0:0:0:5:3:5:1:2:8:39:8
8	1:1:0:0:0:5:3:5:1:2:8:39:8:1
9	1:1:0:0:0:5:3:5:1:2:8:39:8:1:1
10	1:1:0:0:0:5:3:5:1:2:8:39:8:0
24	1:1:0:0:0:5:3:5:1:2:8:39:8
25	1:1:0:0:0:5:3:5:1:2:8:39:8
26	1:1:0:0:0:5:3:5:1:2:8:39
39	1:1:0:0:0:5:3:5
40	1:1:0:0:0:5:3:5
41	1:1:0:0:0:5:3
3	1:1:0:0:0:0

Translation Example III

Example 21.1 (Factorial function; continued)

Altogether:

```

1: PUSH FP;
2: CALL 27;
3: JMP 0;
4: PUSH FP;           % entry F
5: LOAD FP, SP;
6: ADD SP, 0;
7: PUSH <FP-4>;       % x>1
8: PUSH 1;
9: GT;
10: JFALSE 24;
11: LOAD IR, <FP-3>;   % y:=y*x
12: PUSH <IR>;
13: PUSH <FP-4>;
14: MULT;
15: LOAD IR, <FP-3>;
16: POP <IR>;
17: PUSH <FP-4>;       % F(x-1;y)
18: PUSH 1;
19: SUB;
20: PUSH <FP-3>;
21: LOAD IR, <FP-2>;
22: PUSH IR;
23: CALL 4;
24: LOAD SP, FP;       % exit F
25: POP FP;
26: RET 3;
27: PUSH FP;           % entry MAIN
28: LOAD FP, SP;
29: ADD SP, 0;
30: PUSH 1;             % y:=1
31: LOAD IR, <FP-2>;
32: POP <IR-3>;
33: LOAD IR, <FP-2>;   % F(x;y)
34: PUSH <IR-4>;
35: LOAD IR, <FP-2>;
36: PUSH IR-3;
37: PUSH FP;
38: CALL 4;
39: LOAD SP, FP;       % exit MAIN
40: POP FP;
41: RET 1;

```

For $x = 1$, $y = 0$ (bottom = $p(1)$, top = SP , FP , IR):

<i>PC</i>	<i>RS</i>
1	1: 0: 0: 0: 0
2	1: 0: 0: 0: 0 : 5
27	1: 0: 0: 0: 0 : 5: 3
28	1: 0: 0: 0: 0 : 5: 3: 5
29	1: 0: 0: 0: 0 : 5: 3: 5
30	1: 0: 0: 0: 0 : 5: 3: 5
31	1: 0: 0: 0: 0 : 5: 3: 5: 1
32	1: 0: 0: 0: 0 : 5: 3: 5: 1
33	1: 1: 0: 0: 0 : 5: 3: 5
34	1: 1: 0: 0: 0 : 5: 3: 5
35	1: 1: 0: 0: 0 : 5: 3: 5: 1
36	1: 1: 0: 0: 0 : 5: 3: 5: 1
37	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2
38	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8
4	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39
5	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
6	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
7	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
8	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8: 1
9	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8: 1: 1
10	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8: 0
24	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
25	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39: 8
26	1: 1: 0: 0: 0 : 5: 3: 5: 1: 2: 8: 39
39	1: 1: 0: 0: 0 : 5: 3: 5
40	1: 1: 0: 0: 0 : 5: 3: 5
41	1: 1: 0: 0: 0 : 5: 3
3	1: 1: 0: 0: 0
0	1: 1: 0: 0: 0

- 1 Correction: update Function
- 2 Repetition: Procedures with Parameters
- 3 A Translation Example
- 4 Alternative Implementation of Static Links
- 5 Intermediate Code for Data Structures
- 6 Static Data Structures
- 7 Modifying the Abstract Machine
- 8 Translation of EPL with Static Data Structures into AM Programs

- **Alternative implementation** of static link dereferencing:
display technique

- **Alternative implementation** of static link dereferencing:
display technique
- Display directly returns frame pointer for given level difference

