Introduction aux algèbres de processus et LOTOS

Christian Attiogbé
UFR sciences - Nantes
Dpt. Informatique, 1999/2000, maj : janvier 2004
Christian.attiogbe@lina.univ-nantes.fr
www.sciences.univ-nantes.fr/info/perso/permanents/attiogbe/

Slide 1

Introduction

- *Algèbres de processus* =
  une catégorie de formalismes qui permettent de *décrire* et d'*analyser* les comportements de *systèmes (processus) concurrents*.
- Historique
  - Théorie des automates,
  - Théorie des réseaux de PETRI,
  - Algèbres de processus :
    CSP (Communicating Sequential Processes) ; C.A.R. HOARE, 1978,
    `occam + Transputer`
    CCS (Calculus of Communicating Systems) ; R. MILNER, 1985
    ...

Slide 2

Caractéristiques principales des algèbres de processus

Slide 3

- Spécification et étude des systèmes concurrents (communication, synchronisation)
- Abstraction sur les comportements,
- Mode de synchronisation (synchrone, RdV, actions complémentaires), etc
- Mode de composition (parallèle, entrelacement, etc),
- Modèles sémantiques (opérationnelle, traces, bissimulation, *failure-divergence*).

Concepts fondamentaux

Slide 4

Alphabets

L'évolution d'un processus est décrite à l'aide des noms des actions qu'il entreprend : *alphabet*.

Expressions régulières

Combinaison de noms d'action (*alphabet*) à l'aide d'opérateurs prédéfinis. Automate à états.

Processus élémentaire

L'évolution séquentielle, exprimée à l'aide de combinaison d'actions et d'opérateurs (**séquence, choix, arrêt**) : on parle de *comportement* (***behaviour***).

Concepts fondamentaux (suite)

Processus prédéfinis

Slide 5

– Un procesus qui ne termine pas (***deadlock***) :
il ne peut plus effectuer aucune action de son alphabet.
En CSP c'est **STOP**, en CCS c'est **0**.

– Un processus qui termine mais qui ne peut effectuer aucune action. En CSP c'est `skip`.

⌟

Principaux opérateurs (illustration avec CSP)

Séquence (notée ;)

Une action suivie d'un comportement (le reste des actions).
Soit un alphabet $A = \{lire, ecrire, traiter, ouvrir\}$

```
lire;traiter;ecrire;STOP
```

Slide 6

⌟

Principaux opérateurs (illustration avec CSP)

Choix de comportements (noté [])

Le processus s'engage dans un comportement ou un autre :

```
lire;STOP [] ecrire;STOP
```

Slide 7

```
lire;STOP [] ecrire;(traiter;STOP [] ouvrir;STOP)
```

⌟

Interruption (notée [>)

Alphabet pour la suite : $\mathcal{A} = \{a, b, c, d, e, f, g\}$

Soit P un processus avec le *comportement principal* suivant nommé P1 :

```
a;(b;STOP [] c;STOP) [] d;STOP
```

Slide 8

On veut exprimer que P peut être interrompu à tout moment et adopter le comportement `e ;f ;g ;STOP`

On écrit :

```
P = P1 [> e;f;g;STOP
```

Exercice Donner une représentation graphique de P

⌟

Communication

Communication : seul moyen de **synchroniser** et d'**échanger des valeurs** entre les processus.
La communication est effectuée via des **actions** ou des **canaux**.

Utilisation de **canaux** de communication (CSP) ou
de **ports** de communication (CCS).

Slide 9

En CSP
Opérateur d'émission sur un canal : **!**
Opérateur de réception sur un canal : **?**

Exemples

```
ca?va:int ; cb?vb:int ; cc!(va +  vb) ; STOP

g1?xx:Typx !yy[xx>0]      plus d'expression avec LOTOS
```

⌟

Communication par rendez-vous (à la CSP)

Rendez-Vous = moyen de **synchronisation** et de **communication**
– **symétrique** : bloquant pour l'émission **et** la réception

– **binaire** ou **n-aire**

Slide 10

Il existe d'autres modes de communication : asynchrone, diffusion

⌟

structure *if then else*

Exemple

Slide 11

```
ca?vx:int;
if vx >= 0
   then  cb!vx;STOP
   else  cb!-vx;STOP
endif
```

Exercice

Donnez une représentation graphique du processus ainsi décrit.

