Compiler Construction

Lecture 16: Code Generation II (The Translator)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ss-14/cc14/

Summer Semester 2014

Online Registration for Seminars and Practical Courses (Praktika) in Winter Term 2014/15

Who?

Students of:
- Master Courses
- Bachelor Informatik (~~Pro~~Seminar!)

Where?

www.graphics.rwth-aachen.de/apse

When?

04.07.2014 - 20.07.2014

Outline

Syntax of EPL

Definition (Syntax of EPL)

The syntax of EPL is defined as follows:

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

The Abstract Machine AM

Definition (Abstract machine for EPL)

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

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

with

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

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

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

Structure of Procedure Stack I

The semantics of procedure and transfer instructions requires a particular structure of the procedure stack $p \in PS$: it must be composed of frames (or: activation records) of the form

$$sl : dl : ra : v_1 : \ldots : v_k$$

where

static link sl: points to frame of surrounding declaration environment
$\Longrightarrow$ used to access non-local variables

dynamic link dl: points to previous frame (i.e., of calling procedure)
$\Longrightarrow$ used to remove topmost frame after termination of procedure call

return address ra: program label after termination of procedure call
$\Longrightarrow$ used to continue program execution after termination of procedure call

local variables v_i: values of locally declared variables

Structure of Procedure Stack II

- Frames are created whenever a procedure call is performed
- Two special frames:

  I/O frame: for keeping values of `in/out` variables ($sl = dl = ra = 0$)

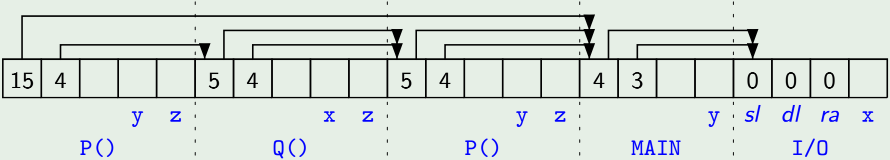
  MAIN frame: for keeping values of top-level block ($sl = dl =$ I/O frame)

Structure of Procedure Stack III

Example (cf. Example 15.4)

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

Procedure stack after second call of P:

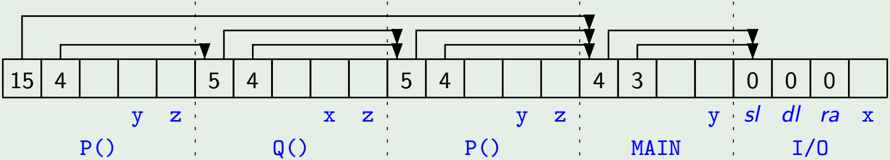


Structure of Procedure Stack III

Example (cf. Example 15.4)

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

Procedure stack after second call of P:



Structure of Procedure Stack IV

Observation:

- The usage of a variable in a procedure body refers to its innermost declaration.
- If the level difference between the usage and the declaration is *dif*, then a chain of *dif* static links has to be followed to access the corresponding frame.

Structure of Procedure Stack IV

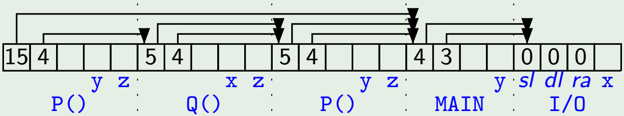
Observation:

- The usage of a variable in a procedure body refers to its innermost declaration.
- If the level difference between the usage and the declaration is *dif*, then a chain of *dif* static links has to be followed to access the corresponding frame.

Example (cf. Example 15.9)

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

Procedure stack after second call of P:



Structure of Procedure Stack IV

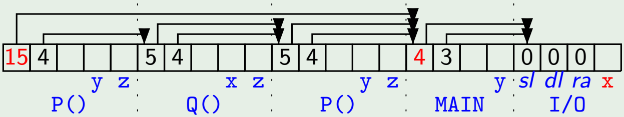
Observation:

- The usage of a variable in a procedure body refers to its innermost declaration.
- If the level difference between the usage and the declaration is *dif*, then a chain of *dif* static links has to be followed to access the corresponding frame.

Example (cf. Example 15.9)

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

Procedure stack after second call of P:



P uses x $\Longrightarrow$ *dif* $= 2$

Structure of Procedure Stack IV

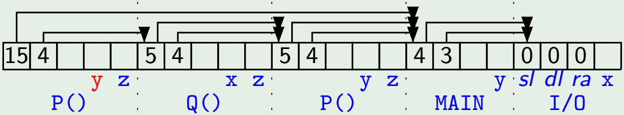
Observation:

- The usage of a variable in a procedure body refers to its innermost declaration.
- If the level difference between the usage and the declaration is *dif*, then a chain of *dif* static links has to be followed to access the corresponding frame.

Example (cf. Example 15.9)

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

Procedure stack after second call of P:



P uses y $\Longrightarrow$ $dif = 0$

Outline

The base Function

Upon procedure call, the static link information is computed by the following auxiliary function which, given a procedure stack and a level difference, determines the begin of the corresponding frame.

Definition 16.1 (base function)

The function

$$\mathrm{base} : PS \times \mathbb{N} \dashrightarrow \mathbb{N}$$

is given by

$$\begin{aligned}\mathrm{base}(p, 0) &:= 1\\ \mathrm{base}(p, dif + 1) &:= \mathrm{base}(p, dif) + p.\mathrm{base}(p, dif)\end{aligned}$$

The base Function

Upon procedure call, the static link information is computed by the following auxiliary function which, given a procedure stack and a level difference, determines the begin of the corresponding frame.