- **Alternative implementation** of static link dereferencing:
display technique
- Display directly returns frame pointer for given level difference
- Usually organized as array

- **Alternative implementation** of static link dereferencing:
display technique
- Display directly returns frame pointer for given level difference
- Usually organized as array
- Possible implementations:
 - Local displays: called procedure maintains display array in its frame
 - Global displays: frame pointer stored in global Static Link Array (SLA)

- **Alternative implementation** of static link dereferencing:
display technique
- Display directly returns frame pointer for given level difference
- Usually organized as array
- Possible implementations:
 - Local displays: called procedure maintains display array in its frame
 - Global displays: frame pointer stored in global Static Link Array (SLA)
- For details see Aho/Lam/Sethi/Ullman: *Compilers: Principles, Techniques, and Tools*, 2nd ed., p. 449 ff)

- 1 Correction: update Function
- 2 Repetition: Procedures with Parameters
- 3 A Translation Example
- 4 Alternative Implementation of Static Links
- 5 Intermediate Code for Data Structures**
- 6 Static Data Structures
- 7 Modifying the Abstract Machine
- 8 Translation of EPL with Static Data Structures into AM Programs

Translation of Data Structures

Source code: **data structures** = arrays, records, lists, trees, ...
⇒ **structured** state space, variables with components

Translation of Data Structures

Source code: **data structures** = arrays, records, lists, trees, ...

⇒ **structured** state space, variables with components

Abstract machine: **linear** memory structure, cells for storing atomic data

Translation of Data Structures

Source code: **data structures** = arrays, records, lists, trees, ...
⇒ **structured** state space, variables with components

Abstract machine: **linear** memory structure, cells for storing atomic data

Translation: **mapping** of structured state space to linear memory
(= **address computation**)

- **static** data structures: memory requirements known at compile time
- **dynamic** data structures: memory requirements runtime dependent
⇒ heap, pointers, garbage collection, ...

Translation of Data Structures

Source code: **data structures** = arrays, records, lists, trees, ...
⇒ **structured** state space, variables with components

Abstract machine: **linear** memory structure, cells for storing atomic data

Translation: **mapping** of structured state space to linear memory
(= **address computation**)

- **static** data structures: memory requirements known at compile time
- **dynamic** data structures: memory requirements runtime dependent
⇒ heap, pointers, garbage collection, ...

First step:

- static data structures (arrays and records)
- inductive type definitions
- no procedures (for simplification; “orthogonal” extension)

- 1 Correction: update Function
- 2 Repetition: Procedures with Parameters
- 3 A Translation Example
- 4 Alternative Implementation of Static Links
- 5 Intermediate Code for Data Structures
- 6 Static Data Structures**
- 7 Modifying the Abstract Machine
- 8 Translation of EPL with Static Data Structures into AM Programs

Modified Syntax of EPL

Definition 21.2 (Modified syntax of EPL)

The **modified syntax of EPL** is defined as follows (where $n \geq 1$):

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

$\mathbb{B} :$ $b ::= \text{true} \mid \text{false}$

$\mathbb{R} :$ r (* r is a real number *)

Con : $c ::= z \mid b \mid r$

Ide : I (* I is an identifier *)

Type : $T ::= \text{bool} \mid \text{int} \mid \text{real} \mid I \mid \text{array}[z_1..z_2] \text{ of } T \mid$
 $\text{record } I_1:T_1;\dots;I_n:T_n \text{ end}$

Var : $V ::= I \mid V[E] \mid V.I$

Exp : $E ::= c \mid V \mid E_1 + E_2 \mid E_1 < E_2 \mid E_1 \text{ and } E_2 \mid \dots$

Cmd : $C ::= \textcolor{red}{V}:=E \mid C_1;C_2 \mid$
 $\text{if } E \text{ then } C_1 \text{ else } C_2 \mid \text{while } E \text{ do } C$

Dcl : $D ::= D_C \textcolor{red}{D}_T D_V$

$D_C ::= \varepsilon \mid \text{const } I_1:=\textcolor{red}{c}_1;\dots;I_n:=\textcolor{red}{c}_n;$

$\textcolor{red}{D}_T ::= \varepsilon \mid \text{type } I_1:=T_1;\dots;I_n:=T_n;$

$D_V ::= \varepsilon \mid \text{var } I_1:\textcolor{red}{T}_1;\dots;I_n:\textcolor{red}{T}_n;$

Pgm : $P ::= \textcolor{red}{D} \textcolor{red}{C}$

- All identifiers in a declaration D have to be different.

