Blocks and Procedures

§1. Bip

§2. The enviroment-store model

§3. Arithmetic and Boolean expressions

§4. Declarations

§5. Statements

§6. Scope rules

§1. Bip

An extension of Bims with blocks and procedures.

- The syntactic categories of Bims are also found in Bip
- There are three additional ones

Pnames	— procedure names
DecV	— variable declarations
DecP	— procedure declarations

Bip:

$$S ::= x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S \mid \text{begin } D_V\ D_P\ S \text{ end} \mid \text{call } p$$

$$D_V ::= \text{var } x := a ; D_V \mid \varepsilon$$

$$D_P ::= \text{proc } p \text{ is } S ; D_P \mid \varepsilon$$

• A <u>block</u> is a statement with - declarations of local variables
- declarations of local procedures
- and a body S.

begin $D_v D_p$ S end.

- the variables and the procedures declared in a block should be available only within the block itself

• A procedure call invokes a procedure name.

call p

- the meaning of a procedure call depends on the scope rules, i.e, which variables and procedures are known during the execution of the procedure

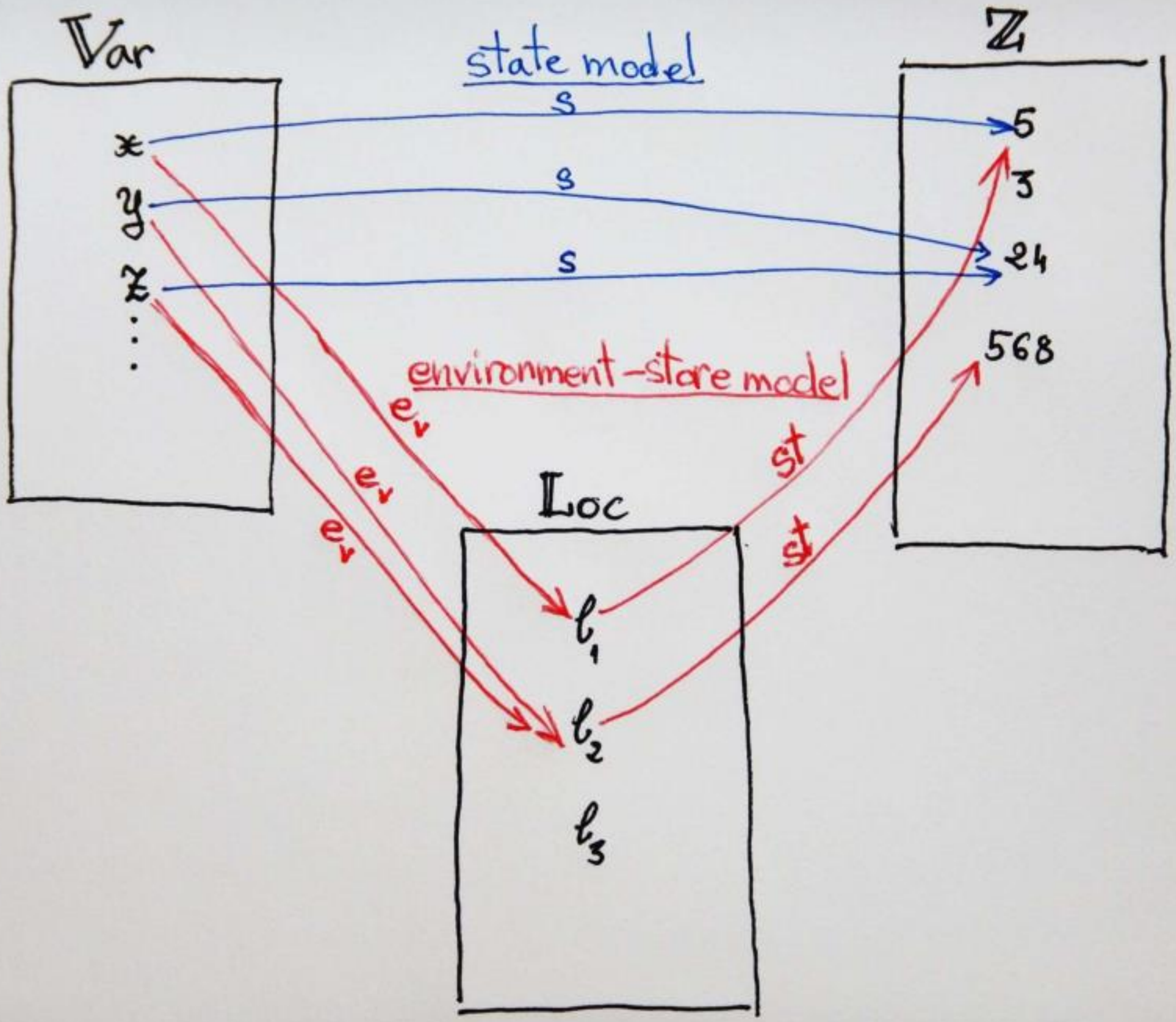
§2. The environment-store model

Previously we presented the <u>state model</u>, which is a simple model of variable bindings.
This model does not describe that variables are actually bound to storage cells in the computer memory.