Definition 16.1 (base function)

The function

$$\text{base} : PS \times \mathbb{N} \dashrightarrow \mathbb{N}$$

is given by

$$\begin{aligned} \text{base}(p, 0) &:= 1 \\ \text{base}(p, dif + 1) &:= \text{base}(p, dif) + p.\text{base}(p, dif) \end{aligned}$$

Example 16.2 (cf. Example 15.10)

In the second call of `P` (from `Q`): $dif = 2$

$$\text{base}(p, 0) = 1$$

The base Function

Upon procedure call, the static link information is computed by the following auxiliary function which, given a procedure stack and a level difference, determines the begin of the corresponding frame.

Definition 16.1 (base function)

The function

$$\mathrm{base} : PS \times \mathbb{N} \dashrightarrow \mathbb{N}$$

is given by

$$\begin{aligned} \mathrm{base}(p, 0) &:= 1 \\ \mathrm{base}(p, \mathit{dif} + 1) &:= \mathrm{base}(p, \mathit{dif}) + p.\mathrm{base}(p, \mathit{dif}) \end{aligned}$$

Example 16.2 (cf. Example 15.10)

In the second call of `P` (from `Q`): $\mathit{dif} = 2$

$$\begin{aligned} & \mathrm{base}(p, 0) = 1 \\ \Longrightarrow\ & \mathrm{base}(p, 1) = 1 + p.1 = 6 \end{aligned}$$

The base Function

Upon procedure call, the static link information is computed by the following auxiliary function which, given a procedure stack and a level difference, determines the begin of the corresponding frame.

Definition 16.1 (base function)

The function

$$\mathrm{base} : PS \times \mathbb{N} \dashrightarrow \mathbb{N}$$

is given by

$$\begin{aligned} \mathrm{base}(p, 0) &:= 1 \\ \mathrm{base}(p, dif + 1) &:= \mathrm{base}(p, dif) + p.\mathrm{base}(p, dif) \end{aligned}$$

Example 16.2 (cf. Example 15.10)

In the second call of `P` (from `Q`): $dif = 2$

$$\begin{aligned} & \mathrm{base}(p, 0) = 1 \\ \Longrightarrow\ & \mathrm{base}(p, 1) = 1 + p.1 = 6 \\ \Longrightarrow\ & \mathrm{base}(p, 2) = 6 + p.6 = 11 \end{aligned}$$

The base Function

Upon procedure call, the static link information is computed by the following auxiliary function which, given a procedure stack and a level difference, determines the begin of the corresponding frame.

Definition 16.1 (base function)

The function

$$\mathrm{base} : PS \times \mathbb{N} \dashrightarrow \mathbb{N}$$

is given by

$$\begin{aligned} \mathrm{base}(p, 0) &:= 1 \\ \mathrm{base}(p, dif + 1) &:= \mathrm{base}(p, dif) + p.\mathrm{base}(p, dif) \end{aligned}$$

Example 16.2 (cf. Example 15.10)

In the second call of `P` (from `Q`): $dif = 2$

$$\begin{aligned} & \mathrm{base}(p, 0) = 1 \\ \Longrightarrow\ & \mathrm{base}(p, 1) = 1 + p.1 = 6 \\ \Longrightarrow\ & \mathrm{base}(p, 2) = 6 + p.6 = 11 \end{aligned}$$

$$\Longrightarrow\ sl = \mathrm{base}(p, 2) + \underbrace{2}_{\mathtt{y,z}} + \underbrace{2}_{ra,dl} = 15$$

Semantics of Procedure Instructions

- `CALL(`ca, dif, loc`)` with
  - code address $ca \in PC$
  - level difference $dif \in \mathbb{N}$
  - number of local variables $loc \in \mathbb{N}$

  creates the new frame and jumps to the given address
  (= starting address of procedure)
- `RET` removes the topmost frame and returns to the calling site

Semantics of Procedure Instructions

- `CALL`(ca, dif, loc) with
  - code address $ca \in PC$
  - level difference $dif \in \mathbb{N}$
  - number of local variables $loc \in \mathbb{N}$

  creates the new frame and jumps to the given address (= starting address of procedure)
- `RET` removes the topmost frame and returns to the calling site

Definition 16.3 (Semantics of procedure instructions)

The semantics of a procedure instruction O, $[\![O]\!] : S \dashrightarrow S$, is defined as follows:

$$[\![\texttt{CALL}(ca, dif, loc)]\!](l, d, p) := (ca, d, \underbrace{(\mathrm{base}(p, dif) + loc + 2)}_{sl} : \underbrace{(loc + 2)}_{dl} : \underbrace{(l + 1)}_{ra} : \underbrace{0 : \ldots : 0}_{loc \text{ times}} : p)$$

$$[\![\texttt{RET}]\!](l, d, p.1 : \ldots : p.t) := (\underbrace{p.3}_{ra}, d, p.(\underbrace{p.2}_{dl} + 2) : \ldots : p.t) \quad \text{if } t \geq p.2 + 2$$

Semantics of Transfer Instructions

- `LOAD`(dif, off) and `STORE`(dif, off) with
  - level difference $dif \in \mathbb{N}$
  - variable offset $off \in \mathbb{N}$

  respectively load and store variable values between data and procedure stack, following a chain of dif static links
- `LIT`(z) loads the literal constant $z \in \mathbb{Z}$

