Formalismes de Description de Systèmes

Alain Griffault
Université Bordeaux I

Frédéric Herbreteau
Enseirb

27 octobre 2004

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Le besoin de formalismes

Montrer le respect du cahier des charges.
Dialoguer sans ambiguité.
Comparer des solutions.
Abstraire un système.

1.2 À quoi sert la formalisation des systèmes ?

systèmes critiques et ou complexes.
historiques (industriels et recherche).

1.3 Comment comparer les formalismes

theorique : expressivite,
pratique : efficacité, adéquation avec les besoins utilisateurs, outillage.

1.4 Le contenu de ce cours

Approche intuitive des besoins pour la représentation des systèmes.
Les systèmes à contrôle fini.
Les systèmes à contrôle infini.
Les langages de description.
Les réseaux de Petri.
Les algèbres de processus.
À chaque fois, définition, exemples, proprietés, utilisation.

Chapitre 2

Modélisation de systèmes

2.1 Étude de programmes parallèles

2.1.1 Communication par variables partagées

Soient les deux processus suivant et la variable globale `compte` initialisée à 10.

Processus A	Processus B
var aux : integer	var aux : integer
begin	begin
aux := compte ;	aux := compte ;
aux := aux + 10 ;	aux := aux - 5 ;
compte := aux ;	compte := aux ;
end.	end.

Question 2.1 *Quelles sont les valeurs possibles de la variable* `compte` *après les exécutions des deux processus A et B ?*

Corrigé question 2.1
$\{5, 15, 20\}$

Question 2.2 *Cherchez une méthode simple qui permette aisément de montrer que votre réponse à la question précédente est correcte ?*

Corrigé question 2.2
On représente l'évolution des valeurs des trois variables, en fonction des actions des deux processus. On obtient la figure 2.1.

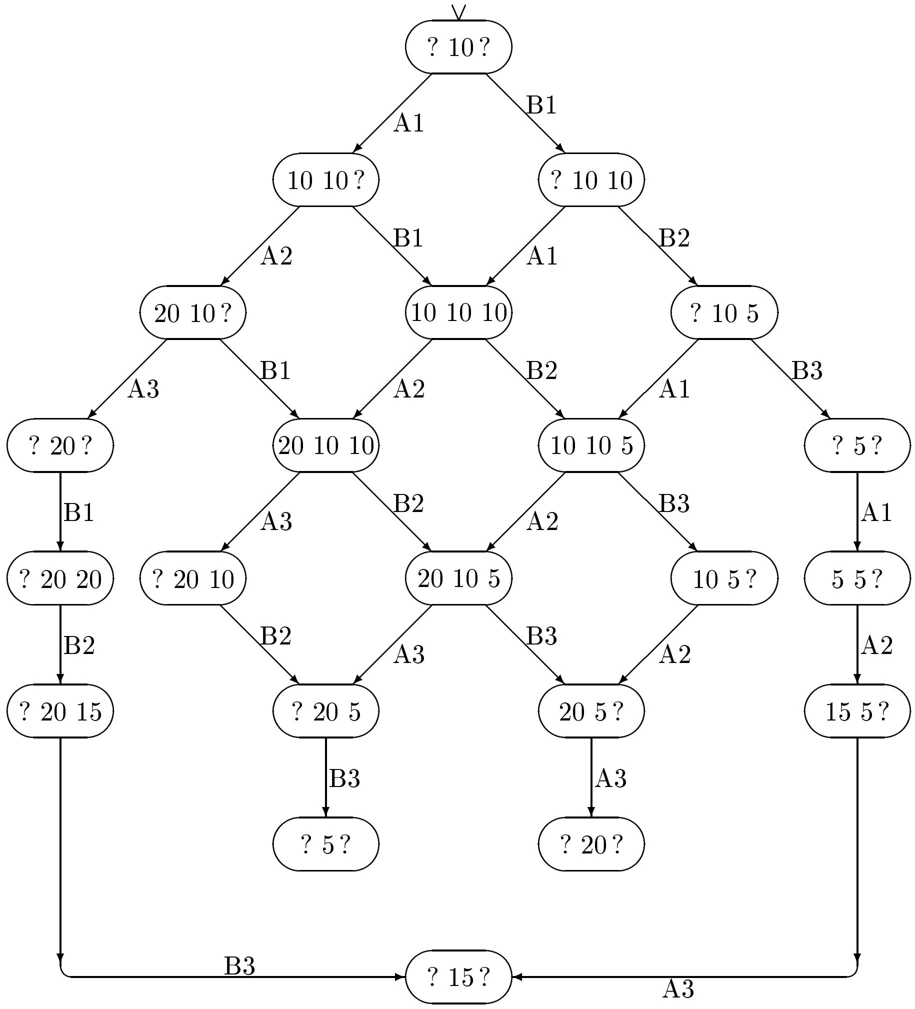

FIG. 2.1 – Les valeurs possibles pour compte

2.1.2 Communication par messages

Soient les deux processus suivant :

```
Processus P(i) /* i de 0 a 1 */
begin
while true do
        begin
        F(i) ! prendre ;
        F(1-i) ! prendre ;
        F(1-i) ! poser ;
        F(i) ! poser ;
        end ;
end.
```

```
Processus F(i) /* i de 0 a 1 */
begin
while true do
        begin
        P(0) ? prendre → P(0) ? poser ;
        or
        P(1) ? prendre → P(1) ? poser ;
        end ;
end.
```

Question 2.3 *Appliquez la méthode précédente pour mettre en évidence le problème principal de ce programme.*

Corrigé question 2.3
On représente les compteurs ordinaux des deux philosophes et qui tient la fourchette. La figure 2.2 correspond

à cette modélisation.

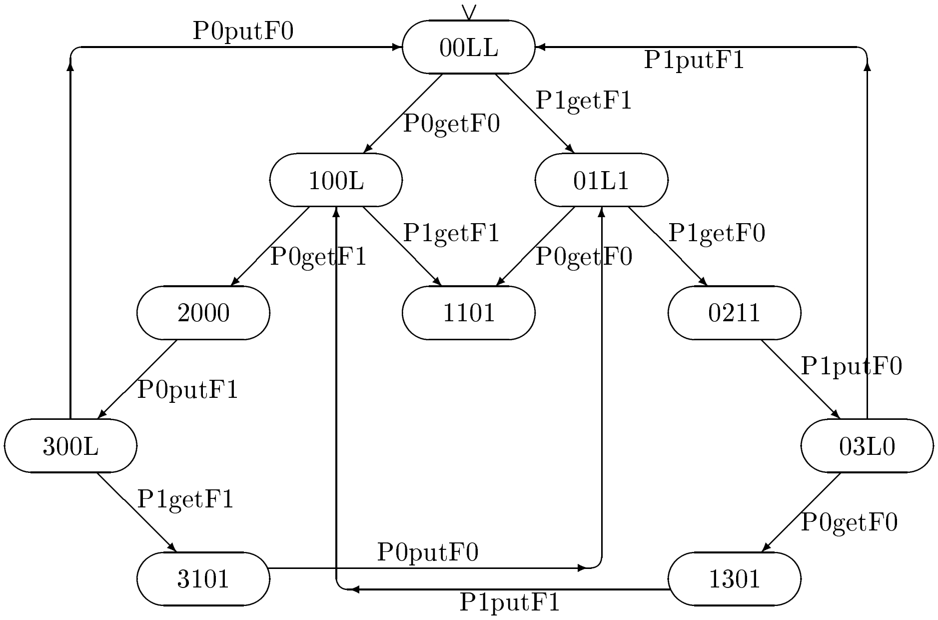

FIG. 2.2 – Les philosophes mangeurs de spaghettis

On peut constater sur la figure 2.2 les faits suivants :
- Il n'y a pas de transition qui parte de l'état 1101. Cela correspond à un blocage du système global.
- Il existe une boucle qui ne passe pas par l'état initial.
- Il existe des boucles *non-équitable*

2.2 Analyse de la méthode

Essayez de répondre aux questions suivantes :

Question 2.4 *A quoi correspondent les sommets d'un tel graphe ?*

Corrigé question 2.4
À une abstraction d'un ou de plusieurs états du système réel.

Question 2.5 *A quoi correspondent les arrêtes d'un tel graphe ?*

Corrigé question 2.5
À une abstraction d'une ou de plusieures évolutions du système réel.

Question 2.6 *A quoi correspond un sommet duquel part aucune transition ?*

Corrigé question 2.6
À une abstraction d'un blocage du système réel. Le système peut ne pas être bloqué, mais pour l'observation induite par l'abstraction, il n'évolue plus.