⌟

Comportements gardés (avec `[garde] -> ...`)

Exemple

Slide 12

```
ca?vx:int;
(
   [ vx > 0 ] ->  cb!vx;STOP
[]
   [ vx = 0 ] -> cb!0;STOP
[]
   [ vx < 0 ] -> cb!vx;STOP
)
```

Exercice

Donner une représentation graphique du processus ainsi décrit.

⌟

Opérateurs de composition parallèle

Notation de CSP : `|[L]|`

`P1 |[L]| P2` signifie :

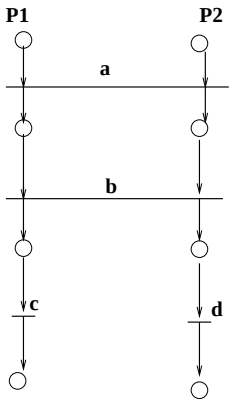
Slide 13

P1 et P2 s'exécutent en parallèle
L est une liste d'actions ou de canaux
P1 et P2 doivent se *synchroniser* sur les actions de L
P1 et P2 ne doivent pas se synchroniser sur des actions n'appartenant pas à L.

Exemple de comportements en parallèle

Slide 14

```
P1 := a;b;c;STOP
P2 := a;b;d;STOP
P1 |[a,b]| P2
```



Sous forme de réseau de Pétri

Modèles sémantiques (interpréter les comportements)

Sémantique opérationnelle

Relation de transition entre les comportements.

Slide 15

$$Terme \rightarrow_{action} Terme$$

Sémantique axiomatique

On donne les propriétés algébriques des différents opérateurs

Failure-divergence

On donne l'ensemble des comportements non autorisés

Conclusion

Slide 16

- De nombreuses autres algèbres processus : ACP, μCRL,...
- Extensions avec gestion de contraintes de temps : TCSP, ...
- Peu utilisées dans les milieux industriels
- LOTOS normalisé, utilisé en industrie
- Des recherches toujours en cours...

Le langage LOTOS

Language of Temporal Ordering Specification

Slide 17

Normalisé ISO 8807, en 1987 après des Drafts en 1985 et 1986
Résultat de travaux de l'OSI dans le cadre de ISO
(années fin 70 - début 80)

Contexte : Techniques de description formelle des protocoles et services

2 groupes de travail OSI/ISO :
- Extension des machines à états ⇒ ESTELLE
- Techniques d'ordonnancement temporel ⇒ LOTOS

Références

Slide 18

- H. Garavel, *Notes de Cours : systèmes temps réels* , ENSIMAG, Grenoble
- Hubert Garavel, *Introduction au Langage LOTOS*, Revue de l'Association des Anciens Elèves de l'ENSIMAG, 1990
- H. Garavel, `http ://www.inrialpes.fr/vasy/cadp/`
- L. Logrippo, M. Faci, M. Haj-Hussein, *An introduction to LOTOS : Learning by Examples*, Computer Networks and ISDN Systems 23(5), 1992

Références (suites)

Slide 19

– A. Strohmeier et D. Buchs, *Génie logiciel : principes, méthodes et techniques*, Presses polytechniques et universitaires romandes, 1996
– Ken Turner, `http ://www.cs.stir.ac.uk/ kjt/research/well/index.html`
– Ken Turner, *Using Formal Description Techniques - An Introduction to Estelle, LOTOS and SDL*, John Wiley and Sons Ltd., 1993

LOTOS - Généralités

Slide 20

LOTOS

utilise CSP et CCS pour décrire le comportement des processus

utilise ACT ONE pour décrire les données
possède de nombreux outils (vérification des propriétés)
par exemple CADP (CAESAR/ALDEBARAN Dev. Pack.) à Grenoble

LOTOS - Généralités

Slide 21

On trouve **trois principales versions** :

- Basic LOTOS
- Full LOTOS
- Enhancement of LOTOS (E-LOTOS)
  nouvelle norme ISO/IEC 15437, 2001

  Complémentarité entre les aspects **données** et **comportements**,
  une expression de comportement représente un état.

⌟

LOTOS - Spécificités

Slide 22

- Exressivité de la composition parallèle
- Synchronisation de processus par **Rendez-vous**

En effet

- synchronisation *multiway* (n-aire) qui vient de CSP
- synchronisation *symétrique* :
  pas de direction, pas de notion d'initiateur de la synchro. et les autres,
  tous les processus participent de façon équitable
- synchronisation *anonyme* (vers l'environnement)
- synchronisation *non-déterministe*

⌟

Spécificités de LOTOS (suite)

Slide 23

– Le comportement d'un processus détermine, à tout moment, quelles actions sont possibles comme action suivante du processus.
On distingue :
  – des actions qu'un processus peut effectuer de façon interne et indépendante : **i** appelée action interne ou silencieuse.
  – des actions qui nécessitent une synchronisation avec l'environnement pour être effectué : elles sont alors *offertes* sur des points de synchronisation appelés *portes* (**gates**).