Semantics of Transfer Instructions

- $\texttt{LOAD(}\mathit{dif}\texttt{,}\mathit{off}\texttt{)}$ and $\texttt{STORE(}\mathit{dif}\texttt{,}\mathit{off}\texttt{)}$ with
  - level difference $\mathit{dif} \in \mathbb{N}$
  - variable offset $\mathit{off} \in \mathbb{N}$

  respectively load and store variable values between data and procedure stack, following a chain of dif static links
- $\texttt{LIT(}z\texttt{)}$ loads the literal constant $z \in \mathbb{Z}$

Definition 16.4 (Semantics of transfer instructions)

The semantics of a transfer instruction O, $[\![O]\!] : S \dashrightarrow S$, is defined as follows:

$$[\![\texttt{LOAD(}\mathit{dif}\texttt{,}\mathit{off}\texttt{)}]\!](l, d, p) := (l + 1, d : p.(\mathrm{base}(p, \mathit{dif}) + \mathit{off} + 2), p)$$
$$[\![\texttt{STORE(}\mathit{dif}\texttt{,}\mathit{off}\texttt{)}]\!](l, d : z, p) := (l + 1, d, p[\mathrm{base}(p, \mathit{dif}) + \mathit{off} + 2 \mapsto z])$$
$$[\![\texttt{LIT(}z\texttt{)}]\!](l, d, p) := (l + 1, d : z, p)$$

AM Programs and Their Semantics

Definition 16.5 (Semantics of AM programs)

An AM program is a sequence of $k \geq 1$ labeled AM instructions:

$$P = 1 : O_1; \ldots; k : O_k$$

The set of all AM programs is denoted by AM.

AM Programs and Their Semantics

Definition 16.5 (Semantics of AM programs)

An AM program is a sequence of $k \geq 1$ labeled AM instructions:

$$P = 1 : O_1; \ldots ; k : O_k$$

The set of all AM programs is denoted by AM.

The semantics of AM programs is determined by

$$[\![.]\!] : AM \times S \dashrightarrow S$$

with

$$[\![P]\!](l, d, p) := \begin{cases} [\![P]\!]([\![O_l]\!](l, d, p)) & \text{if } l \in [k] \\ (l, d, p) & \text{otherwise} \end{cases}$$

Outline

Structure of Symbol Table

Goal: define translation mapping $\mathrm{trans} : Pgm \dashrightarrow AM$

Structure of Symbol Table

Goal: define translation mapping $\mathrm{trans} : Pgm \dashrightarrow AM$

The translation employs a symbol table:

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

whose entries are created by declarations:

- constant declarations: $(\mathtt{const}, z)$
  - value $z \in \mathbb{Z}$
- variable declarations: $(\mathtt{var}, lev, off)$
  - declaration level $lev \in Lev := \mathbb{N}$ ($0 \cong$ I/O, $1 \cong$ MAIN, ...)
  - offset $off \in Off := \mathbb{N}$
  - offset and difference between usage and declaration level determine procedure stack entry
- procedure declarations: $(\mathtt{proc}, ca, lev, loc)$
  - code address $ca \in PC$
  - declaration level $lev \in Lev$
  - number of local variables $loc \in Size := \mathbb{N}$

Maintaining the Symbol Table

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

Maintaining the Symbol Table

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

Definition 16.6 (update function)

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

is defined by

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

The Initial Symbol Table

Reminder: an EPL program $P = \texttt{in/out}\ I_1, \ldots, I_n; K\,.\in Pgm$ has a semantics of type $\mathbb{Z}^n \dashrightarrow \mathbb{Z}^n$.

The Initial Symbol Table

Reminder: an EPL program $P = \texttt{in/out}\ I_1, \ldots, I_n; K. \in Pgm$ has a semantics of type $\mathbb{Z}^n \dashrightarrow \mathbb{Z}^n$.

Given input values $(z_1, \ldots, z_n) \in \mathbb{Z}^n$, we choose the initial state

$$s := (1, \varepsilon, \underbrace{0 : 0 : 0 : z_1 : \ldots : z_n}_{\text{I/O frame}}) \in S = PC \times DS \times PS$$

The Initial Symbol Table

Reminder: an EPL program $P = \texttt{in/out}\ I_1\,,\ldots\,,I_n\,;K\,.\in Pgm$ has a semantics of type $\mathbb{Z}^n \dashrightarrow \mathbb{Z}^n$.

Given input values $(z_1,\ldots,z_n) \in \mathbb{Z}^n$, we choose the initial state

$$s := (1, \varepsilon, \underbrace{0 : 0 : 0 : z_1 : \ldots : z_n}_{\text{I/O frame}}) \in S = PC \times DS \times PS$$

Thus the corresponding initial symbol table has n entries:

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

Outline

Translation of Programs

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

1. Create MAIN frame for executing C
2. Stop program execution after return

Translation of Programs

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

1. Create MAIN frame for executing C
2. Stop program execution after return

Definition 16.7 (Translation of programs)

The mapping

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

is defined by

$$\begin{aligned}\mathrm{trans}(\texttt{in/out}\ I_1, \ldots, I_n; K.) := {} & 1 : \texttt{CALL}(a, 0, \mathrm{size}(K));\\ & 2 : \texttt{JMP}(0);\\ & \mathrm{kt}(K, \mathrm{st}_{I/O}, a, 1)\end{aligned}$$

Outline

Translation of Blocks

Translation of D C:

