

LOTOS COMPLET

LOTOS complet = LOTOS de base + types

Que peut-on faire avec les types?

On peut utiliser les termes d'un type abstrait

- dans les échanges entre processus via des ports : $g! \text{exp}$ $g? x : t$

- pour exprimer des conditions sur certains comportements ou certaines valeurs

$([\text{cond1}] \rightarrow B1 \ [] \ [\text{cond2}] \rightarrow \dots)$

$g? x:t[\text{cond}]$

- pour faire des choix généralisés

choice $x:t[\text{cond}] \dots x\dots$

NB : **choice** $g \text{ in } [g1, \dots, gn] \dots g\dots$

- pour passer des arguments à des processus paramétrés

- pour décrire les résultats d'un processus qui termine avec succès et les passer en séquence
... **exit** (*exp1*, ..., *expn*) ...>> **accept** *x1:t1*, ..., *xn:tn* in B

4.1 ECHANGES DE VALEURS ENTRE PROCESSUS

g!exp *exp* est un terme avec ou sans variable

g?x:t *x* est une variable, *t* est un type déclaré

exemple

a?x:nat ; b?y:nat; c!largest(x,y); stop

Un comportement :

a?x:nat ; b?y:nat; c!largest(x,y); stop

-**a<0>**->

b?y:nat; c!largest(0,y); stop

x liée à 0

-**b<3>**->

c!largest(0,3); stop

-**c<3>**->

stop

Soit v la valeur de exp

- $g!exp$ et $g?x:t$ se synchronisent pour faire $g\langle v \rangle$
- x a pour valeur v

Exemple de Spécification

```

specification Max3[in1, in2, in3, out] : noexit
type natural is
  sorts nat
  opns    zero : -> nat
           succ : nat -> nat
           largest : nat, nat -> nat
  eqns forall x, y :nat
  ofsort nat
  largest (zero, x) = x ;
  largest (x, y) = largest (y, x) ;
  largest (succ(x), succ(y)) = succ (largest(x, y)) ;
endtype
behaviour
  hide mid in
  Max2[in1, in2, mid] | [mid] | Max2[mid, in3, out]
  where
  process Max2[a, b, c] : noexit :=
    a?x:nat ; b?y:nat; c!largest(x,y); stop
    [ ]
    b?x:nat ; a?y:nat; c!largest(x,y); stop
  endproc
endspec

```

Format général d'une spécification

specification *ID* [*liste de ports*] (*liste de paramètres*): **fonctionnalité**

type ... is

sorts ...

opns ...

eqns forall ...

ofsort ...

 ...

endtype

behaviour

expression de comportement

where

type ... is

 ...

endtype

process *ID* [*liste de ports*] (*liste de paramètres*):

fonctionnalité :=

 ...

endproc

endspec

Conditions de synchronisation

P1 || P2 peut exécuter $g\langle v \rangle$ ssi
 parmi les actions exécutables de P1 et P2 on a

- $g!exp1$ et $g!exp2$ et $exp1 = exp2 = v$
- $g!exp1$ et $g?x:t$ et $exp1$ est de sorte t
- $g?x:t$ et $g?y:t$, v est alors une *valeur arbitraire* de type t ...

Cas des définitions conditionnelles de variables

$g?x:t[cond]$

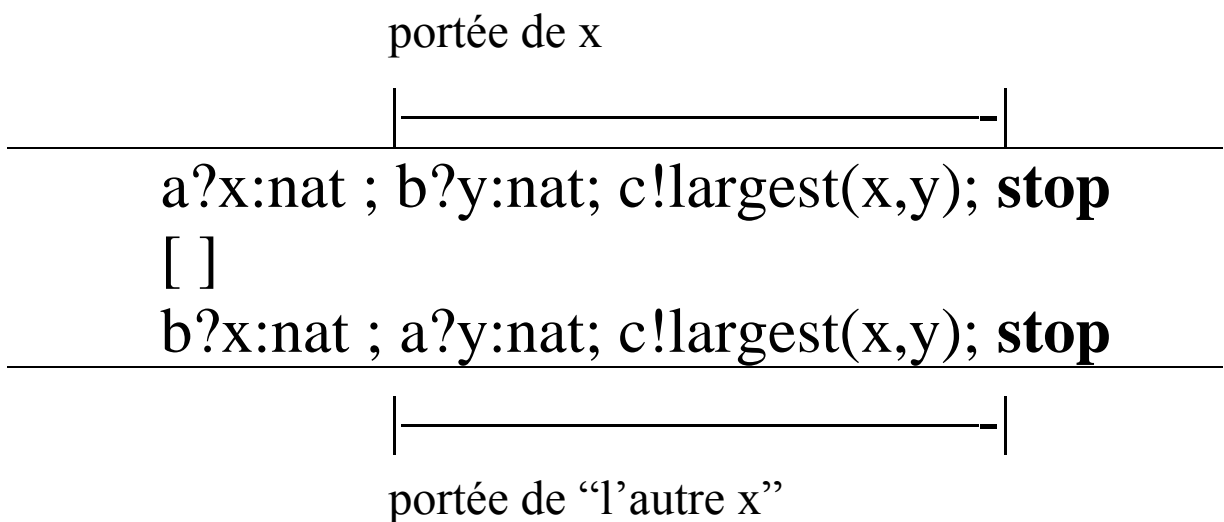
- $g!v$ est la même action que $g?x:t[x eq v]$
- $g!v$ et $g?x:t[cond]$ se synchronisent si v est de sorte t et satisfait $cond$
- $g?x:t[cond1]$ et $g?x:t[cond2]$ se synchronisent avec une valeur de sorte t qui satisfait $cond1$ et $cond2$

