Spécifications et vérifications formelles

Alain Griffault
Université Bordeaux I

17 mars 1998

Première partie

Cours & Travaux dirigés

Chapitre 1

La modélisation de systèmes complexes

1.1 Etude de programmes parallèles

1.1.1 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 1.1.1 *Quelles sont les valeurs possibles de la variable* `compte` *après les exécutions des deux processus A et B ?*

Question 1.1.2 *Cherchez une méthode simple qui permette aisément de montrer que votre réponse à la question précédente est correcte ?*

1.1.2 Communication par messages

Soient les deux processus suivant.

Processus P(i)
/* i de 0 a 1 */

begin
while true do
begin
^{0} *F(i) ! prendre ;*
^{1} F(1-i) ! prendre ;
^{2} *F(1-i) ! poser ;*
^{3} F(i) ! poser ;
end ;
end.

Processus F(i)
/* i de 0 a 1 */

begin
while true do
begin
^{L} *P(0) ? prendre* $\rightarrow$ ^{0} P(0) ? poser ;
or
^{L} *P(1) ? prendre* $\rightarrow$ ^{1} P(1) ? poser ;
end ;
end.

Question 1.1.3 *Appliquez la méthode précédente pour mettre en évidence le problème principal de ce programme.*

1.1.3 Analyse de la méthode

Essayez de répondre aux questions suivantes :

Question 1.1.4 *A quoi correspond les sommets d'un tel graphe ?*

Question 1.1.5 *A quoi correspond les arrêtes d'un tel graphe ?*

Question 1.1.6 *A quoi correspond un sommet duquel part aucune transition ?*

1.2 Le modèle Arnold-Nivat

Dans ce modèle, le comportement d'un ensemble de processus est décrit par un système de transition.

Un système de transition est un quadruplet $S =< Q, I, A, T >$ où :

- Q est un ensemble fini d'états.
- I est le sous-ensemble des états *initiaux*.
- A est un ensemble fini d'actions ou alphabet.
- $T \subseteq Q \times A \times Q$ est un ensemble de transitions.

Remarque : Soit le processus suivant :

```
Processus A
var aux : integer
begin
aux := 0;
while true do
        aux := aux + 1;
end.
```

Un système de transition qui représente la valeur de la variable est un système de transition inifini. Dans notre modèle, ces systèmes sont interdits, il faut nécessairement effectuer une abstraction de la valeur.

1.3 Modélisation par systèmes de transition

1.3.1 Modélisation des données

Dessiner les systèmes de transition des objets suivants :

Question 1.3.1 *Une variable entière comprise entre 0 et 3*

- *les états ;*
- *l'opération d'affectation ;*
- *l'opération de lecture*

Question 1.3.2 *Un buffer de taille 1.*

Question 1.3.3 *Une file d'attente de taille 2.*

Question 1.3.4 *Une pile d'attente de taille 2.*

1.3.2 Modélisation de comportements simples

Dessiner les systèmes de transition des objets suivants :

Question 1.3.5 *Un contrôleur d'une salle limitée à dix places.*

Question 1.3.6 *Un contrôleur de bibliothèque dans laquelle un écrivain est nécessairement seul, et ou les lecteurs peuvent se cotoyer. La première personne à accéder à la bibliothèque est un écrivain.*

Question 1.3.7 *Une porte logique* **ET**.