1. Update symbol table according to D
2. Create code for procedures declared in D
   (using the updated symbol table – recursion!)
3. Create code for C (using the updated symbol table)

Translation of Blocks

Translation of $D\ C$:

1. Update symbol table according to D
2. Create code for procedures declared in D
   (using the updated symbol table – recursion!)
3. Create code for C (using the updated symbol table)

Definition 16.8 (Translation of blocks)

The mapping

$$\mathrm{kt} : Blk \times Tab \times PC \times Lev \dashrightarrow AM$$

("block translation") is defined by

$$\begin{aligned}\mathrm{kt}(D\ C, \mathrm{st}, a, l) := \ & \mathrm{dt}(D, \mathrm{update}(D, \mathrm{st}, l), l) \\ & \mathrm{ct}(C, \mathrm{update}(D, \mathrm{st}, l), a, l) \\ & a' : \mathtt{RET};\end{aligned}$$

Outline

Translation of Declarations

Translation of D: generate code for the procedures declared in D

Translation of Declarations

Translation of D: generate code for the procedures declared in D

Definition 16.9 (Translation of declarations)

The mapping

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

("declaration translation") is defined by

$$
\begin{aligned}
&\mathrm{dt}(D_C\ D_V\ D_P, \mathrm{st}, l) \\
&\quad := \quad \mathrm{dt}(D_P, \mathrm{st}, l) \\
&\mathrm{dt}(\varepsilon, \mathrm{st}, l) \\
&\quad := \quad \varepsilon \\
&\mathrm{dt}(\texttt{proc}\ I_1\texttt{;}K_1\texttt{;}\ \ldots\ \texttt{;proc}\ I_n\texttt{;}K_n\texttt{;}, \mathrm{st}, l) \\
&\quad := \quad \mathrm{kt}(K_1, \mathrm{st}, a_1, l+1) \\
&\qquad\qquad\quad \vdots \\
&\qquad\quad\ \ \mathrm{kt}(K_n, \mathrm{st}, a_n, l+1) \\
&\qquad\quad\ \ \text{where } \mathrm{st}(I_j) = (\texttt{proc}, a_j, \ldots, \ldots) \text{ for every } j \in [n]
\end{aligned}
$$

Outline

Definition 16.10 (Translation of commands)

The mapping

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

("command translation") is defined by

$$\begin{aligned}
\mathrm{ct}(I := A, \mathrm{st}, a, l) &:= \mathrm{at}(A, \mathrm{st}, a, l) \\
&\ a' : \mathtt{STORE}(l - \mathit{lev}, \mathit{off});\\
&\ \text{if } \mathrm{st}(I) = (\mathtt{var}, \mathit{lev}, \mathit{off})\\
\mathrm{ct}(I(), \mathrm{st}, a, l) &:= a : \mathtt{CALL}(\mathit{ca}, l - \mathit{lev}, \mathit{loc});\\
&\ \text{if } \mathrm{st}(I) = (\mathtt{proc}, \mathit{ca}, \mathit{lev}, \mathit{loc})\\
\mathrm{ct}(C_1; C_2, \mathrm{st}, a, l) &:= \mathrm{ct}(C_1, \mathrm{st}, a, l)\\
&\ \mathrm{ct}(C_2, \mathrm{st}, a', l)\\
\mathrm{ct}(\mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, \mathrm{st}, a, l) &:= \mathrm{bt}(B, \mathrm{st}, a, l)\\
&\ a' : \mathtt{JFALSE}(a'');\\
&\ \mathrm{ct}(C_1, \mathrm{st}, a' + 1, l)\\
&\ a'' - 1 : \mathtt{JMP}(a''');\\
&\ \mathrm{ct}(C_2, \mathrm{st}, a'', l)\\
&\ a''' :\\
\mathrm{ct}(\mathtt{while}\ B\ \mathtt{do}\ C, \mathrm{st}, a, l) &:= \mathrm{bt}(B, \mathrm{st}, a, l)\\
&\ a' : \mathtt{JFALSE}(a'' + 1);\\
&\ \mathrm{ct}(C, \mathrm{st}, a' + 1, l)\\
&\ a'' : \mathtt{JMP}(a);
\end{aligned}$$

Outline

Translation of Boolean Expressions

Definition 16.11 (Translation of Boolean expressions)

The mapping

$$\mathrm{bt} : \mathit{BExp} \times \mathit{Tab} \times \mathit{PC} \times \mathit{Lev} \dashrightarrow \mathit{AM}$$

("Boolean expression translation") is defined by

$$
\begin{aligned}
\mathrm{bt}(A_1 < A_2, \mathrm{st}, a, l) &:= \mathrm{at}(A_1, \mathrm{st}, a, l) \\
& \mathrm{at}(A_2, \mathrm{st}, a', l) \\
& a'' : \texttt{LT;} \\
\mathrm{bt}(\texttt{not } B, \mathrm{st}, a, l) &:= \mathrm{bt}(B, \mathrm{st}, a, l) \\
& a' : \texttt{NOT;} \\
\mathrm{bt}(B_1 \texttt{ and } B_2, \mathrm{st}, a, l) &:= \mathrm{bt}(B_1, \mathrm{st}, a, l) \\
& \mathrm{bt}(B_2, \mathrm{st}, a', l) \\
& a'' : \texttt{AND;} \\
\mathrm{bt}(B_1 \texttt{ or } B_2, \mathrm{st}, a, l) &:= \mathrm{bt}(B_1, \mathrm{st}, a, l) \\
& \mathrm{bt}(B_2, \mathrm{st}, a', l) \\
& a'' : \texttt{OR;}
\end{aligned}
$$