Première partie

Les systèmes à contrôle fini

Chapitre 3

Les systèmes réactifs discrets

3.1 Quelques définitions

Definition 3.1 *Un système de transition est un triplet* $S =< Q, I, T >$ *où :*
- Q *est un ensemble d'états.*
- I *est le sous-ensemble des états* initiaux.
- $T \subseteq Q \times Q$ *est un ensemble de transitions.*

Parmi les variantes des systèmes de transition on distingue :

Definition 3.2 *Un système de transition* **étiqueté** *est un quadruplet* $S =< Q, I, A, T >$ *où :*
- Q *est un ensemble d'états.*
- I *est le sous-ensemble des états* initiaux.
- A **est un ensemble d'actions ou alphabet.**
- $T \subseteq Q \times A \times Q$ *est un ensemble de transitions.*

Definition 3.3 *Une machine de Mealy est un quintuplet* $S =< Q, I, In, Out, T >$ *où :*
- Q *est un ensemble d'états.*
- I *est le sous-ensemble des états* initiaux.
- In **est un alphabet représentant des actions.**
- Out **est un alphabet représentant des réactions.**
- $T \subseteq Q \times In \times Out \times Q$ *est un ensemble de transitions.*

On distingue également les systèmes de transition finis des systèmes de transition infinis.

Definition 3.4 *Un système de transition* **étiqueté fini** *est un quadruplet* $S =< Q, I, A, T >$ *où :*
- Q *est un ensemble d'états.*
- I *est le sous-ensemble des états* initiaux.
- A *est un ensemble d'actions ou alphabet.*
- $T \subseteq Q \times A \times Q$ *est un ensemble* **fini** *de transitions.*

Remarque : si le nombre de transitions est fini, le nombre d'états *interressant* et le nombre d'actions *interressantes* de l'alphabet sont finis.

3.2 Les systèmes réactifs discrets finis

Ils sont modélisables par des systèmes de transition finis. L'avantage principal est que tous les problèmes sur une telle structure sont décidables.

3.2.1 Modélisation par systèmes de transition finis

Modélisation des données

Dessiner les systèmes de transition des objets suivants :

Question 3.1 *Une variable entière comprise entre 0 et 3*
- *les états ;*
- *l'opération d'affectation ;*
- *l'opération de lecture*

Corrigé question 3.1

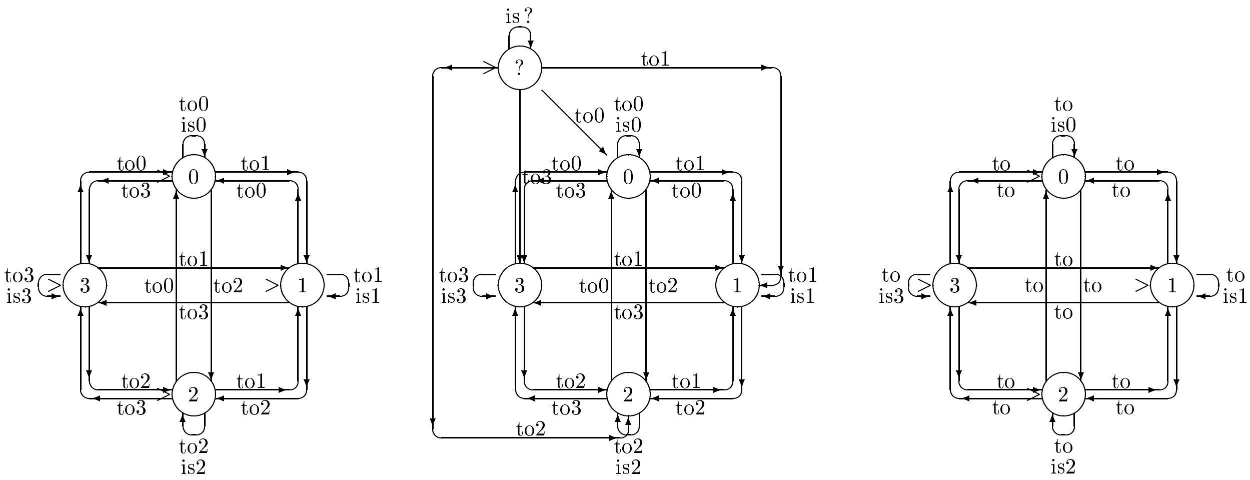

Fig. 3.1 – Trois versions d'une variable entière comprise entre 0 et 3.

Question 3.2 *Un buffer de taille 1.*

Corrigé question 3.2

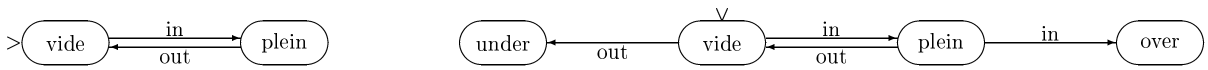

Fig. 3.2 – Deux versions d'un buffer de taille 1.

Question 3.3 *Une file d'attente de taille 2.*

Corrigé question 3.3

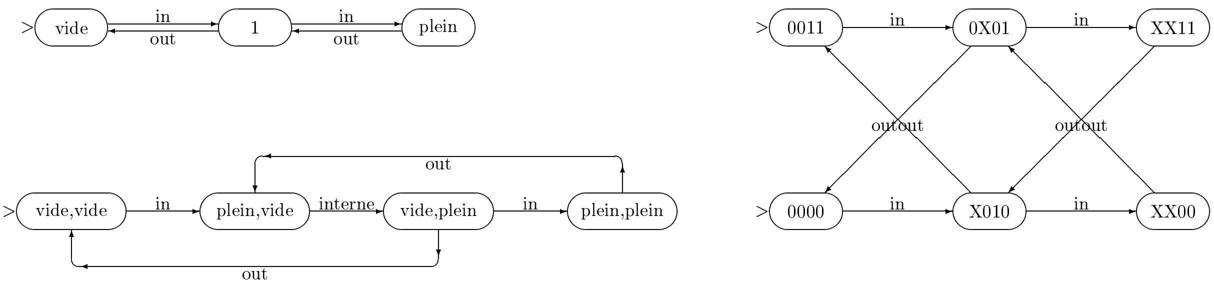

FIG. 3.3 – Trois versions d'une FIFO de taille 2.

Question 3.4 *Une pile d'attente de taille 2.*

Corrigé question 3.4

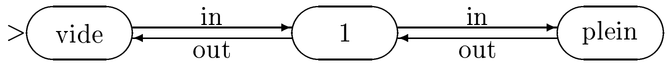

FIG. 3.4 – Une LIFO de taille 2.

Modélisation de comportements simples

Dessiner les systèmes de transition des objets suivants :

Question 3.5 *Un contrôleur d'une salle limitée à dix places.*

Corrigé question 3.5

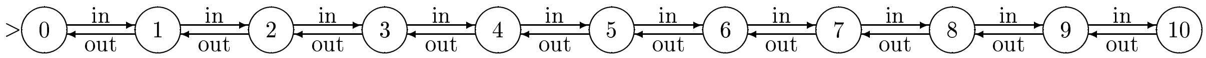

FIG. 3.5 – Une salle de 10 places.

Question 3.6 *Un contrôleur de bibliothèque dans laquelle un écrivain est nécessairement seul, et ou les lecteurs peuvent se côtoyer. La première personne à accéder à la bibliothèque est un écrivain.*

Corrigé question 3.6

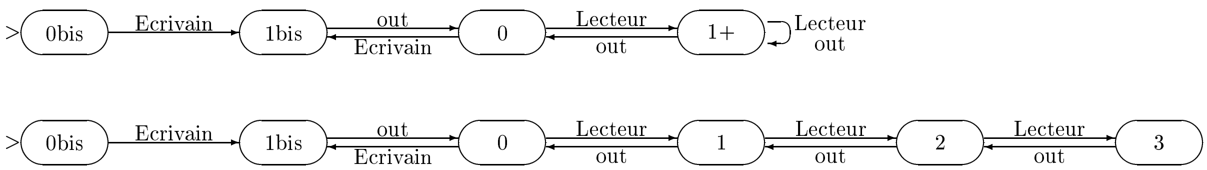

FIG. 3.6 – Deux versions d'un contrôleur de biliothèque.

Question 3.7 *Une porte logique* **ET**.

Corrigé question 3.7

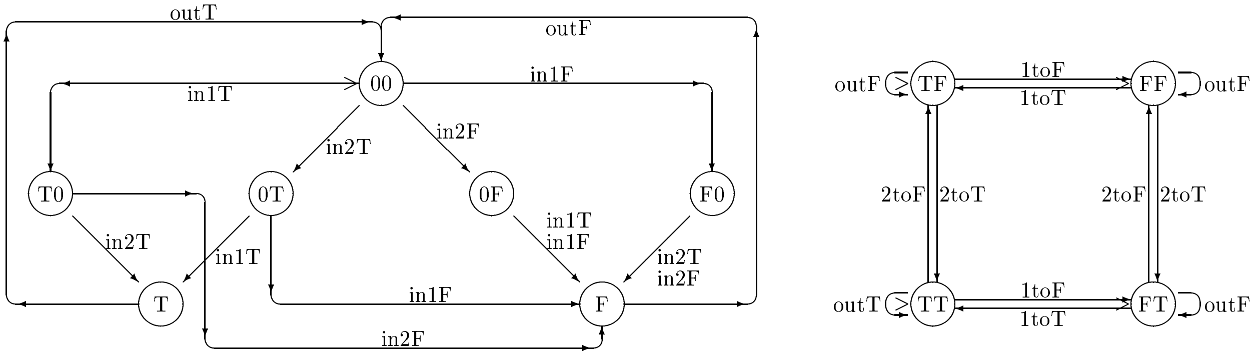

FIG. 3.7 – Deux versions d'une porte ET.