$$s : \mathbb{V}ar \longrightarrow \mathbb{Z}$$

We introduce the <u>environment-store model</u> that describes how variables are actually bound during a program execution:

- each variable is bound to a storage cell
- the content of the storage cell is the value of the variable
- a program state describes • to which storage locations variables are bound
  • which values are found in the individual cells


Var
state model
s
s
s
x
y
z
5
3
24
568
environment-store model
e_v
e_v
e_v
st
st
Loc
l_1
l_2
l_3
ℤ

- The <u>variable environment</u> is a function that for each variable tells us to which storage location it is bound. Hence, it is a symbol table

- The <u>store</u> is a function that for each storage location tells us which value is found at the location. It corresponds to a complete description of the contents of the memory.

- The store cells are called <u>locations</u>.

$$\mathbb{L}oc \ni \ell$$

for simplicity we assume that $\mathbb{L}oc = \mathbb{N}$.

- We also need, for the robustness of the model, a way to reffer to the new locations to be alocated to new variables
  - we introduce a special pointer "next"

Definition:

The set of variable environments is the set of partial functions

$$\mathbb{E}\mathrm{nv}_V = \mathbb{V}\mathrm{ar} \cup \{\mathrm{next}\} \longrightarrow \mathbb{L}\mathrm{oc}$$

We use $e_V \in \mathbb{E}\mathrm{nv}_V$ to range over the set $\mathbb{E}\mathrm{nv}_V$

Question: Why should variable environments be partial functions?

We assume the existence of a function

$$\mathrm{new} : \mathbb{L}\mathrm{oc} \rightarrow \mathbb{L}\mathrm{oc}$$

that returns to each location its successor.

Example:

$$\mathrm{new} : \mathbb{N} \longrightarrow \mathbb{N},$$

$$\mathrm{new}\, l = l + 1$$

Definition:

The set of stores is the set of partial functions from locations to values.

$$\mathbb{S}\mathrm{to} = \mathbb{L}\mathrm{oc} \rightharpoonup \mathbb{Z}$$

We use $st \in \mathbb{S}\mathrm{to}$ to range over stores.

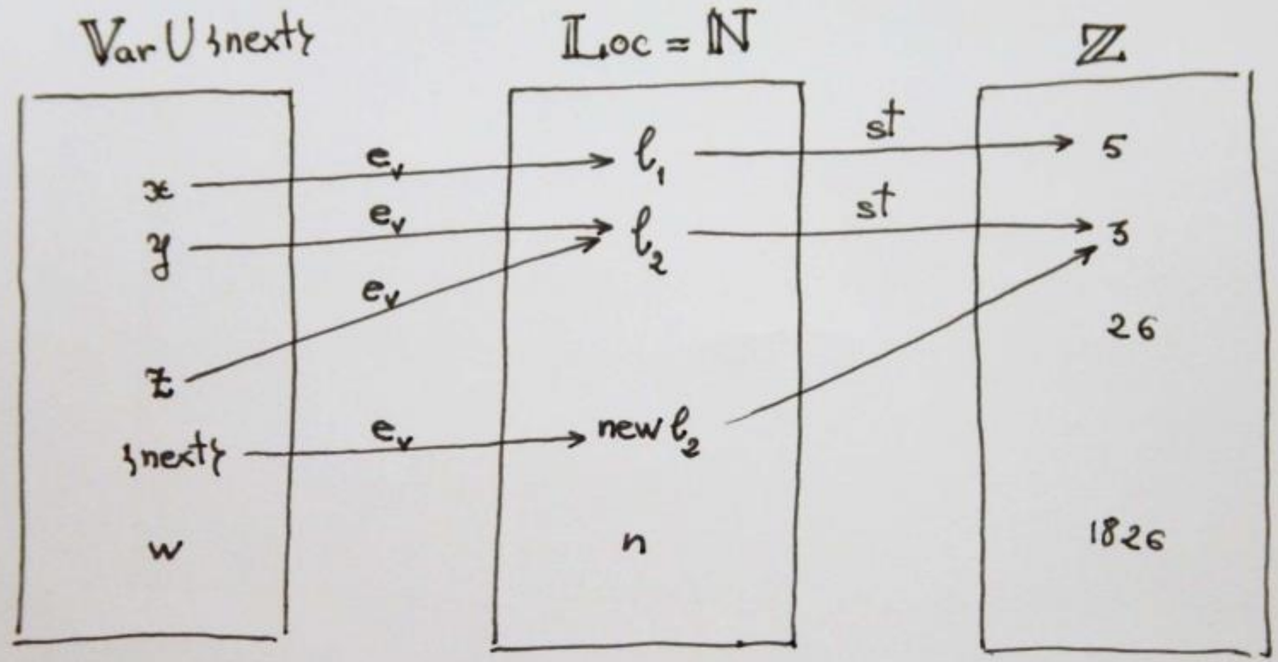
To update $e_v \in \mathbb{E}\mathrm{nv}$: for $x \in \mathbb{V}\mathrm{ar}$, $\ell \in \mathbb{N}$,