généralisation : échanges multiples

$g?x:t1[cond1]!exp1!exp2?y:t2[cond2]$

4.2 PORTEE DES VARIABLES

- $g?x:t$ est une déclaration de x dont la *portée* est la suite du processus



- déclarations imbriquées : comme d’habitude

$g?x:nat ; P[a] (x+1) \gg g!x ; \mathbf{exit}$

where

process $P[gg](h:nat) : \mathbf{exit} :=$

$gg?x:bool; ([x] \rightarrow gg!h; \mathbf{exit}$

$[]$

$[x \text{ eq } false] \rightarrow gg!-h; \mathbf{exit})$

endproc

- déclarations locales

(**let** $x:t = \dots$ **in** $\dots x \dots$)

4.3 – COMPOSITION SEQUENTIELLE

$B1 \gg B2$

$P\dots \gg Q\dots$

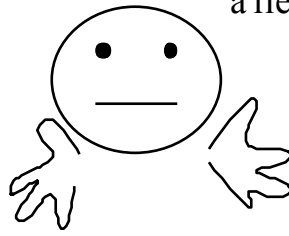
- P doit terminer avec succès pour que Q s'exécute (cf. LOTOS de base)
- Nouveau : P peut terminer avec succès *en passant des résultats à Q*

exemple

$P[a, b] : \mathbf{exit}(\mathbf{nat}) := a?x:\mathbf{nat} ; b?y:\mathbf{nat}; \mathbf{exit}$
 $(\mathbf{largest}(x,y))$

- La fonctionnalité de P est $\langle \mathbf{nat} \rangle$
- La fonctionnalité doit apparaître dans l'entête du processus

ça ne sert
à rien



(les outils ne font pas les vérifications)

exemples

```
process P[a] : exit(nat, bool) :=
  a?x:nat ?y:nat ;
  ( i; exit(x, true) [ ] i; exit(y, false))
endproc
```

```
process Q[a, b] : exit :=
  a?x:nat ; (b!"hello"; exit [ ] i; Q[a,b])
endproc
```

```
process R[a, b] : noexit :=
  a?x:nat ?y:bool ; (b!x; stop [ ] b!y; R[a, b])
endproc
```

Passages de résultats

$B1 \gg \mathbf{accept} \ x1:t1, \dots, xn:tn \ \mathbf{in} \ B2$

- B1 doit être de fonctionnalité $\langle t1, \dots, tn \rangle$
- les résultats de B1 sont nommés $x1, \dots, xn$ dans B2

Exemple :

$P[a] \gg \mathbf{accept} \ x:nat, y:bool \ \mathbf{in}$
 $(b!x; \mathbf{stop} [] b!y; \mathbf{stop})$

a?x:int; exit b!hello; exit	=> exit
a?x:int; exit b!hello; stop	=> noexit
exit(3) exit(5) <i>ne termine pas avec succès</i>	=> exit, ... <i>mais</i>

On doit connaître tous les processus pour
 écrire la terminaison de chaque processus ☹ :

... exit(3, any bool) ||| ... exit(any nat, true)

LES PROCESSUS PARAMETRES

permettent d'avoir de la mémoire et de la transmettre

exemple

process compare [in, out] (min, max :int) : **noexit**
:=

```

in?x:int;
([min < x < max] -> out!x;
    compare[in,out](min, max)
[] [x ≤ min] -> out!min;
    compare[in,out](x, max)
[] [x ≥ max] -> out!max;
    compare[in,out](min, x) )