L'environnement d'un processus est composé des autres processus LOTOS, de l'utilisateur, ou d'autres systèmes externes à LOTOS.

Etude de Basic LOTOS

Slide 24

(Basic LOTOS est aussi appelé Pure LOTOS dans la littérature)

Sous-ensemble de LOTOS où il n'y a pas prise en compte de données.

Seuls les comportements sont spécifiés, pas de spécification des données.

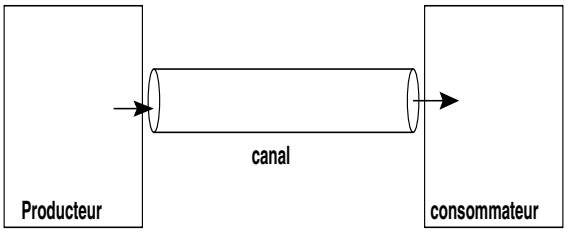
Ici, une action représente l'identification d'une porte.

Pour le reste de l'étude, nous allons nous appuyer sur un exemple illustratif.

Un exemple pour l'étude : producteur-consommateur de ressources

Exemple classique, simple, mais intéressant sur plusieurs points :

Slide 25



Etude (1/28)

- Quel est le comportement du (processus) producteur ?
- Quel est le comportement du consommateur ?
- Qu'est-ce que le canal ? quel modèle ? quel comportement ?
  Informellement structure FIFO à 1 place, 2 places, ...

Slide 26

Quelles stratégies appliquées pour la gestion des ressources ?

- Produire tout, puis consommer tout ?
- produire et consommer alternativement ?

Basic LOTOS / étude (2/28)

Slide 27

Hypothèses de travail :

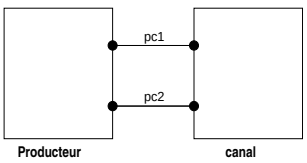
- Le canal représente une mémoire tampon (buffer) de 2 places
- On modélise le canal comme un processus
- Le producteur est un processus qui **doit** produire 2 ressources puis s'arrêter
- Le consommateur est un processus qui doit consommer 2 ressources (ce qui est produit) et s'arrêter.

Soient `pc1`, `pc2` les deux actions du processus producteur : `pc1` représente la production de la ressource 1 vers l'environnement (ici le canal).

Basic LOTOS / étude (2'/28)

Slide 28

Une action identifiant une porte, on peut percevoir le système producteur-canal comme suit :



`pc1`, `pc2` sont des portes.

Basic LOTOS / étude (2"/28)

Le **comportement du processus producteur** s'écrit alors

```
process Producteur [pc1, pc2] : exit :=
  pc1 ; pc2 ; exit
endproc
```

Slide 29

`pc1`, `pc2` sont ici des (portes) paramètres formels (instanciables).
`exit` indique que le processus peut exécuter un `exit` à la fin.

On a utilisé le *préfixage* (noté **;**).

Basic LOTOS / étude (3/28)

Un processus peut se terminer normalement avec succès (`exit`) ou alors sans succès (`noexit`).

Une terminaison avec succès permet au contexte d'exécution de se poursuivre éventuellement avec un autre processus.

Slide 30

Le processus consommateur est spécifié de la même façon par :

```
process Consommateur [cc1, cc2] : exit :=
  cc1 ; cc2 ; exit
endproc
```

`cc1`, `cc2` sont ici les actions de consommation des ressources.
`cc1` représente l'action de consommer la première ressource dans le canal.

Basic LOTOS / étude (4/28)

La **spécification du canal** est comme suit :

```
process Canal [pc1, pc2, cc1, cc2] : exit :=
  pc1; pc2; cc1 ; cc2 ; exit
endproc
```

Slide 31

Maintenant il suffit de composer ces trois processus en parallèle (après avoir étudier les différents opérateurs de composition parallèle)

⌟

Basic LOTOS / étude (5/28)

Nouvelles hypothèses de travail :

– considérons que le canal **peut** délivrer au consommateur, la première ressource produite par le producteur, avant que la deuxième ne soit produite.

Slide 32

Plusieurs possibilités :

– soit le canal se comporte comme dans le cas précédent

– soit après la première production, il se synchronise avec le consommateur sur la porte `cc1`.

⌟

Basic LOTOS / étude (6/28)

On a un choix dans les comportements du canal.