- All identifiers in a declaration D have to be **different**.
- In $T = \text{record } I_1:T_1;\dots;I_n:T_n \text{ end}$, all selectors I_j must be **different**.

- All identifiers in a declaration D have to be **different**.
- In $T = \text{record } I_1:T_1;\dots;I_n:T_n \text{ end}$, all selectors I_j must be **different**.
- In $T = \text{array}[z_1..z_2] \text{ of } T$, $z_1 \leq z_2$.

- All identifiers in a declaration D have to be **different**.
- In $T = \text{record } I_1:T_1;\dots;I_n:T_n \text{ end}$, all selectors I_j must be **different**.
- In $T = \text{array}[z_1..z_2] \text{ of } T, z_1 \leq z_2$.
- Type definitions must **not be recursive**:
if $D_T = \text{type } I_1:=T_1;\dots;I_n:=T_n$; and identifier I occurs in T_j ,
then $I \in \{I_1, \dots, I_{j-1}\}$.

- All identifiers in a declaration D have to be **different**.
- In $T = \text{record } I_1:T_1;\dots;I_n:T_n \text{ end}$, all selectors I_j must be **different**.
- In $T = \text{array}[z_1..z_2] \text{ of } T, z_1 \leq z_2$.
- Type definitions must **not be recursive**:
if $D_T = \text{type } I_1:=T_1;\dots;I_n:=T_n$; and identifier I occurs in T_j ,
then $I \in \{I_1, \dots, I_{j-1}\}$.
- The type identifiers used in in a variable declaration D_V must be **declared**.

- All identifiers in a declaration D have to be **different**.
- In $T = \text{record } I_1:T_1;\dots;I_n:T_n \text{ end}$, all selectors I_j must be **different**.
- In $T = \text{array}[z_1..z_2] \text{ of } T, z_1 \leq z_2$.
- Type definitions must **not be recursive**:
if $D_T = \text{type } I_1:=T_1;\dots;I_n:=T_n$; and identifier I occurs in T_j ,
then $I \in \{I_1, \dots, I_{j-1}\}$.
- The type identifiers used in in a variable declaration D_V must be **declared**.
- Every identifier used in a command C must be **declared** in D (as a constant or variable).

- All identifiers in a declaration D have to be **different**.
- In $T = \text{record } I_1:T_1;\dots;I_n:T_n \text{ end}$, all selectors I_j must be **different**.
- In $T = \text{array}[z_1..z_2] \text{ of } T, z_1 \leq z_2$.
- Type definitions must **not be recursive**:
if $D_T = \text{type } I_1:=T_1;\dots;I_n:=T_n$; and identifier I occurs in T_j ,
then $I \in \{I_1, \dots, I_{j-1}\}$.
- The type identifiers used in a variable declaration D_V must be **declared**.
- Every identifier used in a command C must be **declared** in D (as a constant or variable).
- Variables in expressions and assignments have a **base type** (bool/int/real; possibly via type identifiers).

- Array **indices** must have type `int`.

Static Semantics II

- Array **indices** must have type `int`.
- **Execution conditions** (`while`) and **branching expressions** (`if`) must have type `bool`.

- Array **indices** must have type **int**.
- **Execution conditions** (**while**) and **branching expressions** (**if**) must have type **bool**.
- The types of the left-hand side and of the right-hand side types of an assignment must be **compatible**.