Definition 16.12 (Translation of arithmetic expressions)

The mapping

$$\mathrm{at} : AExp \times Tab \times PC \times Lev \dashrightarrow AM$$

("arithmetic expression translation") is defined by

$$\begin{aligned}
\mathrm{at}(z, \mathrm{st}, a, l) &:= a : \mathtt{LIT}(z)\,; \\
\mathrm{at}(I, \mathrm{st}, a, l) &:= \begin{cases} a : \mathtt{LIT}(z)\,; & \text{if } \mathrm{st}(I) = (\mathtt{const}, z) \\ a : \mathtt{LOAD}(l - lev\,, off\,)\,; & \text{if } \mathrm{st}(I) = (\mathtt{var}, lev, off\,) \end{cases} \\
\mathrm{at}(A_1 + A_2, \mathrm{st}, a, l) &:= \mathrm{at}(A_1, \mathrm{st}, a, l) \\
&\ \mathrm{at}(A_2, \mathrm{st}, a', l) \\
&\ a'' : \mathtt{ADD}\,;
\end{aligned}$$

Outline

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

Intermediate code:

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$\text{trans}(\texttt{in/out } I_1, \ldots, I_n; K.) :=$

$\quad 1 : \texttt{CALL}(a, 0, \text{size}(K));$

$\quad 2 : \texttt{JMP}(0);$

$\quad \text{kt}(K, \text{st}_{I/O}, a, 1)$

$\text{st}_{I/O} = [\texttt{x} \mapsto (\texttt{var}, 0, 1)]$

Intermediate code:

$\text{trans}(\texttt{in/out x}; K.)$

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$\mathrm{kt}(D\ C, \mathrm{st}, a, l) :=$
$\quad \mathrm{dt}(D, \mathrm{update}(D, \mathrm{st}, l), l)$
$\quad \mathrm{ct}(C, \mathrm{update}(D, \mathrm{st}, l), a, l)$
$\quad a' : \texttt{RET;}$

$\mathrm{st}_{I/O} = [\mathtt{x} \mapsto (\mathtt{var}, 0, 1)]$

Intermediate code:

$1 : \texttt{CALL(}a_0\texttt{,0,1);}$
$2 : \texttt{JMP(0);}$
$\quad \mathrm{kt}(K, \mathrm{st}_{I/O}, a_0, 1)$

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

Intermediate code:

1 : CALL(a_0,0,1);
2 : JMP(0);
$\mathrm{dt}(D, \mathrm{update}(D, \mathrm{st}_{I/O}, 1), 1)$
$\mathrm{ct}(C, \mathrm{update}(D, \mathrm{st}_{I/O}, 1), a_0, 1)$
a_2 : RET;

$$\mathrm{update}(\texttt{var}\ I_1, \ldots, I_n;, \mathrm{st}, l) := \mathrm{st}[I_1 \mapsto (\texttt{var}, l, 1), \ldots, I_n \mapsto (\texttt{var}, l, n)]$$

$$\mathrm{update}(\texttt{proc}\ I_1; K_1; \ldots; \texttt{proc}\ I_n; K_n;, \mathrm{st}, l) := \mathrm{st}[I_1 \mapsto (\texttt{proc}, a_1, l, \mathrm{size}(K_1)), \ldots, I_n \mapsto (\texttt{proc}, a_n, l, \mathrm{size}(K_n))]$$

$$\mathrm{st}' = [\mathtt{x} \mapsto (\texttt{var}, 0, 1), \mathtt{y} \mapsto (\texttt{var}, 1, 1), \mathtt{F} \mapsto (\texttt{proc}, a_1, 1, 0)]$$

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

Intermediate code:

$1 : \texttt{CALL}(a_0,0,1);$
$2 : \texttt{JMP}(0);$
$\quad \mathrm{dt}(D, \mathrm{st}', 1)$
$\quad \mathrm{ct}(C, \mathrm{st}', a_0, 1)$
$a_2 : \texttt{RET};$

$$\mathrm{dt}(\texttt{proc } I_1; K_1; \ldots; \texttt{proc } I_n; K_n;, \mathrm{st}, l) :=$$
$$\mathrm{kt}(K_1, \mathrm{st}, a_1, l+1)$$
$$\vdots$$
$$\mathrm{kt}(K_n, \mathrm{st}, a_n, l+1)$$

where $\mathrm{st}(I_j) = (\texttt{proc}, a_j, \ldots, \ldots)$ for every $j \in [n]$

$$\mathrm{st}' = [\texttt{x} \mapsto (\texttt{var}, 0, 1), \texttt{y} \mapsto (\texttt{var}, 1, 1), \texttt{F} \mapsto (\texttt{proc}, a_1, 1, 0)]$$

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$\mathrm{kt}(D\ C, \mathrm{st}, a, l) :=$
  $\mathrm{dt}(D, \mathrm{update}(D, \mathrm{st}, l), l)$
  $\mathrm{ct}(C, \mathrm{update}(D, \mathrm{st}, l), a, l)$
  a' : RET;

$$\mathrm{st}' = [\mathtt{x} \mapsto (\mathtt{var}, 0, 1), \mathtt{y} \mapsto (\mathtt{var}, 1, 1), \mathtt{F} \mapsto (\mathtt{proc}, a_1, 1, 0)]$$

Intermediate code:

