16 Le langage LOTOS

16.1 Aperçu du langage LOTOS

LOTOS = *Language Of Temporal Ordering Specification*

Un des deux langages formels standardisés par l'ISO pour la spécification de systèmes distribués et de protocoles de communication

- Le standard existe depuis la fin des années 80
- Les concepts à la base du langage sont plus anciens:
  - CCS = *Calculus of Communicating Systems*
  - CSP = *Communicating Sequential Processes*

LOTOS dans son ensemble est constitué de deux parties:

1. une notation pour les processus concurrents (aspect contrôle);
2. une notation (algébrique) pour les types de données

De nombreux outils sont disponibles:

- Vérification de la syntaxe et des types
- Simulation et exécution
- Génération de code
- Vérification de propriétés temporelles

16.2 Les principaux concepts de LOTOS

Notion fondamentale = *expression de comportement*

Rôle = décrit les séries d'*actions* que le processus peut exécuter

16.2.1 Processus inactif

STOP = processus bloqué, inactif

EXIT = processus qui termine avec succès

16.2.2 Séquence d'actions et choix

Soient a, b et c des actions possibles:

```
a; a; c; b; STOP
```

Séquence d'actions d'un sémaphore

```
prendreSemaphore; relacherSemaphore
```

Choix entre deux expressions de comportement:

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

Choix entre plusieurs expressions:

```
AjouterClient ... [] Solde ... [] Crediter ... [] Debiter ...
```

Impact du moment du choix:

(1) `a; b; STOP [] a; c; STOP`
(2) `a; (b; STOP [] c; STOP)`

16.2.3 Déclaration de processus et création d'instances

Soit la définition suivante:

```
PROCESS Proc[a1, a2]: EXIT :=
  a1; a2; a1; EXIT
ENDPROC
```

Soit l'expression de comportement suivante:

```
x; z; Proc[a, b]
```

Les actions effectuées seraient les suivantes:

```
x; z; a; b; a
```

16.2.4 Processus récursifs

Premier exemple:

```
PROCESS Semaphore[p, v]: NOEXIT :=
  p; v; Semaphore[p, v]
ENDPROC
```

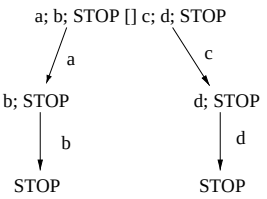
Autre exemple de processus cyclique infini:

```
PROCESS Horloge[tic]: NOEXIT :=
  tic; Horloge[tic]
ENDPROC
```

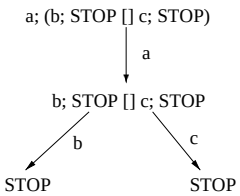
16.2.5 Processus et diagrammes de transition

Expression de comportement ↔ État
Action ↔ Transition

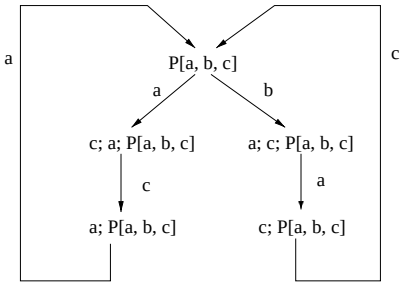
```
a; b; STOP [] c; d; STOP
```




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



```
PROCESS P[a, b, c]: NOEXIT :=
  a; c; a; P[a, b, c]
  []
  b; a; c; P[a, b, c]
ENDPROC
```



16.2.6 Exemples de machines distributrices

- Machine distributrice de biscuits et muffins (Figure 16.4):
  - Accepte des pièces de 0.25$ et de 1.00$;
  - Un muffin coûte 1.00$
  - Un biscuit coûte 0.75$;
  - La monnaie est rendue;
  - Pas d'annulation possible.

- Machine distributrice de café avec bouton d'annulation (Figure 16.7):
  - Accepte uniquement des pièces de 0.25$;
  - Un café coûte 0.75$;
  - Un bouillon coûte 0.50$;
  - Permet d'annuler la transaction en cours.

16.2.7 Problème du producteur–consommateur

Différentes versions:

- Alternance stricte *sans* transmission d'information
- Tampon de taille 1 avec transmission d'information

16.2.8 Calcul du maximum de 3 nombres

Maximum de 3 nombres obtenu par composition parallèle de deux instances de processus pour calculer le maximum de 2 nombres.

16.2.9 Modélisation d'une variable partagée

Premier exemple illustrant un état complexe, c'est-à-dire, un processus paramétrisé par des valeurs qui déterminent son état

16.2.10 Producteur–consommateur avec tampon partagé

Nouvelle version avec tampon partagé (taille `>=` 1) n'obligeant pas à une alternance stricte entre le producteur et le consommateur.

16.2.11 Le problème du paquetage/dépaquetage de Jackson

Un problème relativement difficile à résoudre dans un contexte strictement séquentiel, mais très facile lorsque des processus concurrents et communiquants sont permis

16.3 Éléments additionnels du langage LOTOS

- Opérateurs de composition parallèle:
  - `||`: Synchronisation complète entre processus
  - `|||`: Entrelacement pur (sans synchronisation)
  - `||` = `|[`*tous les ports possibles*`]|`
  - `|||` = `|[]|`
- `EXIT` = processus qui se termine *avec succès*, en exécutant l'action spéciale δ (signal de terminaison)

  Le signal δ peut ensuite servir à activer un autre processus (`>>`)
- `EXIT(v1, ..., vk)` = processus qui se termine avec succès en retournant les résultats indiqués, lesquels peuvent être reçus par une expression `ACCEPT`:

  ```
  >> ACCEPT x1: t1, ..., xk: tk IN ...
  ```
- Déclaration locale: `LET ... IN ...`

16.4 Quelques exemples additionnels de spécification

16.4.1 Maximum parmi deux nombres (bis)

Trouve le maximum parmi deux nombres, mais en permettant de recevoir ces nombres dans n'importe quel ordre

16.4.2 Tri d'une suite de nombres

Version LOTOS d'un exemple déjà vu en Spec (Chap. 14)

16.4.3 Conversion 7 bits à 8 bits par ajout d'un bit de parité

Illustration de l'utilisation de LET

16.5 Sémantique opérationnelle

Règles qui définissent, *de façon formelle*, la sémantique (opérationnelle) des diverses expressions du langage LOTOS

16.6 Équivalences entre processus

Différentes façons pouvant permettre de déterminer si deux processus (deux expressions de comportement) sont équivalents ou non

16.6.1 Équivalence en termes d'ensembles de traces: Deux processus sont équivalents si leurs comportements produisent exactement les mêmes traces

16.6.2 Bisimilarité forte: Deux processus sont équivalents s'ils peuvent se simuler l'un et l'autre de façon *forte*, y compris au niveau des actions invisibles

16.6.3 Bisimilarité faible: Deux processus sont équivalents s'ils peuvent se simuler l'un et l'autre de façon *faible*, en ignorant les actions invisibles

16.6.4 Congruence observationnelle: Compromis entre les deux formes de bisimilarité

16.7 L'exemple de la banque

Présentation d'une version LOTOS d'un exemple vu antérieurement en Spec (qui sera vu en VDM et Z)

Bien que l'on puisse voir relativement facilement la correspondance entre les deux versions, la version LOTOS, à cause des noms de ports/actions à indiquer, semble plus *lourde*