Static Semantics II

- Array **indices** must have type `int`.
- **Execution conditions** (`while`) and **branching expressions** (`if`) must have type `bool`.
- The types of the left-hand side and of the right-hand side types of an assignment must be **compatible**.
- **Type compatibility**: $\mathbb{Z} \subseteq \mathbb{R}$ in mathematics, but not on computers (different representation)

⇒ **type casts**

weak typing: implicit casting by compiler (`2.5 + 1`, `1 + "42"`)

⇒ risc of undetected “real” errors;

for **programming-in-the-small** (script languages)

strong typing: explicit casting by programmer

⇒ enhanced software reliability;

for **programming-in-the-large**

Static Semantics II

- Array **indices** must have type `int`.
- **Execution conditions** (`while`) and **branching expressions** (`if`) must have type `bool`.
- The types of the left-hand side and of the right-hand side types of an assignment must be **compatible**.
- **Type compatibility**: $\mathbb{Z} \subseteq \mathbb{R}$ in mathematics, but not on computers (different representation)

⇒ **type casts**

weak typing: implicit casting by compiler (`2.5 + 1`, `1 + "42"`)

⇒ risc of undetected “real” errors;

for **programming-in-the-small** (script languages)

strong typing: explicit casting by programmer

⇒ enhanced software reliability;

for **programming-in-the-large**

- Instantiation of operators/functions/procedures/... for different parameter types: **polymorphism** or **overloading**

`+` : `int` × `int` → `int` `+` : `real` × `real` → `real`

- 1 Correction: update Function
- 2 Repetition: Procedures with Parameters
- 3 A Translation Example
- 4 Alternative Implementation of Static Links
- 5 Intermediate Code for Data Structures
- 6 Static Data Structures
- 7 Modifying the Abstract Machine**
- 8 Translation of EPL with Static Data Structures into AM Programs

Since (recursive) procedures are no longer supported, a procedure stack is not required anymore.

Definition 21.3 (Modified abstract machine for EPL)

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

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

with

- the **program counter** $PC := \mathbb{N}$,
- the **data stack** $DS := \mathbb{R}^*$, and
- the **main storage** $MS := \{\sigma \mid \sigma : \mathbb{N} \rightarrow \mathbb{R}\}$.

Definition 21.4 (New AM instructions)

- **Procedure instructions** are no longer needed.

Definition 21.4 (New AM instructions)

- **Procedure instructions** are no longer needed.
- **Transfer instructions** ($\text{LOAD}(dif, off)$, $\text{STORE}(dif, off)$) are replaced by the following instructions with the respective semantics $\llbracket O \rrbracket : S \dashrightarrow S$:

$$\begin{aligned}\llbracket \text{LOAD} \rrbracket(a, d : n, \sigma) &:= (a + 1, d : \sigma(n), \sigma) \\ &\quad \text{if } n \in \mathbb{N} \\ \llbracket \text{STORE} \rrbracket(a, d : n : r, \sigma) &:= (a + 1, d, \sigma[n \mapsto r]) \\ &\quad \text{if } n \in \mathbb{N}\end{aligned}$$

Definition 21.4 (New AM instructions)

- **Procedure instructions** are no longer needed.
- **Transfer instructions** ($\text{LOAD}(dif, off)$, $\text{STORE}(dif, off)$) are replaced by the following instructions with the respective semantics $\llbracket O \rrbracket : S \dashrightarrow S$:

$$\begin{aligned}\llbracket \text{LOAD} \rrbracket(a, d : n, \sigma) &:= (a + 1, d : \sigma(n), \sigma) \\ &\quad \text{if } n \in \mathbb{N} \\ \llbracket \text{STORE} \rrbracket(a, d : n : r, \sigma) &:= (a + 1, d, \sigma[n \mapsto r]) \\ &\quad \text{if } n \in \mathbb{N}\end{aligned}$$

- Moreover the following **instruction for checking array bounds** is introduced:

$$\llbracket \text{CAB}(z_1, z_2) \rrbracket(a, d : z, \sigma) := \begin{cases} (a + 1, d : z, \sigma) & \text{if } z \in \{z_1, \dots, z_2\} \\ (0, d : \underbrace{\text{RTE}}_{\text{runtime error}}, \sigma) & \text{otherwise} \end{cases}$$