$$e_v[x \mapsto \ell](y) = \begin{cases} e_v(y), & y \neq x \\ \ell, & y = x \end{cases}$$

To update $st \in \mathbb{S}\mathrm{to}$: for $\ell \in \mathbb{N}$, $v \in \mathbb{Z}$

$$st[\ell \mapsto v](\ell') = \begin{cases} st(\ell'), & \ell \neq \ell' \\ v, & \ell = \ell' \end{cases}$$

The environment-store model



Observe that y and z are both bound to the location ℓ_2. So, if one of them is updated (i.e., st is updated), so is the other one.

<u>Question</u>: Describe a state of the environment-store model.

§3. Arithmetic and Boolean expressions

The semantics of statements and declarations both depend on the semantics of arithmetic expressions.

Arithmetic expressions may contain variables.

Since we have a new model of variable bindings, we need to redefine our semantics of arithmetic and Boolean expressions.

$$e_v, st \vdash a \rightarrow_A v$$

Bsp. $S ::= x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S \mid$
$\text{begin } D_V\, D_P\, S \text{ end} \mid \text{call } p$

$D_V ::= \text{var } x := a \,;\, D_V \mid \varepsilon$

$D_P ::= \text{proc } p \text{ is } S \,;\, D_P \mid \varepsilon$

BS-semantics for $\mathbb{A}exp$

[PLUS-BIP] $$\frac{e_v, st \vdash a_1 \longrightarrow_A v_1 \qquad e_v, st \vdash a_2 \longrightarrow_A v_2}{e_v, st \vdash a_1 + a_2 \longrightarrow_A v}, \; v = v_1 + v_2$$

[MINUS-BIP] $$\frac{e_v, st \vdash a_1 \longrightarrow_A v_1 \qquad e_v, st \vdash a_2 \longrightarrow_A v_2}{e_v, st \vdash a_1 - a_2 \longrightarrow_A v}, \; v = v_1 - v_2$$

[MULT-BIP] $$\frac{e_v, st \vdash a_1 \longrightarrow_A v_1 \qquad e_v, st \vdash a_2 \longrightarrow_A v_2}{e_v, st \vdash a_1 * a_2 \longrightarrow_A v}, \; v = v_1 \cdot v_2$$

[PARENT-BIP] $$\frac{e_v, st \vdash a \longrightarrow_A v}{e_v, st \vdash (a) \longrightarrow_A v}$$

[NUM-BIP] $$e_v, st \vdash n \longrightarrow_A v \;, \; \mathcal{N}[\![n]\!] = v$$

[VAR-BIP] $$e_v, st \vdash x \longrightarrow_A v \;, \; e_v(x) = \ell \text{ and } st(\ell) = v$$

Notice that we shall also need a new semantics of Boolean expressions
Why?

<u>Exercise</u>: Give a new big-step semantics of Boolean expressions under the environment-store model that makes use of the new semantics of Aexp.

§4.1. Declaring variables

- Any non-empty variable declaration will modify the variable environment since new variables will be bound to new locations.
- A variable declaration will also modify the store, since the new locations will be initialized to contain the initial values of the new variables.
- The transition relation describing variable declarations defines a big-step semantics, since the allocation of new address space for newly declared variables is an indiviisible operation

$$\langle D_V, e_V, st\rangle \longrightarrow_{DV} \langle e'_V, st'\rangle$$

BS-Semantics of variable declarations

$$[\text{VAR-DECL}] \quad \frac{\langle D_V, e_V'', st[\ell \mapsto v] \rangle \longrightarrow_{DV} \langle e_V', st' \rangle}{\langle \text{var } x := a; D_V, e_V, st \rangle \longrightarrow_{DV} \langle e_V', st' \rangle}$$

where $e_V, st \vdash a \longrightarrow_A v$

$\ell = e_V(\text{next})$

$e_V'' = e_V[x \mapsto \ell][\text{next} \mapsto \text{new}\,\ell]$

$$[\text{Empty-VAR}] \quad \langle \varepsilon, e_V, st \rangle \longrightarrow_{DV} \langle e_V, st \rangle$$

$$D_V ::= \text{var } x := a ; D_V \mid \varepsilon$$

§4.2. Procedure declarations

- Similarly to variable environments, we will have procedure environments
- A procedure environment holds information about the bindings of procedure names.