Question 3.8 *Un processus de l'algorithme d'exclusion mutuelle de Peterson dans le cas de deux processus.*

```
while true do begin
              F[i] := true;
              Tour := i;
              wait (F[j]=false or Tour = j);
              F[i] := false;
              end;
```

Corrigé question 3.8

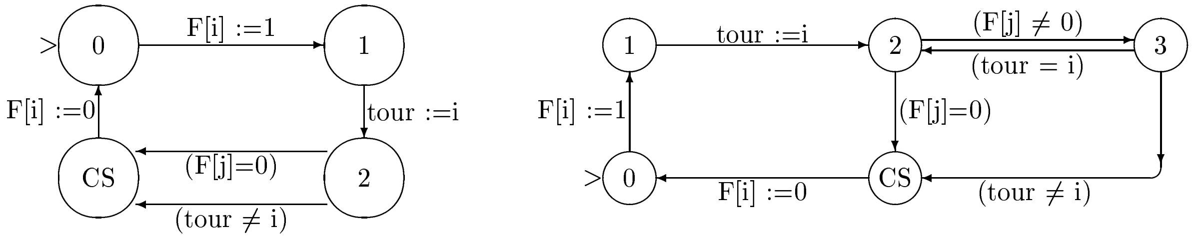

FIG. 3.8 – Deux versions d'un processus de Peterson.

Question 3.9 *L'émetteur dans le protocole du bit alterné.*

```
parite := true
while true do begin
              recevoir(message); /* du producteur */
              emettre(message,parite);
              recevoir(ack,pariteAck);
              while parite <> pariteAck do begin
                                           emettre(message,parite);
                                           recevoir(ack,pariteAck);
                                           end;
              parite := not parite;
              end;
```

Corrigé question 3.9

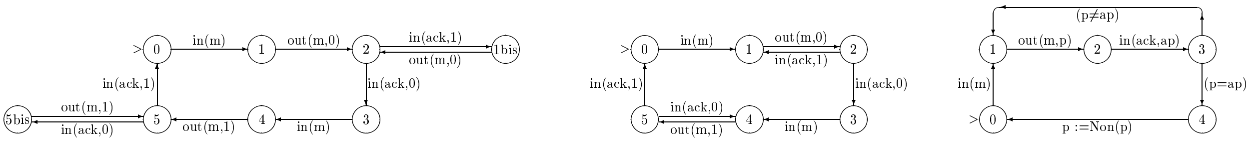

FIG. 3.9 – Trois versions de l'emetteur ABP.

Question 3.10 *le récepteur dans le protocole du bit alterné.*

```
pariteAck := true
while true do begin
              recevoir(message,parite);
              emettre(ack,parite);
              while parite <> pariteAck do begin
                                           recevoir(message,parite);
                                           emettre(ack,parite);
                                           end;
              emettre(message); /* vers le consommateur */
              pariteAck := not pariteAck;
              end;
```

Corrigé question 3.10

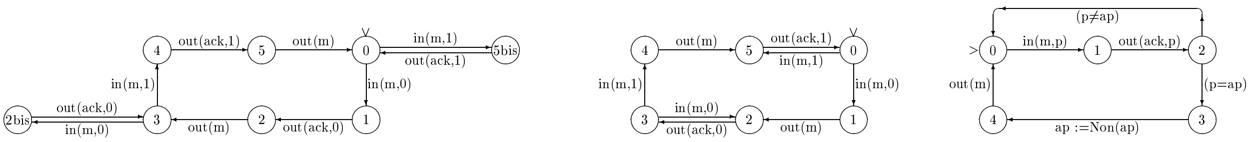

FIG. 3.10 – Trois versions du recepteur ABP.

Modélisation de comportements complexes

Question 3.11 *Essayer de dessiner le système de transition de l'algorithme d'exclusion mutuelle de Peterson dans le cas de deux processus.*

Corrigé question 3.11
Il est pratiquement impossible de dessiner sans erreur un automate ayant plus d'une vingtaine d'états et plus d'une cinquantaine de transitions.

Il faut beaucoup de rigueur et de méthodologie pour dessiner sans erreur un automate ayant plus d'une dizaine d'états et plus d'une vingtaine de transitions.

3.2.2 Le produit de synchronisation

Le comportement global d'un ensemble de processus concurrents est calculé à partir : d'une part des descriptions des comportements de chacun des processus; d'autre part d'une description de leurs interactions. Les comportements des composants sont décrits par des systèmes de transition, et les interactions par un système de synchronisation. Le produit de synchronisation calcule le système de transition comme la restriction du produit libre des systèmes de transition aux seules transitions qui correspondent aux interactions permises. Plus formellement :

Un système de synchronisation pour un ensemble $\{S_1, \ldots, S_n\}$ de systèmes de transition est un ensemble de vecteurs de synchronisation.
Un vecteur de synchronisation est un élément de $\prod_{i=1}^{i=n} A_i$, donc un n-uplet $(a_1, \ldots, a_n)$ et intuitivement représente une évolution globale permise du système à un instant donné et indépendamment du contexte (i.e. de l'état du système).

Le produit de synchronisation d'un ensemble $\{S_1, \ldots, S_n\}$ de systèmes de transition, et d'un système de synchronisation V est le système de transition $S =< Q, I, A, T >$ défini par :
- $Q = \prod_{i=1}^{i=n} Q_i$.
- $I = \prod_{i=1}^{i=n} I_i$.
- $A = \prod_{i=1}^{i=n} A_i$.
- $T = T_l \cap Q \times V \times Q$ avec
  $T_l = \{((q_1, \ldots, q_n)(a_1, \ldots, a_n)(q'_1, \ldots, q'_n)) \ / \ (q_i, a_i, q'_i) \in T_i\}$

Intuitivement, T est égal au produit libre des systèmes de transition T_l diminué de toutes les transitions dont l'étiquette $(a_1, \ldots, a_n)$ n'est pas dans le système de synchronisation V.

Un exemple

La figure 3.11 représente le comportement d'un compteur à un chiffre (2 valeurs possibles) avec une possibilité d'un comportement erroné lorsqu'il se trouve dans l'état `1`.
Il faut noter, sur chacun des états, la présence d'une boucle étiquetée par `e`. La sémantique intuitive de cette transition est : *je ne fais rien, je ne participe pas.* Cette transition lorsqu'elle est présente sur tous les états de tous les composants permet de modéliser tous les phénomènes asynchrones dans ce modèle fondamentalement synchrone.

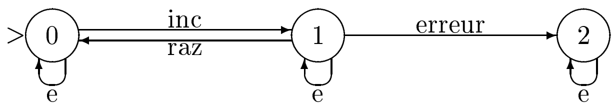

FIG. 3.11 – Une roue de compteur kilométrique

La définition formelle de ce système de transition est :
- $Q = \{0, 1, 2\}$ l'ensemble des états.
- $I = \{0\}$ l'état initial.
- $A = \{e, erreur, inc, raz\}$ l'ensemble d'actions.
- $T = \{(0, e, 0), (1, e, 1), (2, e, 2), (0, inc, 1), (1, raz, 0), (1, erreur, 2)\}$
  l'ensemble des transitions.

Considérons un système global composé de deux instances de l'objet représenté par la figure 3.11. La table 3.2 représente les actions globales autorisées à un instant donné. Dans cet exemple, deux variables à un chiffre permettent de construire un compteur kilométrique à deux chiffres. Le phénomène d'entraînement des roues dentées est modélisé par une synchronisation entre : d'une part une incrémentation `inc` sur la première composante du système ; d'autre part une remise à zéro `raz` sur la deuxième composante du système.

ST variable	ST variable
e	inc
inc	raz
erreur	e
e	erreur
raz	raz

TAB. 3.1 – Un système de synchronisation

Le système de synchronisation correspondant a ce tableau est :

– $V = \{(e, inc), (inc, raz), (erreur, e), (e, erreur), (raz, raz)\}$.

La figure 3.12 représente le comportement global des deux variables en tenant compte des contraintes définies dans la table 3.2. Elle est obtenue en effectuant le produit de synchronisation de deux instances du système de transition et du système de synchronisation définis précédemment.