1 : CALL(a_0,0,1);
2 : JMP(0);
  $\mathrm{kt}(K_{\mathtt{F}}, \mathrm{st}', a_1, 2)$
  $\mathrm{ct}(C, \mathrm{st}', a_0, 1)$
a_2 : RET;

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$\mathrm{ct}(\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2, \mathrm{st}, a, l) :=$

$\quad \mathrm{bt}(B, \mathrm{st}, a, l)$
$\quad a' : \texttt{JFALSE}(a'');$
$\quad \mathrm{ct}(C_1, \mathrm{st}, a' + 1, l)$
$\quad a'' - 1 : \texttt{JMP}(a''');$
$\quad \mathrm{ct}(C_2, \mathrm{st}, a'', l)$
$\quad a''' :$

$$\mathrm{st}' = [\mathtt{x} \mapsto (\mathtt{var}, 0, 1), \mathtt{y} \mapsto (\mathtt{var}, 1, 1), \mathtt{F} \mapsto (\mathtt{proc}, a_1, 1, 0)]$$

Intermediate code:

$1 : \texttt{CALL}(a_0, 0, 1);$
$2 : \texttt{JMP}(0);$
$\quad \mathrm{ct}(C_{\mathtt{F}}, \mathrm{st}', a_1, 2)$
$a_3 : \texttt{RET};$
$\quad \mathrm{ct}(C, \mathrm{st}', a_0, 1)$
$a_2 : \texttt{RET};$

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$$\mathrm{bt}(A_1 > A_2, \mathrm{st}, a, l) := \mathrm{at}(A_1, \mathrm{st}, a, l)\ \mathrm{at}(A_2, \mathrm{st}, a', l)\ a'' : \mathtt{GT};$$

$$\mathrm{st}' = [\mathtt{x} \mapsto (\mathtt{var}, 0, 1), \mathtt{y} \mapsto (\mathtt{var}, 1, 1), \mathtt{F} \mapsto (\mathtt{proc}, a_1, 1, 0)]$$

Intermediate code:

1 : $\mathtt{CALL}(a_0, 0, 1);$
2 : $\mathtt{JMP}(0);$
$\quad \mathrm{bt}(\mathtt{x} > 1, \mathrm{st}', a_1, 2)$
a_4 : $\mathtt{JFALSE}(a_3);$
$\quad \mathrm{ct}(C_1, \mathrm{st}', a_4 + 1, 2)$
a_3 : $\mathtt{RET};$
$\quad \mathrm{ct}(C, \mathrm{st}', a_0, 1)$
a_2 : $\mathtt{RET};$

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

Intermediate code:

$$\begin{array}{rl}
1: & \texttt{CALL(}a_0\texttt{,0,1);}\\
2: & \texttt{JMP(0);}\\
 & \mathrm{at}(\mathtt{x}, \mathrm{st}', a_1, 2)\\
 & \mathrm{at}(1, \mathrm{st}', a', 2)\\
 & \texttt{GT;}\\
a_4: & \texttt{JFALSE(}a_3\texttt{);}\\
 & \mathrm{ct}(C_1, \mathrm{st}', a_4 + 1, 2)\\
a_3: & \texttt{RET;}\\
 & \mathrm{ct}(C, \mathrm{st}', a_0, 1)\\
a_2: & \texttt{RET;}
\end{array}$$

$$\mathrm{at}(I, \mathrm{st}, a, l) := \begin{cases} a : \texttt{LIT(}z\texttt{);} & \text{if } \mathrm{st}(I) = (\texttt{const}, z)\\ a : \texttt{LOAD(}l - \mathit{lev}\texttt{,}\mathit{off}\texttt{);} & \text{if } \mathrm{st}(I) = (\texttt{var}, \mathit{lev}, \mathit{off})\end{cases}$$

$$\begin{aligned}\mathrm{st}' = [&\mathtt{x} \mapsto (\texttt{var}, 0, 1),\\ &\mathtt{y} \mapsto (\texttt{var}, 1, 1),\\ &\mathtt{F} \mapsto (\texttt{proc}, a_1, 1, 0)]\end{aligned}$$

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$\mathrm{at}(z, \mathrm{st}, a, l) := a : \texttt{LIT}(z)\texttt{;}$

$$\begin{aligned}\mathrm{st}' = [&\texttt{x} \mapsto (\texttt{var}, 0, 1),\\ &\texttt{y} \mapsto (\texttt{var}, 1, 1),\\ &\texttt{F} \mapsto (\texttt{proc}, a_1, 1, 0)]\end{aligned}$$

Intermediate code:

$$\begin{aligned}
1 &: \texttt{CALL(}a_0\texttt{,0,1);}\\
2 &: \texttt{JMP(0);}\\
a_1 &: \texttt{LOAD(2,1);}\\
&\quad \mathrm{at}(1, \mathrm{st}', a', 2)\\
&\quad \texttt{GT;}\\
a_4 &: \texttt{JFALSE(}a_3\texttt{);}\\
&\quad \mathrm{ct}(C_1, \mathrm{st}', a_4 + 1, 2)\\
a_3 &: \texttt{RET;}\\
&\quad \mathrm{ct}(C, \mathrm{st}', a_0, 1)\\
a_2 &: \texttt{RET;}
\end{aligned}$$

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$\mathrm{ct}(I := A, \mathrm{st}, a, l) :=$
$\quad \mathrm{at}(A, \mathrm{st}, a, l)$
$\quad a' : \mathtt{STORE}(l - lev, off);$
if $\mathrm{st}(I) = (\mathtt{var}, lev, off)$