- As for variable declarations, the procedure declarations are indivisible operations, so we will have again a BS-semantics

$$e_v \vdash \langle D_P, e_P \rangle \longrightarrow_{DP} e'_P$$

where e_P denotes procedure environments.

For technical reasons we postpone for a while this definition

§5. Statements

- The effect of a statement is that the store may change, since a statement may modify the values of variables involved through assignments.

- A statement should not modify the variable environment.

- We define a BS-semantics for statements (except procedure calls)
  - transitions $e_v, e_p \vdash \langle S, st \rangle \rightarrow st'$
  - transition system

$$((\mathrm{Stm} \times \mathrm{Sto}) \cup \mathrm{Sto}, \rightarrow, \mathrm{Sto})$$

BS-semantics for statements in $\mathbb{B}ip$

[ASSGN] $e_v, e_p \vdash \langle x := a, st \rangle \rightarrow st[\ell \mapsto v]$

where $e_v, st \vdash a \rightarrow_A v$ and $e_v(x) = \ell$

[SKIP] $e_v, e_p \vdash \langle \text{skip}, st \rangle \rightarrow st$

[COMP] $$\frac{e_v, e_p \vdash \langle S_1, st \rangle \rightarrow st'' \quad e_v, e_p \vdash \langle S_2, st'' \rangle \rightarrow st'}{e_v, e_p \vdash \langle S_1; S_2, st \rangle \rightarrow st'}$$

[IF-T] $$\frac{e_v, e_p \vdash \langle S_1, st \rangle \rightarrow st'}{e_v, e_p \vdash \langle \text{if } b \text{ then } S_1 \text{ else } S_2, st \rangle \rightarrow st'} \quad st \vdash b \rightarrow_B \top$$

[IF-$\perp$] $$\frac{e_v, e_p \vdash \langle S_2, st \rangle \rightarrow st'}{e_v, e_p \vdash \langle \text{if } b \text{ then } S_1 \text{ else } S_2, st \rangle \rightarrow st'} \quad st \vdash b \rightarrow_B \perp$$

BS-semantics for statements in Bip

[WHILE-T]

$$\frac{e_v, e_p \vdash \langle S, st\rangle \rightarrow st'' \qquad e_v, e_p \vdash \langle \text{while}\ b\ \text{do}\ S, st''\rangle \rightarrow st'}{e_v, e_p \vdash \langle \text{while}\ b\ \text{do}\ S, st\rangle \rightarrow st'}$$

if $e_v, st \vdash b \rightarrow_B \bot$

[WHILE-⊥]

$$e_v, e_p \vdash \langle \text{while}\ b\ \text{do}\ S, st\rangle \rightarrow st$$

if $e_v, st \vdash b \rightarrow_B \bot$

[BLOCK]

$$\frac{\begin{array}{c}\langle D_v, e_v, st\rangle \rightarrow_{DV} \langle e_v', st''\rangle \\ e_v' \vdash \langle D_p, e_p\rangle \rightarrow_{DP} e_p' \\ e_v', e_p' \vdash \langle S, st''\rangle \rightarrow st'\end{array}}{e_v, e_p \vdash \langle \text{begin}\ D_v\ D_p\ S\ \text{end}, st\rangle \rightarrow st'}$$

§6. Scope Rules

The role of the scope rules is to tell us which bindings are in effect during the execution of a procedure call.

Dynamic scope rules - employ the bindings known when the procedure is called.

Static scope rules - employ the bindings known when the procedure was declared.

```
Program
begin - variable declaration list 1 (x, y, z)
      - procedure declaration list 1 (p, q, r)
      begin - variable declaration list 2 (x, y, z)
            - procedure declaration list 2 (p, q)
            call r  (implies p, q, x, y, z)
      end
end
```

Example:

```
begin
    var x:= 0;
    var y:=42;

    proc p is  x:=x+3;
    proc q is  call p;

    begin
        var x:=9;
        proc p is  x:=x+1;
        call q;
        y:=x
    end
end.
```

Question 1: Assume a fully dynamic scope rules for both variables and procedures: during the execution of procedure q, we know the variables and the procedures known when q was called.