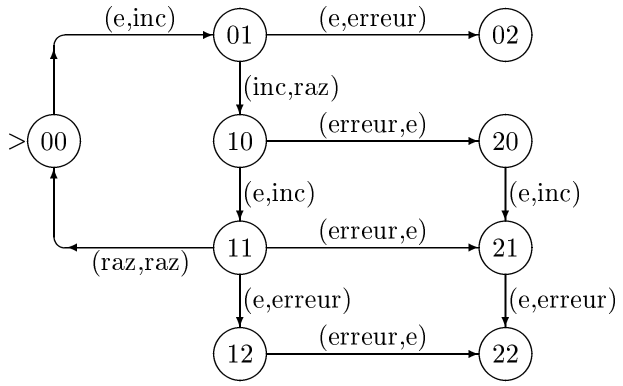

FIG. 3.12 – Un compteur kilométrique obtenu par produit synchronisé

3.2.3 Calcul de comportements complexes

Après avoir modélisé tous les composants par des systèmes de transitions et les interactions entre ces composants par un système de synchronisation :

Question 3.12 *Calculer le système de transition du protocole du bit alterné.*

Corrigé question 3.12

Question 3.13 *Calculer le système de transition de l'algorithme d'exclusion mutuelle de Peterson dans le cas de deux processus.*

Corrigé question 3.13
Le système est composé de P0 et P1, deux systèmes 3.8, de F0 et F1, deux systèmes représentant une variable booléenne, et de Tour, un système représentant une variable entière comprise entre 0 et 1.

Les intéractions décrivent les actions des processus sur les variables.

P0	P1	Tour	F0	F1
FiToT	e	e	toT	e
FiToF	e	e	toF	e
TourToI	e	to0	e	e
TourIsJ	e	is1	e	e
FjIsF	e	e	e	isF
e	FiToT	e	e	toT
e	FiToF	e	e	toF
e	TourToI	to1	e	e
e	TourIsJ	is0	e	e
e	FjIsF	e	isF	e

TAB. 3.2 – Le système de synchronisation pour Peterson

Le produit de synchronisation calcule le graphe de la figure 3.13.

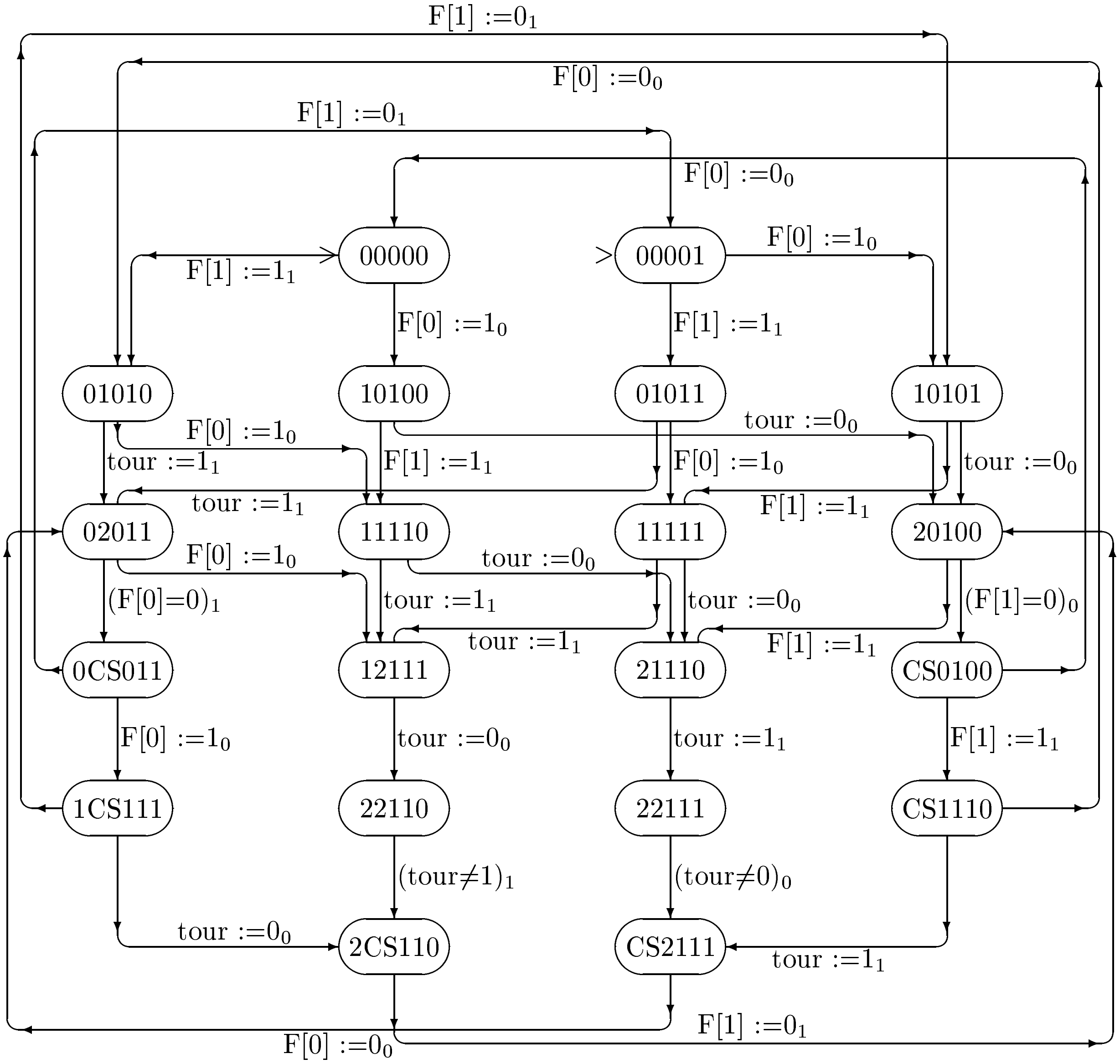

FIG. 3.13 – L'algorithme de Peterson.

3.3 Bilan

- Systèmes de transitions pour algo : correct
- Systèmes de transitions pour données : pénible
- Produit de synchronisation lorsque le nombre de sous-systèmes est grand peut être pénible.

Chapitre 4

Les automates à contraintes

4.1 Idée des automates à contraintes

1. Dans un systèmes de transitions, les états sont nommés.
2. Une variable `nomEtat` et un domaine énuméré de chaînes de caractères.
3. Une variable sur un domaine quelconque.
4. Un vecteur de variables et des contraintes sur les domaines.

4.2 Les automates à contraintes

Definition 4.1 *Un automate à contraintes (AC) est un triplet :*
- *un ensemble de variables $\vec{s}$,*
- *un ensemble d'événements E,*
- *un ensemble de transitions :*

$$G(\vec{s}) \quad \xrightarrow{e} \quad \vec{s} := \sigma(\vec{s})$$

$G(\vec{s})$ est un(e) garde (c-à-d une expression booléenne) et $e \in E$,

4.2.1 Un exemple

Un interrupteur est ouvert ou fermé.

```
variable ouvert : boolean;
evenement    push;
transitions
  ouvert = true |- push -> ouvert := false;
  ouvert = false |- push -> ouvert := true;
```

Une version plus élégante du même automate :

```
variable ouvert : boolean;
evenement    push;
transitions
  true |- push -> ouvert := not ouvert;
```

4.2.2 Avec ou sans post-condition

Soit l'automate à contraintes suivant :

```
variable nb : [0,5];
evenement    plus;
transitions
  true |- plus -> nb := nb + 1;
```

La garde de la transition est en réalité : `(true & nb >= 0 & nb <= 5)` ; d'où la question naturelle : **La transition `(nb=5) |- plus -> nb := 6;` fait-elle partie du modèle ?**

Si la réponse est **oui**, alors le modèle est dit **sans post-conditon**, et si la réponse est **non**, alors le modèle est dit **avec post-condition**.

4.2.3 Élimination de la post-condition

Soit $A1$ un automate à contraintes avec post-condition :$A =< \vec{s}, E, T1 >$, avec $T1$

$$G(\vec{s}) \quad \xrightarrow{e} \quad \vec{s} := \sigma(\vec{s})$$

Notons $D(\vec{s})$, le produit des domaines de définitions des variables.

Posons $A2 =< \vec{s}, E, T2 >$, un système sans post-condition avec $T2$

$$G(\vec{s})\& \; D(\sigma(\vec{s})) \quad \xrightarrow{e} \quad \vec{s} := \sigma(\vec{s})$$

Nous avons $A1 \equiv A2$.

Dans toute la suite, les automates à contraintes sont avec post-condition.

4.3 Les automates à contraintes avec assertion

Definition 4.2 *Un automate à contraintes avec assertion (ACA) est un quadruplet :*
- *un ensemble de variables $\vec{s}$,*
- *un ensemble d'événements E,*
- *un ensemble de transition :*

$$G(\vec{s}) \quad \xrightarrow{e} \quad \vec{s} := \sigma(\vec{s})$$

  $G(\vec{s})$ est un(e) garde (c-à-d une expression booléenne) et $e \in E$,
- *une contrainte globale (ou assertion) $A(\vec{s})$, qui doit toujours être vraie.*

4.3.1 Un exemple

Un interrupteur est ouvert ou fermé. Il possède deux vitesses, mais la fermeture et l'ouverture de l'interrupteur ne sont possibles qu'en vitesse lente.

```
variable ouvert, rapide : boolean;
evenement    push, reguler;
transitions
  true |- push    -> ouvert := not ouvert;
  true |- reguler -> rapide := not rapide;
assertion
  rapide => not ouvert; // not (rapide & ouvert)
```

Propriété 4.1 *Les automates à contraintes avec assertion définis en 4.2 ont la même expressivité que ceux de la définition 4.1.*

Question 4.1 *Démontrez la propriété 4.1.*

Corrigé question 4.1

1. Soit $A1 =< \vec{s}, E, T1, A(\vec{s}) >$ un automate à contraintes avec assertion
   - un ensemble de transition : $G_i(\vec{s}) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$
   - une contrainte globale (ou assertion) $A(\vec{s})$,

   Posons $A2 =< \vec{s}, E, T2 >$ l'automate à contraintes avec $T2$ défini par :

$$A(\sigma_i(\vec{s})) \ \& \ A(\vec{s}) \ \& \ G_i(\vec{s}) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$$

   Nous avons $A1 \equiv A2$.