$\mathrm{st}' = [\mathtt{x} \mapsto (\mathtt{var}, 0, 1),$
$\quad \mathtt{y} \mapsto (\mathtt{var}, 1, 1),$
$\quad \mathtt{F} \mapsto (\mathtt{proc}, a_1, 1, 0)]$

Intermediate code:

1 : $\mathtt{CALL}(a_0,0,1);$
2 : $\mathtt{JMP(0)};$
a_1 : $\mathtt{LOAD(2,1)};$
$\quad \mathtt{LIT(1)};$
$\quad \mathtt{GT};$
a_4 : $\mathtt{JFALSE}(a_3);$
$\quad \mathrm{ct}(C_1, \mathrm{st}', a_4 + 1, 2)$
a_3 : $\mathtt{RET};$
$\quad \mathrm{ct}(C, \mathrm{st}', a_0, 1)$
a_2 : $\mathtt{RET};$

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$$\begin{aligned}\mathrm{at}(A_1 \texttt{ + } A_2, \mathrm{st}, a, l) := {} & \mathrm{at}(A_1, \mathrm{st}, a, l) \\ & \mathrm{at}(A_2, \mathrm{st}, a', l) \\ & a'' : \texttt{ADD};\end{aligned}$$

$$\begin{aligned}\mathrm{st}' = [& \texttt{x} \mapsto (\texttt{var}, 0, 1), \\ & \texttt{y} \mapsto (\texttt{var}, 1, 1), \\ & \texttt{F} \mapsto (\texttt{proc}, a_1, 1, 0)]\end{aligned}$$

Intermediate code:

$$\begin{aligned}
1 &: \texttt{CALL(}a_0\texttt{,0,1);} \\
2 &: \texttt{JMP(0);} \\
a_1 &: \texttt{LOAD(2,1);} \\
& \texttt{LIT(1);} \\
& \texttt{GT;} \\
a_4 &: \texttt{JFALSE(}a_3\texttt{);} \\
& \mathrm{at}(\texttt{y * x}, \mathrm{st}', a_4 + 1, 2) \\
& \texttt{STORE(1,1);} \\
& \mathrm{at}(\texttt{x - 1}, \mathrm{st}', a', 2) \\
& \texttt{STORE(2,1);} \\
& \mathrm{ct}(\texttt{F()}, \mathrm{st}', a'', 2) \\
& a_3 : \texttt{RET;} \\
& \mathrm{ct}(C, \mathrm{st}', a_0, 1) \\
a_2 &: \texttt{RET;}
\end{aligned}$$

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$\mathrm{ct}(I(), \mathrm{st}, a, l) :=$

$\quad a : \texttt{CALL(}ca\texttt{,}l - lev\texttt{,}loc\texttt{);}$

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

$\mathrm{st}' = [\texttt{x} \mapsto (\texttt{var}, 0, 1),$
$\quad \texttt{y} \mapsto (\texttt{var}, 1, 1),$
$\quad \texttt{F} \mapsto (\texttt{proc}, a_1, 1, 0)]$

Intermediate code:

1 : CALL(a_0,0,1);
2 : JMP(0);
a_1 : LOAD(2,1);
LIT(1);
GT;
a_4 : JFALSE(a_3);
LOAD(1,1);
LOAD(2,1);
MULT;
STORE(1,1);
LOAD(2,1);
LIT(1);
SUB;
STORE(2,1);
$\mathrm{ct}(\texttt{F()}, \mathrm{st}', a'', 2)$
a_3 : RET;
$\mathrm{ct}(C, \mathrm{st}', a_0, 1)$
a_2 : RET;

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$$\mathrm{st}' = [\mathtt{x} \mapsto (\mathtt{var}, 0, 1),\ \mathtt{y} \mapsto (\mathtt{var}, 1, 1),\ \mathtt{F} \mapsto (\mathtt{proc}, a_1, 1, 0)]$$

Intermediate code:

1 : CALL(a_0,0,1);
2 : JMP(0);
a_1 : LOAD(2,1);
LIT(1);
GT;
a_4 : JFALSE(a_3);
LOAD(1,1);
LOAD(2,1);
MULT;
STORE(1,1);
LOAD(2,1);
LIT(1);
SUB;
STORE(2,1);
CALL(a_1,1,0);
a_3 : RET;
$\mathrm{ct}(C, \mathrm{st}', a_0, 1)$
a_2 : RET;

Example: Factorial Function I

Example 16.13 (Factorial function; cf. Example 15.3)

Source code:

```
in/out x;
  var y;
  proc F;
    if x > 1 then
      y := y * x;
      x := x - 1;
      F()
  y := 1;
  F();
  x := y.
```

$$\mathrm{st}' = [\mathtt{x} \mapsto (\mathtt{var}, 0, 1),\ \mathtt{y} \mapsto (\mathtt{var}, 1, 1),\ \mathtt{F} \mapsto (\mathtt{proc}, a_1, 1, 0)]$$

Intermediate code:

```
 1 : CALL(a0,0,1);
 2 : JMP(0);
a1 : LOAD(2,1);
     LIT(1);
     GT;
a4 : JFALSE(a3);
     LOAD(1,1);
     LOAD(2,1);
     MULT;
     STORE(1,1);
     LOAD(2,1);
     LIT(1);
     SUB;
     STORE(2,1);
     CALL(a1,1,0);
a3 : RET;
a0 : LIT(1);
     STORE(0,1);
     CALL(a1,0,0);
     LOAD(0,1);
     STORE(1,1);
a2 : RET;
```