La **spécification du canal** devient comme suit :

Slide 33

```
process Canal [pc1, pc2, cc1, cc2] : exit :=
  pc1; (  pc2 ; cc1 ; cc2 ; exit
        []
          cc1 ; pc2 ; cc2 ; exit
       )
endproc
```

On a utilisé l'opérateur de choix (note `[]`).

Basic LOTOS / étude (6'/28)

L'opérateur de choix `[]` est :
– *commutatif* et
– *associatif*.
On parle alors de *comportements équivalents*,

Slide 34

avec la notion d'**équivalence observationnelle**.

Il y a d'autres types d'équivalence [Milner,Van Glabbek,...].

Basic LOTOS / étude (7/28)

Particularité de `[]` :
Si `C` est un comportement de processus `(C [] stop)` = `C`

Slide 35

Dans un choix, (toutes) les alternatives qui mènent à un deadlock ne sont pas prises en compte.

`a ; b ; exit [] c ; e ; stop` est équivalent à `a ; b ; exit`

Basic LOTOS / étude (8/28)

Dans le comportement précédent du canal, le sous-comportement `cc2 ;exit` est effectué à la fin des deux possibilités.
On peut en faire un processus composé séquentiellement avec le comportement du début du canal.

Slide 36

LOTOS offre l'opérateur `>>` qui permet de **composer deux comportements en séquence**.
Le premier comportement **doit se terminer avec succès** (`exit`) pour que le suivant soit effectué.
Dans ce cas, ce `exit` se transforme en une action δ (une porte spéciale).
Avec `>>`, une action δ se transforme en `i`.

Dans le cas où le premier comportement ne se termine pas avec succès (donc `stop`), la séquence n'est pas poursuivie plus loin.

Basic LOTOS / étude (9/28)

Slide 37

La **spécification du canal** devient comme suit (avec >>) :

```
process Canal [pc1, pc2, cc1, cc2] : exit :=
  pc1; (  pc2 ; cc1 ; exit
        []
          cc1 ; pc2 ; exit
       )
       >> cc2 ; exit
endproc
```

Basic LOTOS / étude (10/28)

Slide 38

Nouvelles hypothèses de travail :

- On considère que le canal n'est pas fiable.
  Il peut perdre des messages.
- La spécification du système doit prendre en compte cette propriété.
- Les ressources consommées sont celles qui ne sont pas perdues sur le canal.

Du point de vue de la spécification, la perte d'un message sur le canal peut se traduire par une *action nulle* du canal pour ce message.

Basic LOTOS / étude (11/28)

Un message produit (`pc1` ou `pc2`) peut ne pas être consommé ; donc absence de cc1 ou `cc2`.

L'action interne `i` peut simuler les pertes comme action nulle.

Slide 39

Différentes possibilités pour le comportement du canal :

- récupérer pc1
  - récupérer pc2 ; délivrer pc1
  - récupérer pc2 ; perte de pc1
  - délivrer pc1 ; récupérer pc2
  - perte de pc1 ; récupérer pc2
- puis soit délivrer pc2 soit perdre pc2

Basic LOTOS / étude (12/28)

Une **spécification du canal** est comme suit :

Slide 40

```
process Canal [pc1, pc2, cc1, cc2] : exit :=
  pc1; (
            pc2 ; cc1 ; exit
          []
            cc1 ; pc2 ; exit  []  i ; pc2 ; exit
        )
          >> (cc2 ; exit [] i ; exit)
endproc
```

Basic LOTOS / étude (13/28)

Modification de la spécification du consommateur.

Il peut :

– consommer les deux ressources si aucune perte,

– consommer une des deux ressources non perdues,

Slide 41

– ne rien consommer si les deux ressources sont perdues.

```
process Consommateur[cc1, cc2] : exit :=
    cc1 ; (  cc2 ; exit []  exit )      (* pas de perte *)
  []
    cc2 ; exit                          (* cc1  perdue *)
  []
    exit                                (* les deux perdues *)
endproc
```

Basic LOTOS

A présent, étude des opérateurs de composition parallèle.

Slide 42

Basic LOTOS / étude comp. parallèle (14/28)

Slide 43

LOTOS offre **trois opérateurs** pour la composition parallèle des processus.
Nous les étudions afin d'utiliser ceux qui sont adéquats pour l'étude du producteur-consommateur.