```

endproc

instantiation \Leftrightarrow substitution

compare[un, deux] (x, 2*x)

est équivalent à

```

un?xx:int;    (*renommage de x *)
([x < xx < 2*x] -> deux!xx;
    compare[un, deux](x, 2*x)
[] (*etc*)

```

Autre exemple

facturation pour deux(!) abonnés

```

process factures [a, b, pr] (la, lb : liste_com) :
noexit :=
    ((a?n:num ?d:duree ;
        factures [a, b, pr] (add(la, n, d), lb)
    []
    b?n:num ?d:duree ;
        factures (la, add(lb, n, d))
    []
    [size(la) > 100] -> pr!la; factures(empty, lb)
    []
    [size(lb) > 100] -> pr!lb; factures(la, empty) )
endproc

```

Limitations désagréables

- pas de structures de données pour les ports
- pas de création dynamique de ports ou de processus
 - et de plus, pas de temporisations

EXERCICE 1

Comment généraliser la facturation à un ensemble quelconque d'abonnés, qui peut changer, en utilisant la paramétrisation ?

```

process Nfactures [in, out, pr] (T : Table) : noexit :=
  (* T contient la table des abonnés x list_comm *)
  (in ?a :abonné?n:num ?d:duree ;
    ( [size(search(a,T)) ≤ 100] ->
      Nfactures [in, out, pr] (change(T, a,
add(search(a,T), n, d))
    [ ]
    [size(search(a,T)) > 100] ->
      pr! add(search(a,T), n, d)));
    Nfactures [in, out, pr] (change(T, a,
empty))
endproc

```

COMMENT AJOUTER ET SUPPRIMER DES ABONNES ?

EXERCICE 2

Comment spécifier le système de guidage d'une automobile ?

- Le système de guidage d'une voiture reçoit périodiquement les coordonnées de celle-ci
- Quand il n'est pas activé, il se contente d'afficher ces coordonnées
- Le conducteur de la voiture peut activer le système de guidage en lui donnant une destination
 - Le système calcule alors le plus court chemin et indique au conducteur la direction à prendre
 - Si le conducteur suit le chemin indiqué, le système affiche les coordonnées et indique éventuellement un changement de direction
 - Sinon, le système affiche les coordonnées, recalcule le plus court chemin et indique éventuellement un changement de direction
 - Le processus se termine avec succès quand les coordonnées correspondent à la destination
 - À tout moment le conducteur peut désactiver le système ou changer de destination

Spécifier en LOTOS ce système de guidage

LES TYPES DE DONNEES :

- les coordonnées
- un graphe dont les sommets sont des coordonnées
- pour simplifier, les successeurs d'un sommets sont numérotés de gauche à droite, et ce numéro est une direction (on peut imaginer une conversion de ces numéro en "à gauche", "1^{ère} à droite", etc)

LES PORTS DE COMMUNICATION

- "gps" est le port où arrivent les coordonnées
- "écran" est le port où elles sont affichées, avec éventuellement la direction à prendre
- "HP" est le haut-parleur qui indique la direction à prendre
- Comment modéliser l'interaction avec le conducteur ? ports ou paramètres ?

UN PREMIER JET

```

process guidage_inactif
[gps, ecr] (carte:Graphe) : noexit :=
    gps?x:Nat?y:Nat ;
    ecr!x!y ;
    guidage_inactif [gps, ecr, HP]
(carte)
endproc

```

```

process guidage
[gps, ecr, HP, cond] (carte:Graphe) :
noexit :=
    guidage_inactif [gps, ecr]
(carte)
    [> activation [gps, ecr, HP,
cond] (carte)

```

where

```

    process activation [gps, ecr,
HP, cond] (carte:Graphe) : noexit :=
        cond?dx:Nat?dy:Nat ;
        guidage_actif [gps, ecr, HP,
cond] (carte, dx, dy)
    where ...
    endproc
endproc

```

```

process guidage_actif [gps, ecr,
HP, cond] (carte:Graphe, dx:Nat,
dy:Nat) : noexit :=
    ( gps?x:Nat?y:Nat ; ecr!x!y ;
      ([x≠dx or y≠dy] ->
        HP!no_suivant(x, y, carte, dx,
dy); guidage_actif [gps, ecr, HP,
cond] (carte, dx, dy) )
      []
      [x=dx and y=dy] ->
        HP!"vous etes arrive"; guidage
[gps, ecr, HP, cond] (carte) )
    [> activation [gps, ecr, HP,
cond] (carte)
    [> desactivation [gps, ecr, HP,
cond] (carte)
where
    process desactivation [gps,
ecr, HP, cond] (carte:Graphe) : noexit
:=
    cond!desactive ;
    guidage [gps, ecran, HP, cond]
(carte)
    endproc
endproc

```