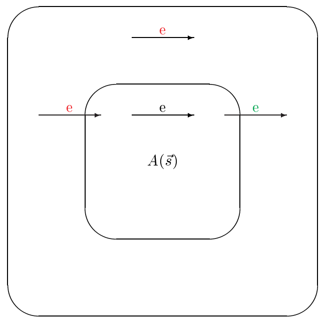

FIG. 4.1 – Schéma pour la preuve de $AC \equiv ACA$

2. Dans l'autre sens, il suffit de prendre l'assertion *true* : $< \vec{s}, E, T > \equiv < \vec{s}, E, T, true >$

Question 4.2 *Appliquez la transformation sur l'exemple.*

Corrigé question 4.2
La transformation donne sur l'exemple précédent :

```
variable ouvert, rapide : boolean;
evenement    push, reguler;
transitions
  rapide     => ouvert &
  rapide     => not ouvert &
  true                        |- push    -> ouvert := not ouvert;
  not rapide => not ouvert &
  rapide     =>  notouvert &
  true                        |- reguler -> rapide := not rapide;
```

qui se simplifie en :

```
variable ouvert, rapide : boolean;
evenement    push, reguler;
transitions
  not rapide |- push    -> ouvert := not ouvert;
  not ouvert |- reguler -> rapide := not rapide;
```

4.4 Les automates à contraintes à priorités

Definition 4.3 *Un automate à contraintes à priorités (ACP) est un quadruplet :*
- *un ensemble de variables $\vec{s}$,*
- *un ensemble d'événements E,*

– *un ensemble de transitions :*

$$G(\vec{s}) \quad \xrightarrow{e} \quad \vec{s} := \sigma(\vec{s})$$

$G(\vec{s})$ est un(e) garde (c-à-d une expression booléenne) et $e \in E$,

– *des priorités (un ordre partiel sur E noté $\prec_E$), qui indique qu'un événement n'est possible que si tous les événements plus prioritaires ne le sont pas.*

4.4.1 Un exemple

```
variable nb1, nb2 : [0,2];
evenement    put1, put2, get1 > get2;
transitions
  nb1<2 |- put1 -> nb1 := nb1 +1;
  nb2<2 |- put2 -> nb2 := nb2 +1;
  nb1>0 |- get1 -> nb1 := nb1 -1;
  nb2>0 |- get2 -> nb2 := nb2 -1;
```

représente un ordonnanceur de files à priorités. On ne peut retirer un élément de la seconde file que si la première est vide.

Propriété 4.2 *Les automates à contraintes à priorités définis en 4.3 ont la même expressivité que ceux de la définition 4.1.*

Question 4.3 *Démontrez la propriété 4.2.*

Corrigé question 4.3

1. Soit $A1 =< \vec{s}, E, T1, \prec_E>$ un automate à contraintes avec priorité avec
   – $T1$ un ensemble de transitions : $G_i(\vec{s}) \ \xrightarrow{e_i} \ \vec{s} := \sigma_i(\vec{s})$
   Posons $A2 =< \vec{s}, E, T2 >$ l'automate à contraintes avec $T2$ défini par :
   $$(\bigwedge_{k \in E}^{i \prec_E k} \neg G_k(\vec{s})) \ \& \ G_i(\vec{s}) \ \xrightarrow{e_i} \ \vec{s} := \sigma_i(\vec{s})$$
   Nous avons $A1 \equiv A2$.
2. Dans l'autre sens, il suffit de prendre l'ordre partiel *vide* : $< \vec{s}, E, T > \equiv < \vec{s}, E, T, >$.

Question 4.4 *Appliquez la transformation sur l'exemple.*

La transformation donne sur l'exemple précédent :

```
variable nb1, nb2 : [0,2];
evenement    put1, put2, get1 > get2;
transitions
  nb1<2 |- put1 -> nb1 := nb1 +1;
  nb2<2 |- put2 -> nb2 := nb2 +1;
  nb1>0 |- get1 -> nb1 := nb1 -1;
  (not nb1>0) & nb2>0 |- get2 -> nb2 := nb2 -1;
```

qui se réduit en :

```
variable nb1, nb2 : [0,2];
evenement    put1, put2, get1 > get2;
transitions
  nb1<2 |- put1 -> nb1 := nb1 +1;
  nb2<2 |- put2 -> nb2 := nb2 +1;
  nb1>0 |- get1 -> nb1 := nb1 -1;
  nb1=0 & nb2>0 |- get2 -> nb2 := nb2 -1;
```

4.5 Les automates à contraintes avec assertion et priorités

Question 4.5 *Montrez l'éventuelle ambiguïté des automates à contraintes avec assertion et priorités.*

Corrigé question 4.5
Soit un automate à contraintes avec assertion globale et priorité.

- un ensemble de transitions : $G(\vec{s}) \xrightarrow{e_i} \vec{s} := \sigma(\vec{s})$
- une contrainte globale (ou assertion) $A(\vec{s})$,
- un ordre partiel $\prec_E$.

1. Première sémantique : on élimine d'abord l'assertion globale, puis les priorités. On obtient l'automate suivant :

$$(\bigwedge_{k\in E}^{i\prec_E k} \neg(A(\sigma_k(\vec{s})) \;\&\; A(\vec{s}) \;\&\; G_k(\vec{s}))) \;\&\; (A(\sigma_i(\vec{s})) \;\&\; A(\vec{s}) \;\&\; G_i(\vec{s})) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$$

que l'on peut simplifier en :

$$(\bigwedge_{k\in E}^{i\prec_E k} \neg(A(\sigma_k(\vec{s})) \;\&\; G_k(\vec{s}))) \;\&\; (A(\sigma_i(\vec{s})) \;\&\; A(\vec{s}) \;\&\; G_i(\vec{s})) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$$

2. Seconde sémantique : on élimine d'abord les priorités, puis l'assertion globale. On obtient l'automate suivant :

$$A(\sigma_i(\vec{s})) \;\&\; A(\vec{s}) \;\&\; (\bigwedge_{k\in E}^{i\prec_E k} \neg G_k(\vec{s})) \;\&\; (G_i(\vec{s})) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$$

Le modèle est bien ambigu car en général

$$(\bigwedge_{k\in E}^{i\prec_E k} \neg(A(\sigma_k(\vec{s})) \;\&\; G_k(\vec{s}))) \neq (\bigwedge_{k\in E}^{i\prec_E k} \neg G_k(\vec{s}))$$

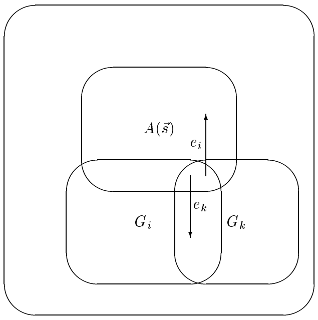

FIG. 4.2 – Schéma montrant l'ambiguïté des *ACAP*

Definition 4.4 *Un automate à contraintes avec assertion et priorités (ACAP) est un quintuplet :*

- *un ensemble de variables $\vec{s}$,*
- *un ensemble d'événements E,*
- *un ensemble de transitions :*

$$G(\vec{s}) \xrightarrow{e} \vec{s} := \sigma(\vec{s})$$

$G(\vec{s})$ est un(e) garde (c-à-d une expression booléenne) et $e \in E$,

- *une contrainte globale (ou assertion) $A(\vec{s})$, qui doit toujours être vraie.*
- *des priorités (un ordre partiel sur E noté $\prec_E$), qui indique qu'un événement n'est possible que si tous les événements plus prioritaires et respectant la post-condition de l'assertion ne le sont pas.*

Question 4.6 *Justifiez le choix de la définition 4.4.*

Corrigé question 4.6

4.6 Le produit d'automates à contraintes

Question 4.7 *Proposez une extension du produit de synchronisation de systèmes de transition pour les automates à contraintes.*

Corrigé question 4.7
Soit deux automates à contraintes A et A'

- deux ensembles de transitions : $G_i(\vec{s}) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$ et $G'_j(\vec{s'}) \xrightarrow{e'_j} \vec{s'} := \sigma'_j(\vec{s'})$
- un ensemble V de vecteur de synchronisation $\{< e_i, e'_j >\}$ inclut dans $E \times E'$.