- opérateur de composition avec *entrelacement* (noté `|||`)
- opérateur de composition *sélective* (noté `|[L]|`)
- opérateur de composition avec *synchronisation complète* (noté `||`)

Basic LOTOS / étude comp. parallèle (15/28)

Slide 44

Opérateur parallèle entrelacement `|||` :
utilisé lorsque les processus parallèles ne se synchronisent pas.

Exemple :

```
b ; c ; exit ||| c ; exit
```

est équivalent à

```
b ;( c ; c ; exit [] c ; c ; exit) [] c ; b ; c ; exit)
```

ou

```
(b ; c ; c ; exit [] c ; b ; c ; exit)
```

L'opérateur `|||` est **commutatif** et **associatif**

Basic LOTOS / étude comp. parallèle (16/28)

Opérateur parallèle sélectif `|[L]|` :
on utilise une liste d'actions ou portes de synchronisation L.
La synchronisation **doit se faire sur les actions de L**.

Note : `|[]|` = `|||`

Slide 45

Exercice : écrire le comportement équivalent à

```
 a ; b ; c ; exit
|[a]|
 d ; a ; c ; exit
```

L'opérateur `|[L]|` est **commutatif**.
L' **associativité** dépend de L.

Basic LOTOS / étude comp. parallèle (17/28)

Opérateur parallèle de synchronisation complète `||` :
Ici l'alphabet d'actions en entier correspond à la liste de synchronisation.
(c'est comme si `||` = `|[alphabet]|`)

Slide 46

Les processus doivent donc se synchroniser sur toutes les actions.

`a ; b ; c ; exit || c ; a ; b ; exit` est équivalent à `stop` (deadlock).

`a ; b ; c ; exit || (a ; b ; exit [] a ; c ; exit)` est équivalent à `a ; stop [] a ; b ; exit`

Basic LOTOS / étude comp. parallèle (18/28)

Opérateur parallèle `||` (suite)

`a ; b ; exit || ( a ; b ; exit [] i ; b ; exit)` est équivalent à

Slide 47

`a ; b ; exit` ou à

`i ; stop`

selon le choix issu de `[]`.

L'opérateur `||` est **commutatif** et **associatif**

⌟

Basic LOTOS / étude (19/28)

Spécification globale du producteur-consommateur :

Slide 48

```
specification producteur_consommateur[pc1,pc2,cc1,cc2]:exit
  behavior ( Producteur [pc1, pc2]
             |||
              Consommateur [cc1, cc2]
           )
           ||
             Canal [pc1, pc2, cc1, cc2]
           (* du fait de || le comportement de Canal prime *)
  where
      ...
```

⌟

Basic LOTOS / étude (20/28)

Spécification globale du producteur-consommateur (suite)

Slide 49

```
where
(* proc. définis auparavent mais avec paramètres formels *)
      process Producteur [o1, o2] : exit := ...

      process Consommateur [i1, i2] : exit := ...

      process Canal [rc1, rc2, cr1, cr2] : exit := ...
endspec
```

Paramètres formels et paramètres effectifs : **instanciation**.

⌟

Basic LOTOS / étude (21/28)

Instanciation des paramètres formels

La liste de portes formelles d'un processus est renommé avec les paramètres effectifs.

Le renommage est fait **dynamiquement** et non statiquement

Slide 50

Le renommage est fait avant que les actions renommées ne soient offertes à l'environnement du processus renommé.

⌟

Basic LOTOS / étude (22/28)

Slide 51

Soit un processus P

```
process P [a, b, c] : noexit :=
   a ; b ; stop |[a]| a ; c ; stop
endproc
```

`P[c, c, a]` équivaut à `c ;(c ;a ;stop [] a ;c ;stop)`

car transformé en `a ;(b ;c ;stop[]c ;b ;stop)` avant renommage

Exercice : quel serait le résultat d'un renommage statique ?

⌟

Basic LOTOS / étude (23/28)

Slide 52

Spécification globale du producteur-consommateur

```
specification producteur_consommateur [pc1,pc2,cc1,cc2]:exit
  behavior (
             Producteur [pc1, pc2]
            |[pc1, pc2]|
             Canal [pc1, pc2, cc1, cc2]
             |[cc1, cc2]|
             Consommateur [cc1, cc2]
           )
  where
      ...
```

⌟

Basic LOTOS / étude (24/28)

Spécification d'un canal non contraint
(qui retire et délivre dans n'importe quel ordre)

Slide 53

```
specification Canal [pc1, pc2, cc1, cc2] :  exit :=

   pc1 ; cc1 ; exit
|||
   pc2 ; cc2 ; exit

endspec
```

Basic LOTOS / étude (25/28)

Spécification d'un canal non fiable

Slide 54

```
specification Canal [pc1, pc2, cc1, cc2] :  exit :=

   pc1 ; (cc1 ; exit [] i ; exit)
|||
   pc2 ; (cc2 ; exit [] i ; exit)

endspec
```

Basic LOTOS / étude (26/28)

L'opérateur d'échappement

LOTOS propose l'opérateur `[>` qui modélise l'interruption **à tout moment** d'un processus principal par un autre auxilliaire.