Example: Factorial Function II

Example 16.13 (Factorial function; continued)

Code with symbolic addresses:

```
 1 : CALL(a0,0,1);
 2 : JMP(0);
a1 : LOAD(2,1);
     LIT(1);
     GT;
a4 : JFALSE(a3);
     LOAD(1,1);
     LOAD(2,1);
     MULT;
     STORE(1,1);
     LOAD(2,1);
     LIT(1);
     SUB;
     STORE(2,1);
     CALL(a1,1,0);
a3 : RET;
a0 : LIT(1);
     STORE(0,1);
     CALL(a1,0,0);
     LOAD(0,1);
     STORE(1,1);
a2 : RET;
```

Linearized ($a_0 = 17, a_1 = 3, a_2 = 22, a_3 = 16, a_4 = 6$):

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	$\underline{0:0:0:2}$

Example 16.13 (Factorial function; continued)

Computation for $\texttt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	$\underline{0:0:0:2}$
17	ε	$\underline{4:3:2:0}:\underline{0:0:0:2}$
18	1	$\underline{4:3:2:0}:\underline{0:0:0:2}$
19	ε	$\underline{4:3:2:1}:\underline{0:0:0:2}$

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	DS	PS
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	DS	PS
1	ε	$\underline{0:0:0:2}$
17	ε	$\underline{4:3:2:0}:\underline{0:0:0:2}$
18	1	$\underline{4:3:2:0}:\underline{0:0:0:2}$
19	ε	$\underline{4:3:2:1}:\underline{0:0:0:2}$
3	ε	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
4	2	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
5	$2:1$	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	$\underline{0:0:0:2}$
17	ε	$\underline{4:3:2:0}:\underline{0:0:0:2}$
18	1	$\underline{4:3:2:0}:\underline{0:0:0:2}$
19	ε	$\underline{4:3:2:1}:\underline{0:0:0:2}$
3	ε	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
4	2	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
5	2 : 1	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
6	1	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
1 : CALL(17,0,1);
2 : JMP(0);
3 : LOAD(2,1);
4 : LIT(1);
5 : GT;
6 : JFALSE(16);
7 : LOAD(1,1);
8 : LOAD(2,1);
9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
1 : CALL(17,0,1);
2 : JMP(0);
3 : LOAD(2,1);
4 : LIT(1);
5 : GT;
6 : JFALSE(16);
7 : LOAD(1,1);
8 : LOAD(2,1);
9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	DS	PS
1	ε	$0:0:0:2$
17	ε	$4:3:2:0:0:0:0:2$
18	1	$4:3:2:0:0:0:0:2$
19	ε	$4:3:2:1:0:0:0:2$
3	ε	$3:2:20:4:3:2:1:0:0:0:2$
4	2	$3:2:20:4:3:2:1:0:0:0:2$
5	$2:1$	$3:2:20:4:3:2:1:0:0:0:2$
6	1	$3:2:20:4:3:2:1:0:0:0:2$
7	ε	$3:2:20:4:3:2:1:0:0:0:2$
8	1	$3:2:20:4:3:2:1:0:0:0:2$
9	$1:2$	$3:2:20:4:3:2:1:0:0:0:2$
10	2	$3:2:20:4:3:2:1:0:0:0:2$

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	DS	PS
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for x = 2:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	DS	PS
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	DS	PS
1	ε	$\underline{0:0:0:2}$
17	ε	$\underline{4:3:2:0}:\underline{0:0:0:2}$
18	1	$\underline{4:3:2:0}:\underline{0:0:0:2}$
19	ε	$\underline{4:3:2:1}:\underline{0:0:0:2}$
3	ε	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
4	2	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
5	2 : 1	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
6	1	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
7	ε	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
8	1	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
9	1 : 2	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
10	2	$\underline{3:2:20}:\underline{4:3:2:1}:\underline{0:0:0:2}$
11	ε	$\underline{3:2:20}:\underline{4:3:2:2}:\underline{0:0:0:2}$
12	2	$\underline{3:2:20}:\underline{4:3:2:2}:\underline{0:0:0:2}$
13	2 : 1	$\underline{3:2:20}:\underline{4:3:2:2}:\underline{0:0:0:2}$
14	1	$\underline{3:2:20}:\underline{4:3:2:2}:\underline{0:0:0:2}$

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	DS	PS
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2

Example: Factorial Function III

Example 16.13 (Factorial function; continued)

Computation for $\mathtt{x} = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
2	ε	0 : 0 : 0 : 2

Example 16.13 (Factorial function; continued)

Computation for $x = 2$:

```
 1 : CALL(17,0,1);
 2 : JMP(0);
 3 : LOAD(2,1);
 4 : LIT(1);
 5 : GT;
 6 : JFALSE(16);
 7 : LOAD(1,1);
 8 : LOAD(2,1);
 9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

PC	*DS*	*PS*
1	ε	0 : 0 : 0 : 2
17	ε	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
19	ε	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
3	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7	ε	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
3	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
16	ε	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22	ε	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
2	ε	0 : 0 : 0 : 2
0	ε	0 : 0 : 0 : 2

Outline

Correctness of the Translation

Theorem 16.14 (Correctness of translation)

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

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

Correctness of the Translation

Theorem 16.14 (Correctness of translation)

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

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

Proof.

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