Parameters

1. The language Bump
2. Call-by-reference
3. Recursive and non-recursive procedure calls
4. Call-by-value
5. Call-by-name

§1. Bump

We focus in what follows on parameter passing: $\begin{cases} \text{call-by-reference} \\ \text{call-by-value} \\ \text{call-by-name} \end{cases}$

Syntax of Bump

$$S ::= x:=a \mid \text{skip} \mid S_1;S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid \text{while } b \text{ do } S \mid$$
$$\text{begin } D_V\ D_P\ S \text{ end} \mid \text{call } p(y)$$
$$D_V ::= \text{var } x:=a\ ;\ D_V \mid \varepsilon$$
$$D_P ::= \text{proc } p(\text{var } x) \text{ is } S\ ;\ D_P \mid \varepsilon$$

We consider procedures with only one parameter.

In a procedure declaration proc p(var x) is S x is the formal parameter

In a procedure call call p(y) y is the actual parameter.

A parameter mechanism describes the relation between the formal and the actual parameters.

In what follows we assume the environment-store model.

$$\mathbb{E}\mathrm{nv}_V = \mathbb{V}\mathrm{ar} \cup \{\mathrm{next}\} \to \mathbb{L}\mathrm{oc}, \qquad \mathbb{S}\mathrm{to} = \mathbb{L}\mathrm{oc} \to \mathbb{Z}$$

In the context of parametrized procedures, we have

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

The role of $\mathbb{V}\mathrm{ar}$ is to designate the formal parameter.

Rules for procedure declaration assuming static scope rules

$$[\mathrm{PROC}] \quad \frac{e_V \vdash \langle D_P, e_P[p \mapsto \langle S, x, e_V, e_P\rangle]\rangle \to_{DP} e'_P}{e_V \vdash \langle \texttt{proc } p(\texttt{var } x) \texttt{ is } S ;\ D_P, e_P\rangle \to_{DP} e'_P}$$

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

Note that the definition of $\mathbb{E}\mathrm{nv}_P$ is completely independent of our choice of the parameter mechanism.
It only depends our choice of scope rules and that the name of the formal parameter must be remembered.

In the previous slide we defined $\longrightarrow_{DP}$ for the fully static scope rules, as it is formally the most complex one. Similar definitions can however be given for each type of scope rules

Fully dynamic scope rules: $\mathbb{E}\mathrm{nv}_P = \mathbb{P}\mathrm{names} \longrightarrow \mathbb{S}\mathrm{tm} \times \mathbb{V}\mathrm{ar}$

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

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

Note that the definition of $\mathbb{E}\mathrm{nv}_p$ is completely independent of our choice of the parameter mechanism.

It only depends our choice of scope rules and that the name of the formal parameter must be remembered.

In the previous slide we defined $\longrightarrow_{DP}$ for the fully static scope rules, as it is formally the most complex one. Similar definitions can however be given for each type of scope rules

Mixed scope rules: $\mathbb{E}\mathrm{nv}_p = \mathbb{P}\mathrm{names} \longrightarrow \mathbb{S}\mathrm{tm} \times \mathbb{V}\mathrm{ar} \times \mathbb{E}\mathrm{nv}_p$

$$[\mathrm{PROC}] \quad \frac{e_v \vdash \langle D_p, e_p[p \mapsto \langle S, x, e_p \rangle] \rangle \longrightarrow_{DP} e'_p}{e_v \vdash \langle \mathrm{proc}\ p(x)\ \mathrm{is}\ S; D_p, e_p \rangle \longrightarrow_{DP} e'_p}$$

$$[\mathrm{PROC\text{-}EMPTY}] \quad e_v \vdash \langle \varepsilon, e_p \rangle \longrightarrow_{DP} e_p$$

While the BS-semantics of the declaring procedures does not depend on the choice of the parameter mechanism, the semantics of statements, i.e., of procedure calls does depend on the choice of the parameter mechanism.

We will have a different type of procedure call rule for

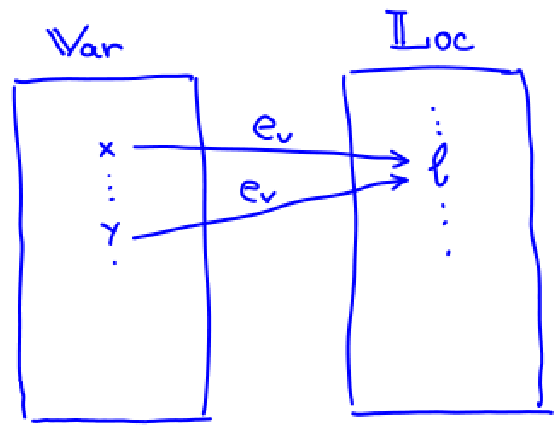
- call-by-reference
- call-by-value
- call-by-name.

§2. Call-by-reference

The call-by-reference parameter mechanism
is present in: Pascal, C, C++, C#
not present in Java (uses call-by-value)

The basic idea:
the formal parameter is a reference to the address of the actual parameter. Hence, the actual parameter must be a variable.

```
...
proc p(var x) is x:=x+1
...
call p(y)
```



§2. Call-by-reference

The call-by-reference parameter mechanism
is present in: Pascal, C, C++, C#
not present in Java (uses call-by-value)

The basic idea:
the formal parameter is a reference to the address of the actual parameter. Hence, the actual parameter must be a variable.