Slide 55

L'expression `Cp [> Ca` signifie :
à tout moment pendant l'exécution du comportement principale `Cp`, on a le choix entre une action de `Cp` et une action du comportement auxilliaire `Ca`.

Si `Cp` est terminé avant son interruption, `Ca` n'est plus faisable.
Si `Ca` est commencé, on ne peut plus exécuter aucune action de `Cp`.

⌟

Basic LOTOS / étude (27/28)

Spécification d'un canal non fiable

Supposons que le canal peut devenir défectueux à tout moment.
Il pourrait devenir silencieux après n'importe quelle action.

Slide 56

```
process Canal [pc1, pc2, cc1, cc2] : exit :=
 (       ...              (* comme précédemment *)
 )
 [> ( i ; exit)     (* à tout moment le canal peut 'tomber' *)
endproc
```

Dans un tel cas le comportement du consommateur doit changer.
Exercice : écrire la spécification du consommateur dans le cas où le canal peut est défectueux.

⌟

Basic LOTOS / étude (28/28)

Slide 57

L'opérateur de masquage

Lors de la composition de processus, on peut masquer certaines actions.

LOTOS propose l'opérateur `hide L in ...` qui permet de masquer une liste d'actions L dans un comportement donné.

Les actions masquées ne sont pas observables, elles sont transformées en action interne : **i**.

Exemple :

`(hide b in a ; b ; c ; exit) || a ; c ; exit`

équivaut à `a ;i ;c ;exit`

Exercice : hide `b` **in** `(a ; b ; c ; exit || a ; c ; exit)` = ??

Références

Slide 58

- H. Garavel, *Notes de Cours : systèmes temps réels* , ENSIMAG, Grenoble
- Hubert Garavel, *Introduction au Langage LOTOS*, Revue de l'Association des Anciens Elèves de l'ENSIMAG, 1990
- H. Garavel, `http ://www.inrialpes.fr/vasy/cadp/`
- L. Logrippo, M. Faci, M. Haj-Hussein, *An introduction to LOTOS : Learning by Examples*, Computer Networks and ISDN Systems 23(5), 1992

Références (suite)

Slide 59

– A. Strohmeier et D. Buchs, *Génie logiciel : principes, méthodes et techniques*, Presses polytechniques et universitaires romandes, 1996
– Ken Turner, `http ://www.cs.stir.ac.uk/ kjt/research/well/index.html`
– Ken Turner, *Using Formal Description Techniques - An Introduction to Estelle, LOTOS and SDL*,John Wiley and Sons Ltd., 1993

Etude de Full LOTOS

LOTOS avec prise en compte de données.

Slide 60

Augmentation de Basic LOTOS avec
– spécification des données et
– leur utilisation dans les comportements.
Ici, une action est composée de :
– l'identification d'une **porte**,
– une liste d'**événements** et
– optionnellement un **prédicat**.

Etude de Full LOTOS

Slide 61

Un **événement** peut
– soit offrir (**!**) une valeur,
– soit accepter (**?**) une valeur.
Les processus se synchronisent si
– **ils utilisent le même nom d'action (porte)**
– **ils ont des listes d'événements qui sont en correspondance**
– **leur prédicats sont vrais**

Etude de Full LOTOS

Exemples d'actions

Slide 62

– `p ? x:Nat !1 [ x >= 0 ]`
attend de l'environnement sur la porte p, un entier positif ou nul, puis émet 1.
– `p ? x:Nat !0 [ x < 0 ]`
attend de l'environnement sur la porte p, un entier négatif, puis émet 0.
– `p ? x:Nat !2 [ x =< 2 ]`

Etude de Full LOTOS

Spécificités de Full LOTOS

La terminaison avec succès (`exit`) peut être paramétrée.

```
exit(e1,...,en)
```

Slide 63

cela correspond à une émission sur la porte spéciale δ

Les comportements peuvent être gardés

Pour le reste de l'étude, nous allons encore nous appuyer sur un exemple illustratif.