Sous l'hypothèse $\vec{s} \cap \vec{s'} = \emptyset$, le produit $A \times B$ est défini par l'ensemble de transitions :

$$\begin{array}{lll} < e_i, e'_j > \in V & G_i(\vec{s})\ \&\ G'_i(\vec{s'}) & \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\sigma_i(\vec{s}), \sigma'_j(\vec{s'})) \\ < e_i, e'_j > \notin V & false & \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\vec{s}, \vec{s'}) \end{array}$$

Question 4.8 *Le produit synchronisé de deux systèmes de transition à contraintes avec assertion est-il ambigu ?*

Corrigé question 4.8
Soit deux systèmes de transition à contraintes avec assertion ayant des ensembles $\vec{s}$ et $\vec{s'}$ disjoints.

- deux ensembles de transitions : $G_i(\vec{s}) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$ et $G'_j(\vec{s'}) \xrightarrow{e'_j} \vec{s'} := \sigma'_j(\vec{s'})$
- un ensemble V de vecteur de synchronisation $\{< e_i, e'_j >\}$ inclut dans $E \times E'$.
- deux contraintes globales (ou assertion) $A(\vec{s})$ et $A'(\vec{s'})$.

1. Première sémantique : les assertions sont enlevées, puis le produit est effectué :

$$(A(\sigma_i(\vec{s}))\ \&\ A(\vec{s})\ \&\ G_i(\vec{s}))\ \&\ (A'(\sigma'_j(\vec{s'}))\ \&\ A'(\vec{s'})\ \&\ G'_i(\vec{s'})) \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\sigma_i(\vec{s}), \sigma'_j(\vec{s'}))$$

2. Seconde sémantique : le produit est calculé par conjonction des assertions, puis les assertions sont enlevées.

$$(A(\sigma_i(\vec{s}), \sigma'_j(\vec{s'}))\ \&\ A'(\sigma_i(\vec{s}), \sigma'_j(\vec{s'})))\ \&\ (A(\vec{s})\ \&\ A'(\vec{s'}))\ \&\ (G_i(\vec{s})\ \&\ G'_i(\vec{s'})) \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\sigma_i(\vec{s}), \sigma'_j(\vec{s'}))$$

Du fait que les ensembles de variables sont disjoints, c'est équivalent à :

$$(A(\sigma_i(\vec{s}))\ \&\ A'(\sigma'_j(\vec{s'}))))\ \&\ (A(\vec{s})\ \&\ A'(\vec{s'}))\ \&\ (G_i(\vec{s})\ \&\ G'_i(\vec{s'})) \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\sigma_i(\vec{s}), \sigma'_j(\vec{s'}))$$

Le produit avec assertion n'est donc pas ambigu.

Question 4.9 *Le produit synchronisé de deux systèmes de transition à contraintes avec priorité est-il ambigu ?*

Corrigé question 4.9
Soit deux systèmes de transition à contraintes avec priorité ayant des ensembles $\vec{s}$ et $\vec{s'}$ disjoints.

- deux ensembles de transitions : $G_i(\vec{s}) \xrightarrow{e_i} \vec{s} := \sigma_i(\vec{s})$ et $G'_j(\vec{s'}) \xrightarrow{e'_j} \vec{s'} := \sigma'_j(\vec{s'})$
- un ensemble V de vecteur de synchronisation $\{< e_i, e'_j >\}$ inclut dans $E \times E'$.
- deux ordres partiels $\prec_E$ et $\prec_{E'}$.

1. Première sémantique : les priorités sont enlevées, puis le produit est effectué :

$$(\bigwedge_{k \in E}^{i \prec_E k} \neg G_k(\vec{s})) \;\&\; G_i(\vec{s}) \;\&\; (\bigwedge_{k \in E'}^{j \prec_{E'} k} \neg G'_k(\vec{s'})) \;\&\; G'_j(\vec{s'}) \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\sigma_i(\vec{s}), \sigma'_j(\vec{s'}))$$

2. Seconde sémantique : le produit est calculé, puis les priorités sont enlevées.

$$\underset{\prec_{E \times E'}}{G_i(\vec{s}) \;\&\; G'_i(\vec{s'})} \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\sigma_i(\vec{s}), \sigma'_j(\vec{s'}))$$

Ce qui donne du fait que pour $(i, j) \notin V$ la garde est fausse :

$$(\bigwedge_{(k,k') \in V}^{(i,j) \prec_{E \times E'} (k,k')} \neg(G_k(\vec{s}) \& G'_k(\vec{s'})) \;\&\; G_i(\vec{s}) \;\&\; G'_j(\vec{s'}) \xrightarrow{<e_i, e'_j>} (\vec{s}, \vec{s'}) := (\sigma_i(\vec{s}), \sigma'_j(\vec{s'}))$$

Le produit avec priorité n'est donc pas ambigu si et seulement si il existe un ordre partiel $\prec_{E \times E'}$ tel que :

$$(\bigwedge_{(k,k') \in V}^{(i,j) \prec_{E \times E'} (k,k')} \neg(G_k(\vec{s}) \& G'_k(\vec{s'})) = (\bigwedge_{k \in E}^{i \prec_E k} \neg G_k(\vec{s})) \;\&\; (\bigwedge_{k \in E'}^{j \prec_{E'} k} \neg G'_k(\vec{s'}))$$

Malheureusement, soit $E = \{e_1, e_2\}, E' = \{e'_1, e'_2\}, V = \{e_1, e'_1\}$ et tel que $e_1 \prec_E e_2$ et $e'_1 \prec_{E'} e'_2$. Nous avons $< e_1, e'_1 >$ possible avec la seconde sémantique et impossible avec la première.

Le produit synchronisé de deux systèmes de transition à contraintes avec priorité est donc ambigu.

Question 4.10 *Donnez un argument convaincant pour la première sémantique.*

Corrigé question 4.10
La seconde sémantique fait apparaître des comportements dans le produit qui n'existent pas dans les composants. Les *projections* du produit contiennent des scénarios qui n'existent pas dans les composants.

Definition 4.5

4.7 Le produit hiérarchique d'automates à contraintes

Question 4.11 *Proposez une sémantique (et un algorithme pour la calculer) pour la synchronisation hiérarchique d'automates à contraintes avec assertions et priorités.*

Corrigé question 4.11

1. Assimilez la hiérarchie à un arbre.
2. Parcours en profondeur de l'arbre, traitement au dernier passage dans un sommet
   - Si c'est une feuille, calculez AC en choisissant une des deux sémantiques possibles :
     - $ACAP \longrightarrow ACA \longrightarrow AC$
     - $ACAP \longrightarrow ACP \longrightarrow AC$
   - Si c'est un nœud, calculez $ACAP$ comme le produit synchronisé $ACAP = AC_1 \times AC_2 \times \ldots \times AC_n$, puis utilisez la même sémantique que pour les feuilles.

4.8 Modélisation avec des automates à contraintes

Question 4.12 *Modélisez avec une hiérarchie d'automates à contraintes l'algorithme d'exclusion mutuelle de Peterson*

Corrigé question 4.12

```
Automate Processus
  variable etat : [0,4]; Fi, Fj : bool; tour; i : [0,1];
  evenement run;
  transition
    etat = 0 |- run -> etat := 1; Fi := true;
    etat = 1 |- run -> etat := 2; tour := i;
    Fj & etat = 2 |- run -> etat := 3;
    not Fj & etat = 2 |- run -> etat := 4;
    (tour = i) & etat = 3 |- run -> etat := 2;
    (tour != i) & etat = 3 |- run -> etat := 4;
    etat = 4 |- run -> etat := 0; Fi := false;
    true |- e -> Fj := true; tour := 0;
    true |- e -> Fj := true; tour := 1;
    true |- e -> Fj := false; tour := 0;
    true |- e -> Fj := false; tour := 1;
  assert
    (etat = 0) => (Fi = false);
  init
    etat := 0;

Automate Peterson
  sous-automate P0, P1 : Processus;
  sync <e, run>;
       <run, e>;
  assert
    (P0.Fi = P1.Fj) & (P0.Fj = P1.Fi) & (P0.tour = P1.tour) & (P0.i = 0) & (P1.i = 1);
```

Question 4.13 *Modélisez avec des systèmes de transition à contraintes le protocole du bit alterné.*

Corrigé question 4.13
Une première version :

```
Automate Emetteur
  variable state : [0,5]; parite, pariteAck : bool;
  evenement run, inM, outMP, inMP;
  transition
    etat = 0 |- run -> etat := 1; parite := true;
    etat = 1 |- inM -> etat := 2;
    etat = 2 |- outMP -> etat := 3;
    etat = 3 |- inMP -> etat := 4;
    etat = 4 & (parite != pariteAck) |- run -> etat := 2;
    etat = 4 & (parite = pariteAck) |- run -> etat := 5;
    etat = 5 |- run -> etat := 2; parite := not parite;
    true |- e -> pariteAck := true;
    true |- e -> pariteAck := false;

Automate Recepteur
```

```
  variable state : [0,5]; parite, pariteAck : bool;
  evenement run, outM, outMP, inMP;
  transition
    etat = 0 |- run -> etat := 1; pariteAck := true;
    etat = 1 |- inMP -> etat := 2;
    etat = 2 |- outMP -> etat := 3;
    etat = 3 & (parite != pariteAck) |- run -> etat := 1;
    etat = 3 & (parite = pariteAck) |- run -> etat := 4;
    etat = 4 |- outM -> etat := 5;
    etat = 5 |- run -> etat := 2; pariteAck := not pariteAck;
    true |- e -> parite := true;
    true |- e -> parite := false;

Automate ABP
  sous-automate E : Emetteur;
                R : Recepteur;
  sync <inM, e>;
       <outMP, inMP>;
       <inMP, outMP>;
       <e, outM>;
       <run, e>;
       <e, run>;
  assert
       E.parite = R.parite;
       E.pariteAck = R.pariteAck;
```

Cette version pose problème. Dans un scénario idéal, l'émetteur reçoit l'acquittement attendu, mais avant de faire le test d'égalité, il attend suffisamment longtemps pour que le récepteur modifie `pariteAck`. Le test est alors faux, alors qu'il n'y a eu aucune erreur de transmission.