The call-by-reference parameter mechanism associates the formal parameter with the location of the actual parameter

The BS-semantics of statements for $\mathbb{B}$ump is different from the one of $\mathbb{B}$ip only for the procedure-call rule.

However, remember that the semantics of the procedure declarations is also redefined for $\mathbb{B}$ump and this will, of course, influence further the outcome of the semantics.

Transition rule for calling a call-by-reference procedure

$$[\text{CALL-R}] \quad \frac{e'_v[x \mapsto \ell][next \mapsto \ell'], e'_p \vdash \langle S, st\rangle \to st'}{e_v, e_p \vdash \langle \text{call } p(y), st\rangle \to st'}$$

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

$\ell = e_v(y)$

$\ell' = e_v(next)$

§3. Recursive and Non-recursive procedure calls

Notice that the rule [CALL-R] does not allow p to call itself recursively.

Why not?

Because the procedure body S is being executed in e'_p, which knows only the procedures that were known immediately prior to the declaration of p.

Example:

```
begin
 var y:=0;
 var x:=1
 proc f(var x) is
     begin
        var z:= x-1;
        y:= y*x;
        if x>1 then call f(z)
        else skip
     end
   y:=4;
   call f(y);
   z:=y
 end
```

! Remember that we work with fully static scope rules !

However, we can solve this problem:

we add a binding for p in e'_p such that p is associated to the information needed to call the correct version of p

Transition rule for procedure calls that allow recursive calls

$$[\text{CALL-R-Rec}]\ \frac{e'_v[x \mapsto \ell][next \mapsto \ell'],\ e''_p \vdash \langle S, st\rangle \longrightarrow st'}{e_v, e_p \vdash \langle \text{call } p(y), st\rangle \longrightarrow st'}$$

where
$$e_p(p) = \langle S, x, e'_v, e'_p\rangle$$
$$e_v(y) = \ell$$
$$e_v(next) = \ell'$$
$$e''_p = e'_p[p \mapsto \langle S, x, e'_v, e'_p\rangle]$$

Notice that all these modifications are required by the fully static scope rules. If we had chosen dynamic scope rules, no modification of the [CALL-R] rule would be necessary.

§4. Call-by-value

Syntax of **Bump**:

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

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

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

The actual parameter is now an arithmetic expression a.
The formal parameter is a local variable in the body of the procedure.
In a procedure call call $p(a)$ the initial value of the formal parameter (variable) is the value of the actual parameter a.

Example:

. . .
proc $p(x)$ is $x:=x+1$
. . .
call $p(y+2)$

1. we find the value of $y+2$;
2. this value becomes the initial value of x
3. the body of the procedure is executed.

The semantics of the procedure declaration is similar to the one for call-by-reference

Fully static scope rules: $\mathbb{E}\mathrm{nv}_P = \mathbb{P}\mathrm{names} \to \mathbb{S}\mathrm{tm} \times \mathbb{V}\mathrm{ar} \times \mathbb{E}\mathrm{nv}_V \times \mathbb{E}\mathrm{nv}_P$

$$[\mathrm{PROC}] \quad \frac{e_V \vdash \langle D_P, e_P[p \mapsto \langle S, x, e_V, e_P\rangle]\rangle \to_{DP} e'_P}{e_V \vdash \langle \mathrm{proc}\ p(\mathrm{var}\ x)\ \mathrm{is}\ S;\ D_P, e_P\rangle \to_{DP} e'_P}$$

Fully dynamic scope rules: $\mathbb{E}\mathrm{nv}_P = \mathbb{P}\mathrm{names} \to \mathbb{S}\mathrm{tm} \times \mathbb{V}\mathrm{ar}$

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

Mixed scope rules: $\mathbb{E}\mathrm{nv}_P = \mathbb{P}\mathrm{names} \to \mathbb{S}\mathrm{tm} \times \mathbb{V}\mathrm{ar} \times \mathbb{E}\mathrm{nv}_P$

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

Transition rules for procedure calls using call-by-value for fully static scope rules

$$[\text{CALL-V}]\ \frac{e_v'[x \mapsto \ell][\text{next} \mapsto \text{new}\,\ell],\, e_p' \vdash \langle S, st[\ell \mapsto v]\rangle \to st'}{e_v, e_p \vdash \langle \text{call } p(a), st\rangle \to st'}$$

where $e_p(p) = \langle S, x, e_v', e_p'\rangle$

$e_v, st \vdash a \to_A v$

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

[CALL-V-Rec]

$$\frac{e_v'[x \mapsto \ell][\text{next} \mapsto \text{new}\,\ell],\, e_p'[p \mapsto \langle S, x, e_v', e_p'\rangle] \vdash \langle S, st[\ell \mapsto v]\rangle \to st'}{e_v, e_p \vdash \langle \text{call } p(a), st\rangle \to st'}$$

where $e_p(p) = \langle S, x, e_v', e_p'\rangle$

$e_v, st \vdash a \to_A v$

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