⌟

Etude de cas : le protocole du bit alterné (ABP)

Description

Le protocole du bit alterné (Alternating Bit Protocol) fait partie de la couche transport du modèle OSI.

Il permet le **transfert de données entre une paire d'entités**

Slide 64

pour lesquelles une **connexion bi-directionnelle** a été préalablement établie.

On distingue deux entités : T (Transmitter) émet des messages à destination de R (Receiver).

Les **messages sont modélisés par des numéros** compris entre 1 et un entier naturel maximal N ; ils sont spécifiés par un type abstrait général `MESSAGE` offrant la sorte `MSG`.

⌟

Description du protocole

Le **fonctionnement idéal** du protocole ABP est le suivant :

T envoie un message à R ;
à la réception de ce message, R renvoie un acquittement à T.

Slide 65

La **liaison entre T et R n'est pas fiable** $\Rightarrow$ perte de messages ou d'acquittements.
En cas de perte, le medium peut de façon **facultative**, signaler cette perte au destinataire (T ou R) en envoyant une **indication de perte**.

⌟

Description du protocole (suite)

Pour **détecter les pertes non signalées**, les messages et les acquittements contiennent un **bit de contrôle**.
Le bit de contrôle de chaque acquittement est égal au bit de contrôle du message qu'il acquitte.
Les bits de contrôle de deux messages successivement émis ont des valeurs distinctes (**la valeur du bit alterne à chaque émission**).

Slide 66

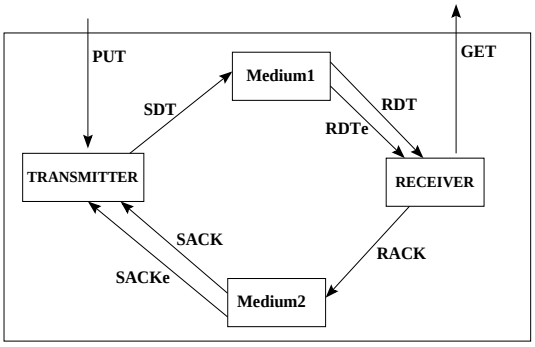
Si l'entité T reçoit une indication de perte d'acquittement ou un acquittement avec un bit de contrôle erroné, elle réemet le dernier message envoyé.

Si l'entité R reçoit une indication de perte de message ou un message avec un bit de contrôle erroné, elle réemet le dernier acquittement envoyé.

⌟

Architecture du protocole

Slide 67



Spécification en LOTOS

Les processus : `TRANSMITTER, RECEIVER, MEDIUM1, MEDIUM2`

Les signaux principaux : `PUT, GET`
Les signaux internes : `SDT,RDT, RDTe, RACK, SACK, SACKe`

Slide 68

`RDTe` et `SACKe` représentent les pertes de message et d'acquittement.

`SDT !M !B` représente l'émission du message de `T` vers `MEDIUM1`.
`RACK !B` représente l'émission d'un acquit. de `R` vers `MEDIUM2`.
`i` représente un signal spécial (*silence*).

Spécification globale du protocole

Slide 69

```
specification ABP [PUT, GET] : noexit behaviour
    hide SDT, RDT, RDTe, RACK, SACK, SACKe in
    (( TRANSMITTER [PUT, SDT, SACK, SACKe] (0)
        (* le 0 permet d'initialiser le bit de controle du premier mess. *)
       |||
       RECEIVER[GET, RDT, RDTe, RACK](0)
     )
    |[SDT,RDT,RDTe, RACK, SACK, SACKe]|
     ( MEDIUM1 [SDT, RDT, RDTe]
       |||
       MEDIUM2 [RACK, SACK, SACKe]
     )
    ) ...
```

Slide 70

```
...
   where

     type BIT is
        sorts BIT
        opns 0 : -> BIT
             1 : -> BIT
           not : BIT -> BIT
     endtype
endspec
```

Spécification du medium des messages (Medium1)

Slide 71

En recevant un message M avec un bit de contrôle égal à B, le medium 1 peut réagir de la façon suivante :

– transmettre correctement le message et son bit de contrôle (la valeur du bit de contrôle ne change pas).

– perdre le message et envoyer une indication de perte à l'entité réceptrice.

– perdre silencieusement le message (pas d'indication)

on écrit alors (un choix entre) ces comportements disctincts.