La seconde version corrige le problème en mémorisant le résultat du test au plus tôt.

```
node Emetteur
  variable state : [0,5]; parite, pariteAck, ok : bool;
  evenement run, inM, outMP, inMP;
  transition
    etat = 0 |- run -> etat := 1; parite := true;
    etat = 1 |- inM -> etat := 2;
    etat = 2 |- outMP -> etat := 3;
    etat = 3 |- inMP -> etat := 4; ok := (parite = pariteAck);
    etat = 4 & (not ok) |- run -> etat := 2;
    etat = 4 & ok |- run -> etat := 5;
    etat = 5 |- run -> etat := 2; parite := not parite;
    true |- e -> pariteAck := true;
    true |- e -> pariteAck := false;
edon

node Recepteur
  variable state : [0,5]; parite, pariteAck, ok : bool;
  evenement run, outM, outMP, inMP;
  transition
    etat = 0 |- run -> etat := 1; pariteAck := true;
    etat = 1 |- inMP -> etat := 2;
    etat = 2 |- outMP -> etat := 3; ok := (parite = pariteAck);
    etat = 3 & (not ok) |- run -> etat := 1;
```

```
    etat = 3 & ok |- run -> etat := 4;
    etat = 4 |- outM -> etat := 5;
    etat = 5 |- run -> etat := 2; pariteAck := not pariteAck;
    true |- e -> parite := true;
    true |- e -> parite := false;
edon

node ABP
  sub E : Emetteur;
      R : Recepteur;
  sync <inM, e>;
       <outMP, inMP>;
       <inMP, outMP>;
       <e, outM>;
       <run, e>;
       <e, run>;
  assert
       E.parite = R.parite;
       E.pariteAck = R.pariteAck;
edon
```

4.9 Les automates à contraintes AltaRica

Definition 4.6 *Un automate à contraintes* AltaRica *est un sextuplet :*

- *un ensemble fini d'automates à contraintes* AltaRica $\vec{AC}$
- *un ensemble fini de variables de d'états* $\vec{s}$*,*
- *un ensemble fini de variables de flux* $\vec{f}$*,*
- *un ensemble fini d'événements* E*,*
- *un ensemble fini de transitions :*

$$G(\vec{s}, \vec{f}, \vec{AC}.f) \quad \xrightarrow{e} \quad \vec{s} := \sigma(\vec{s}, \vec{f}, \vec{AC}.f)$$

  $G(\vec{s}, \vec{f}, \vec{AC}.f)$ *est un(e) garde (c-à-d une expression booléenne) et* $e \in E$*.*
- *une assertion* $A(\vec{s}, \vec{f}, \vec{AC}.f)$
- *un ordre partiel* $\prec_E$ *pour des priorités.*
- *un ensemble fini de vecteur de synchronisation muni d'un ordre partiel. Un vecteur de synchronisation est un élement de* $(E \cup \epsilon) \times \prod(AC.E \cup \epsilon)$*.*

Tous les ensembles finis peuvent être vides. Les feuilles ont notamment un ensemble vide d'automates à contraintes.

La sémantique d'un automate peut-être obtenue par une mise à plat qui produit un automate à contraintes AltaRica dont l'ensemble $\vec{AC}$ est **vide**.

Chapitre 5

Expressivité des automates à contraintes

5.1 Quelques rappels

5.1.1 Identification problème/langage

Un problème est une question *générique* s'appliquant à un ensemble d'éléments (exemple : Déterminer si un nombre naturel est pair). Pour chaque instance du problème, la question admet une réponse (OUI/NON dans les problèmes de décision).

Un langage est un ensemble fini ou infini de mots sur un alphabet σ.

Résoudre un problème, c'est trouver toutes les instances pour lesquelles la réponse est OUI. En codant par des mots toutes les instances, il est facile de se convaincre que résoudre un problème, c'est déterminer un langage des mots.

5.1.2 Identification algorithme/procédure effective

Un algorithme est une procédure réalisable par un ordinateur. Une procédure effective est une procédure *réalisable par une machine*. Une machine devant pouvoir : lire les données d'une instance d'un problème, exécuter des instructions, stocker des résultats intermédiaires dans une mémoire, produire un résultat.

Si la mémoire est finie, la machine est tout simplement un ordinateur, si elle est infinie, c'est une machine un peu plus universelle. Mais dans un algorithme, le fait que la mémoire soit finie ou non n'a pas d'importance. Il est donc naturel d'identifier algorithme et procédure effective.

5.1.3 Machines de Turing

Intuitivement, une machine de Turing est composée de :
- une mémoire infinie, appelée ruban, divisée en case numérotée de 0 à l'inifini ;
- une tête de lecture qui se déplace soit d'une case vers la droite, soit d'une case vers la gauche,
- un ensemble fini d'états dont un état initial et un ensemble F d'états finaux,
- une fonction de transition :

  1. $(q, a) \to (q, b)$ (modification mémoire),
  2. $(q, a) \to q'$ (nouvelle instruction),
  3. $(q, a) \to \mathit{sens} \in \{\mathit{droite}, \mathit{gauche}\}$, (nouvelle entrée).

La fonction de transition et les états constituent le jeu d'instructions.

Pour une instance d'un problème donné, la machine est initialisée ainsi :
- le ruban avec un codage du problème,
- la tête de lecture sur la case 0,

– état initial.

L'exécution peut conduire à :

un arret la fonction de transition n'est pas défini pour le couple (q, a). Deux cas :

1. $q \in F$, le mot (l'instance d'un problème) est accepté.
2. $q \notin F$, le mot (l'instance d'un problème) est rejeté.

une exécution infinie : la machine ne sait pas décider si le mot est accepté ou refusé.

Definition 5.1 *Le langage accepté par une machine de Turing M est l'ensemble des mots acceptés par M.*

Definition 5.2 *Un langage L est décidé par une machine de Turing M ssi :*
- *L est accepté par M,*
- *M n'a pas d'exécution infinie. M s'arrête donc pour toutes les instances du problème.*

5.1.4 Quelques langages indécidables

Definition 5.3 *La classe de décidabilité R (Récursif) est l'ensemble des langages décidables par une machine de Turing.*

Definition 5.4 *La classe de décidabilité RE (Récursivement énumérable) est l'ensemble des langages acceptés par une machine de Turing.*

Remarque : $R \subseteq RE$

Un premier langage indécidable

Soit la matrice suivante dans laquelle :
- les w_j représentent l'ensemble dénombrable des mots finis (les instances de tous les problèmes),
- les M_i représentent l'ensemble dénombrable des machines de Turing (toutes les procédures effectives),
- $A[M_i, w_j = 0$ ssi la machine M accepte le mot w_j. ($A[M_i, w_j] = N$ signifie que soit M_i rejete w_j, soit M_i a une exécution infinie pour w_j)

A	w_0	w_1	$\dots$	w_j	$\dots$
M_0	O	N		O	
M_1	N	N		O	
$\vdots$					
M_i	N	O		O	
$\vdots$					

Considérons le langage de la diagonale $L_0 = \{w \text{ tel que } w = w_i \text{ et } A[M_i, w_j] = N\}$.

Supposons $L_0 \in RE$

$$\begin{aligned} L_0 \in RE \quad &\Rightarrow \quad \exists M_k \text{ accepte } L_0 \\ &\Rightarrow \quad \exists M_k \text{ telle que } A[M_k, w_j] = 0 \text{ ssi } A[M_j, w_j] = N \\ &\Rightarrow \quad \exists M_k \text{ telle que } A[M_k, w_k] = 0 \text{ ssi } A[M_k, w_k] = N \end{aligned}$$

Il y a contradiction, donc $L_0 \notin RE$,

Propriété 5.1 *Il existe des langages non décidables, donc des problèmes pour lesquels il n'existe pas de procédure effective.*

Un deuxième langage indécidable

Propriété 5.2

$$L \in R \Rightarrow \bar{L} \in R$$

La machine $\bar{M}$ est obtenue de M en inversant les réponses.

Propriété 5.3

$$\left\{ \begin{array}{l} L \in RE \\ \bar{L} \in RE \end{array} \right. \Rightarrow \left\{ \begin{array}{l} L \in R \\ \bar{L} \in R \end{array} \right.$$

Soit la machine M' obtenue de $M \parallel \bar{M}$, avec $A[M', w] = 0$ si $A[M, w] = 0$ et $A[M', w] = N$ si $A[\bar{M}, w] = 0$.