What is the value of y after the previous statement is executed?

Question 2: Assume dynamic scope rules for variables and static scope rules for procedures: during the execution of q we know the variables that exist when q was called and the procedures known when q was declared

What is the value of y after the statement is executed?

Question 3 : Assume dynamic scope rules for procedures and static scope rules for variables: during the execution of q we know the variables that exist when q was declared and the procedures known when q was called.

What is the value of y after the statement is executed?

Question 4 : Assume fully static scope rules for both variables and procedures: during the execution of q we know only the variables and procedures that were known when q was declaired.

What is the value of y after the statement is executed

§6.1. Fully dynamic scope rules

- only the bindings known at the time when the procedure is called are considered.
- when the procedure is declaired, we need to remember only the body of the procedure

$$\mathbf{Env}_P = \mathbf{Pnames} \longrightarrow \mathbf{Stm}$$

Transition rules for procedure declarations

[PROC] $$\frac{e_v \vdash \langle D_P, e_P[p \mapsto S] \rangle \longrightarrow_{DP} e'_P}{e_v \vdash \langle \text{proc } p \text{ is } S;\ D_P, e_P \rangle \longrightarrow_{DP} e'_P}$$

[PROC-EMPTY] $$e_v \vdash \langle \varepsilon, e_P \rangle \longrightarrow_{DP} e_P$$

Transition rule for procedure call

[CALL-DYN-DYN] $$\frac{e_v, e_p \vdash \langle S, st\rangle \longrightarrow st'}{e_v, e_p \vdash \langle \mathrm{call}\ p, st\rangle \longrightarrow st'}, \ e_p(p) = S$$

Exercise: Prove the value of y in the previous example by applying this BS-semantics.

§6.2. Mixed scope rules

We consider the case where we have dynamic scope rules for variables and static scope rules for procedures.

$$\mathbf{Env}_P = \mathbf{Pnames} \longrightarrow \mathbf{Stm} \times \mathbf{Env}_P$$

Transition rules for procedure declarations

$$[\text{PROC}] \quad \frac{e_V \vdash \langle D_P, e_P[p \mapsto \langle S, e_P \rangle] \rangle \longrightarrow_{DP} e'_P}{e_V \vdash \langle \text{proc } p \text{ is } S; D_P, e_P \rangle \longrightarrow_{DP} e'_P}$$

$$[\text{PROC-EMPTY}] \quad e_V \vdash \langle \varepsilon, e_P \rangle \longrightarrow_{DP} e_P$$

Transition rule for procedure call

$$[\text{CALL-DYN-STAT}] \quad \frac{e_v, e'_p \vdash \langle S, st \rangle \longrightarrow st'}{e_v, e_p \vdash \langle \text{call } p, st \rangle \longrightarrow st'} \quad e_p(p) = \langle S, e'_p \rangle$$

Exercise: Prove the value of y in the previous example by applying this BS-semantics.

Exercise: Give a BS-semantics for a version of $\mathbb{B}$ip with static scope rules for variables and dynamic scope rules for procedures.

§6.3. Fully static scope rules

- The execution of a procedure call can use only the variable bindings and the procedure bindings known when the procedure was declared.
  - these bindings must be remembered by the procedure environment

$$\mathbb{E}\mathrm{nv}_P = \mathbb{P}\mathrm{names} \longrightarrow \mathbb{S}\mathrm{tm} \times \mathbb{E}\mathrm{nv}_V \times \mathbb{E}\mathrm{nv}_P$$

Transition rules for procedure declarations

[PROC] $$\frac{e_V \vdash \langle D_P, e_P[p \mapsto \langle S, e_V, e_P\rangle]\rangle \longrightarrow_{DP} e'_P}{e_V \vdash \langle \mathrm{proc}\ p\ \mathrm{is}\ S; D_P, e_P\rangle \longrightarrow_{DP} e'_P}$$

[PROC-EMPTY] $$e_V \vdash \langle \varepsilon, e_P\rangle \longrightarrow_{DP} e_P$$

Transition rule for procedure call

$$[\text{CALL-STAT-STAT}] \quad \frac{e'_v[\text{next} \mapsto \ell],\ e'_p \vdash \langle S, st \rangle \longrightarrow st'}{e_v,\ e_p \vdash \langle \text{call}\, p, st \rangle \longrightarrow st'}$$

where $e_p(p) = \langle S, e'_v, e'_p \rangle$

and $\ell = e_v(\text{next})$

Exercise: Prove the value of y in the previous example by applying this BS-semantics.

2