Slide 72

```
process MEDIUM1 [SDT, RDT, RDTe] : noexit :=
  SDT ?M:MSG  ?B:BIT; (* reception d'un message*)
   (
    RDT !M !B;       (* transmission correcte *)
       MEDIUM1 [SDT, RDT, RDTe]
    []
    RDTe;            (* perte avec indication *)
       MEDIUM1 [SDT, RDT, RDTe]
    []
    i;               (* perte silencieuse *)
       MEDIUM1 [SDT, RDT, RDTe]
   )
  endproc
```

Spécification du medium 2 des acquittements

Le fonctionnement est similaire à celui du medium1 (les noms ne sont pas les mêmes, il n'y a que le bit de contrôle).

Slide 73

```
process MEDIUM2 [RACK, SACK, SACKe] : noexit :=
  RACK ?B:BIT;        (* reception d'un acquit *)
   ( SACK  !B;        (* transmission correcte *)
       MEDIUM2 [RACK, SACK, SACKe]
    []
    SACKe;             (* perte avec indication *)
       MEDIUM2 [RACK, SACK, SACKe]
    []
    i;                (* perte silencieuse *)
       MEDIUM2 [RACK, SACK, SACKe]
   )
  endproc
```

⌟

Spécification de l'émetteur (T)

L'émetteur reçoit un bit via PUT et le transmet au medium 1 après avoir ajouté la valeur courante B du bit de contrôle. S'il reçoit en réponse un acquittement avec un bit de contrôle B la transmission a réussi, sinon il faut réemettre le message.

Slide 74

Il y a **3 cas de réémission :**

- l'émetteur a reçu un acquittement ayant la valeur $(\neg B)$
- l'émetteur a reçu une indication de perte d'acquittement `SACKe`
- l'émetteur peut réémettre spontanément le message afin d'éviter le blocage dans le cas où le medium1 (2) aurait perdu silencieusement un message (un acquittement) ;
  c'est un contournement du *timeout*

⌟

Une spécification de l'émetteur

Slide 75

```
process TRANSMITTER [PUT, SDT, SACK, SACKe] (B : BIT) :noexit :=
 PUT ?M:MSG;   (* acquisition d'un message *)
  TRANSMIT [PUT, SDT, SACK, SACKe] (B, M)
where
  process TRANSMIT [PUT, SDT, SACK, SACKe](B:BIT, M:MSG) :noexit :=
    SDT !M !B;    (* emission du message *)
    (SACK !B;     (* Bit de controle correct *)
      TRANSMIT [PUT, SDT, SACK, SACKe] (B,M)
  []   SACKe;       (* indication de perte : reemission *)
       TRANSMIT [PUT, SDT, SACK, SACKe] (B,M)
  []   i;           (* timeout : reemission *)
      TRAMSIT  [PUT, SDT, SACK, SACKe] (B,M)
     )
  endproc
endproc
```

Spécification du récepteur

Slide 76

```
process RECEIVER [GET, RDT, RDTe, RACK] (B:BIT) : noexit :=
     RDT ?M:MSG !B;     (* bit de control correct *)
       GET !M;           (* emission du message *)
        RACK !B;         (* envoi d'acquit *)
         RECEIVER [GET, RDT, RDTe, RACK](not(B))
 []  RDT ?M:MSG !(not(B)) ; (*bit de controle incorrect *)
       RACK !(not(B));  (* envoi d'un acquitt. incorrect *)
        RECEIVER [GET, RDT, RDTe, RACK](B)
 []  RDTe;  RACK !(not(B));  (* envoi d'un aquitt incorrect *)
       RECEIVER [GET, RDT, RDTe, RACK](B)
 []  i;                (* timeout *)
       RACK !(not(B));  (* envoi d'un acquittement incorrect *)
        RECEIVER [GET, RDT, RDTe, RACK](B)
    endproc
```

Références

Slide 77

- H. Garavel, *Notes de Cours : systèmes temps réels* , ENSIMAG, Grenoble
- Hubert Garavel, *Introduction au Langage LOTOS*, Revue de l'Association des Anciens Elèves de l'ENSIMAG, 1990
- H. Garavel, `http ://www.inrialpes.fr/vasy/cadp/`
- L. Logrippo, M. Faci, M. Haj-Hussein, *An introduction to LOTOS : Learning by Examples*, Computer Networks and ISDN Systems 23(5), 1992

Références (suite)

Slide 78

- A. Strohmeier et D. Buchs, *Génie logiciel : principes, méthodes et techniques*, Presses polytechniques et universitaires romandes, 1996
- Ken Turner, `http ://www.cs.stir.ac.uk/ kjt/research/well/index.html`
- Ken Turner, *Using Formal Description Techniques - An Introduction to Estelle, LOTOS and SDL*,John Wiley and Sons Ltd., 1993