M' s'arrête pour tous les mots, car soit M, soit $\bar{M}$ s'arrête, et M'' accepte L, donc $L \in R$, et par le lemme précédent $\bar{L} \in R$.

Propriété 5.4 *Pour tous langage L, soit :*

1. $L \in R, \bar{L} \in R$
2. $L \notin RE, \bar{L} \notin RE$
3. $L \notin RE, \bar{L} \in (RE \cap \bar{R})$

C'est une conséquence immédiate des deux propriétés précédentes.

Propriété 5.5 *Il existe des langages non décidables dans $RE \cap \bar{R}$.*

Considérons le langage de la diagonale $L_0 = \{w \text{ tel que } w = w_i \text{ et } A[M_i, w_j] = N\}$, et son complémentaire $\bar{L_0} = \{w \text{ tel que } w = w_i \text{ et } A[M_i, w_j] = 0\}$

Soit un mot w. Construisons la machine M suivante :

1. Trouver i tel que $w = w_i$ (énumération finie).
2. Construire la machine Mi.
3. Poser $A[M, w] = A[M_i, w_i]$

La machine M accepte donc w ssi M_i accepte w_i, on en déduit : M accepte $\bar{L_0}$, puis $\bar{L_0} \in RE$, puis $\bar{L_0} \in (RE \cap \bar{R})$.

Remarque : Les langages de $RE \cap \bar{R}$ sont interressants car *presque décidables*.

5.1.5 La technique de la réduction

Le principe général de la réduction d'un problème à un autre est le suivant :

1. Soit une suite strictement croissante d'ensemble $C_0 \subset C_1 \subset \ldots \subset C_n$,
2. Pour montrer qu'un objet X est dans $\bar{C_i}$, on choisit un objet C connu pour être dans $C_{i+1} \cap \bar{C_i}$, puis on montre $X \in C_i \Rightarrow C \in C_i$,
3. On conclut que $X \in \bar{C_i}$.

On a réduit le problème de $C \in C_i$ au problème de $X \in C_i$.

Les problèmes d'indécidabilité sont très souvent démontrés à l'aide de la technique de réduction. Ainsi, pour montrer qu'un langage L_2 est indécidable ($L_2 \in \bar{R}$) :

1. On considère $R \subset RE \subset \{L\}$,
2. On choisit un langage $L_1 \in (RE \cap \bar{R})$, (ex $\bar{L_0}$)
3. On montre $L_2 \in R \Rightarrow L_1 \in R$. Généralement cela consiste à construire une machine de Turing M_1 décidant L_1 à partir de la machine de Turing M_2 qui décide L_2.
4. On conclut que $L_2 \in \bar{R}$, donc que L_2 est indécidable.

Indécidabilité du langage universel

Soit le langage $LU = \{< M, w > \text{ tel que} M \text{accepte} w\}$, montrons que LU est indécidable.

1. Supposons l'existence de M_{LU} une machine de Turing qui décide LU,
2. Soit un mot w ; construisons M ainsi :
   (a) Trouvez l'indice i tel que $w = w_i$.
   (b) Construire la machine M_i.
   (c) Appliquer M_{LU} à $< M_i, w_i >$.
   (d) Poser $A[M, w] = A[M_{LU}, < M_i, w_i >]$
3. $M_{LU} \in R$, donc le calcul termine toujours, donc $M \in R$.
4. Mais le langage accepté par M est $\bar{L_0}$, il y a donc contradiction avec le fait que $\bar{L_0}$ est indécidable.
5. Il n'existe donc pas de machine de Turing qui décide LU,

Le langage LU est indécidable. En fait par une démonstration analoque à celle faite pour L_0, on peut montrer que $LU \in (RE \cap \bar{R})$.

Indécidabilité de l'arrêt d'une machine de Turing

Soit le langage $H = \{< M, w > \text{ tel que} M \text{s'arrête sur} w\}$, montrons que H est indécidable.

1. Supposons l'existence de M_H une machine de Turing qui décide H,
2. Soit un mot $< M, w >$; construisons M' ainsi :
   (a) Appliquer M_H à $< M, w >$.
   (b) Poser $A[M', w] =$:
      – NON si $A[M_H, < M, w >= NON]$
      – $A[M, w]$ si $A[M_H, < M, w >= OUI]$
3. $M_H \in R$, et on n'utilise la machine M que sur les mots pour lesquels elle termine, donc le calcul termine toujours, donc $M' \in R$.
4. Mais le langage accepté par M' est LU, il y a donc contradiction avec le fait que LU est indécidable.
5. Il n'existe donc pas de machine de Turing qui décide H,

Le langage H est donc indécidable.

5.2 Les problèmes de décision pour la vérification

Les systèmes de transitions étiquetés (STE) et les automates à contraintes sont classés en fonction de la décidabilité de certaines questions.

L'accessibilité : l'ensembles des triplets (STE, s, s'), où $s \in STE, s' \in STE$, et $s \xrightarrow{*}_{STE} s'$.

Borné : l'ensembles des couples (STE, s), où $s \in STE$, et $\{s' \in STE, s \xrightarrow{*}_{STE} s'\}$ est fini.

Blocage : l'ensembles des couples (STE, s), où $s \in STE$, et $\exists s' \in STE$ tel que $\{s'' \in STE, s' \xrightarrow{*}_{STE} s''\} = \emptyset$.

Model-Checking pour une classe Φ de formules : l'ensembles des triplets (STE, s, ϕ), où $s \in STE, \phi \in \Phi$, et $(STE, s) \models \phi$.

5.2.1 Quelques résultats

1. Le problème du model-checking est décidable pour les STE finis.
2. Le problème du model-checking est indécidable pour les systèmes de transition à contraintes.
3. Le problème de l'accessibilité est indécidable pour les systèmes de transition à contraintes.

Les recherches ont abouti à la définition de classes pour lesquelles certains problèmes sont décidables et d'autres indécidables.

5.3 Quelques classes d'automates à contraintes particulieres

5.3.1 Les systèmes finis

Une restiction consiste à prendre toutes les variables du système à valeurs dans des domaines finis.

Definition 5.5 *Un automate à contraintes fini est un triplet :*

- *un ensemble de variables* **entières bornées $\vec{s}$ fini**
- *un ensemble d'événements E,*
- *un ensemble de transition* **fini** *:*

$$G(\vec{s}) \quad \xrightarrow{e} \quad \vec{s} := \sigma(\vec{s})$$

$G(\vec{s})$ est un(e) garde (c-à-d une expression booléenne) et $e \in E$, et toutes les opérations sont permises dans σ.

Cette restriction est importante. De nombreux travaux ont regardé les systèmes dans lesquels les variables représentent les objets les plus utilisés en contrôle de systèmes : les compteurs et les buffers.

5.3.2 Les systèmes à compteurs

Definition 5.6 *Un automate à contraintes à compteurs est un quadruplet :*

- *un ensemble de variables de contrôle* **entières bornées $\vec{s}$ fini**
- *un ensemble de compteurs* **entiers $\vec{c}$ fini**
- *un ensemble d'événements E,*
- *un ensemble de transition* **fini** *:*

$$G(\vec{s,c}) \quad \xrightarrow{e} \quad \vec{s,c} := \sigma(\vec{s,c})$$

$G(\vec{s,c})$ est un(e) garde (c-à-d une expression booléenne) et $e \in E$, et les opérations permises dans σ sont $\{assign(s, cste),\ add(c, v),\ test(c, 0),\ reset(c),\ transfert(c, c')\}$

Quelques résultats pour les automates à contraintes à compteurs :

1. si $\sigma \in \{add(c, v)\}$, l'accessibilite est décidable et la logique LTL est décidable.
2. si $\sigma \in \{add(c, v)\}$, et que le nombre de compteurs est au maximum 2, l'accessibilite est calculable.
3. si $\sigma \in \{add(c, v),\ test(c, 0)\}$, la logique LTL n'est pas décidable.
4. ...

5.3.3 Les systèmes à files

Definition 5.7 *Un automate à contraintes à files est un quintuplet :*

- *un ensemble de variables de contrôle* **entières bornées $\vec{s}$ fini**
- *un ensemble de canaux $\vec{c}$* **fini**
- *un ensemble d'événements E,*
- **un ensemble fini de messages** *M.*
- *un ensemble de transition* **fini** *:*

$$G(\vec{s}) \quad \xrightarrow{e} \quad \vec{s,c} := \sigma(\vec{s,c})$$

$G(\vec{s})$ est un(e) garde (c-à-d une expression booléenne) et $e \in E$, et les opérations permises dans σ sont $\{assign(s, cste),\ c?m,\ c!m\}$, où $m \in M$ et c est une variable représentant un canal.

Quelques résultats pour les automates à contraintes à files :

1. l'accessibilité est indécidable.
2. l'accessibilité avec un seul canal est indécidable.
3. ...

Question 5.1 *Démontrez que dans les automates à files, l'accessibilité avec un seul canal est indécidable.*