Question 1.3.8 *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;
```

Question 1.3.9 *L'émetteur dans le protocole du bit alterné.*

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

Question 1.3.10 *le récepteur dans le protocole du bit alterné.*

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

1.3.3 Modélisation de comportements complexes

Question 1.3.11 *Essayer de dessiner le système de transition de l'algorithme d'exclusion mutuelle de Peterson dans le cas de deux processus.*

1.4 Le modèle Arnold-Nivat (complément)

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 intéractions. 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 :

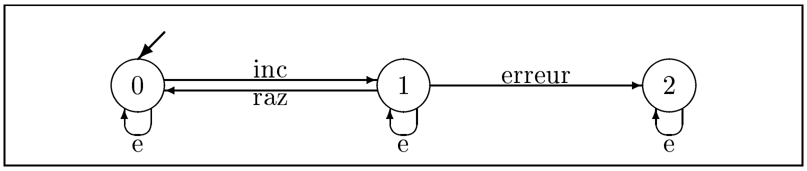

FIG. 1.1: Système de transition d'une variable

- $Q = \prod_{i=1}^{i=n} Q_i$.
- $I = \prod_{i=1}^{i=n} I_i$.
- $A = \prod_{i=1}^{i=n} A_i$.
- $T = T_l \cap Q \times V \times Q$ 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.

1.4.1 Un exemple

La figure 1.1 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 phénomenes asynchrones dans ce modèle fondamentalement synchrone.

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 1.1. La table 1.1 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 phénomêne d'entrainement 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 remize à zéro `raz` sur la deuxième composante du système.

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

TAB. 1.1: 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 1.2 représente le comportement global des deux variables en tenant compte des contraintes définies dans la table 1.1. Elle est obtenue en effectuant le produit de synchronisation de deux instances du système de transition et du système de synchronisation définis précédemment.

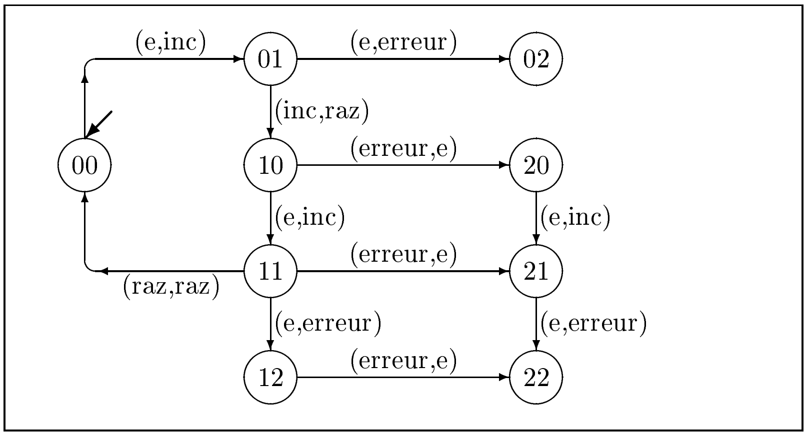

FIG. 1.2: Le produit synchronisÝ reprÝsentant un compteur kilomÝtrique

1.5 Calcul de comportements complexes

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

Question 1.5.1 *Calculer le système de transition du protocole du bit alterné.*

Question 1.5.2 *Calculer le système de transition de l'algorithme d'exclusion mutuelle de Peterson dans le cas de deux processus.*

Chapitre 2

La spécification de systèmes complexes

2.1 Vérification de propriétés de systèmes de transition

2.1.1 Les marques d'états et de transitions

Une façon simple et naturelle d'exprimer les propriétés d'un système de transition est de fournir comme résultat du calcul, les états ou les transitions qui confirment ou infirment cette propriété. A cette fin, le modèle des systèmes de transition est étendu au modèle des systèmes de transition marqués.
Une marque d'état (resp de transition) est un sous-ensemble des états (resp des transitions).

2.1.2 Les fonctions de base d'une logique

Les marques élémentaires

{} : l'ensemble vide est une marque soit d'état, soit de transition.

***** : l'ensemble total est une marque soit d'état, soit de transition.

initial : les états initiaux d'un système de transition

!label *op* "chaine-de-caracteres" : marque de transition.
avec *op* un opérateur parmi : {=, #, <, <=, >, >=}.

!state *op* "chaine-de-caracteres" : marque d'état.

!label[i] *op* "chaine-de-caracteres" : marque de transition.

!state[i] *op* "chaine-de-caracteres" : marque d'état.

marque-transition[i] : marque de transition.

marque-etat[i] : marque d'état.

Les opérateurs ensemblistes sur les marques

union : A \/ B, avec A et B des marques soit d'état, soit de transition.

intersection : A /\ B, avec A et B des marques soit d'état, soit de transition.

différence : A - B, avec A et B des marques soit d'état, soit de transition.

Les fonctions relatives à un ensemble de transitions

src : les états source d'une marque de transition.

tgt : les états but d'une marque de transition.

rsrc : fonction *inverse* de src.
Attention src(rsrc(S)) = S, mais rsrc(src(T)) $\subseteq$ T.

rtgt : fonction *inverse* de tgt.
Attention tgt(rtgt(S)) = S, mais rtgt(tgt(T)) $\subseteq$ T.

Quelques fonctions pratiques

succ : Cette expresssion calcule les successeurs d'un ensemble d'états par un ensemble de transitions.
succ(S,T) = tgt(rsrc(S) /\ T).

pred : Cette expresssion calcule les prédécesseurs d'un ensemble d'états par un ensemble de transitions.
pred(S,T) = src(rtgt(S) /\ T).

reach : Cette expresssion calcule les successeurs d'un ensemble d'états par un ensemble de chemins.
reach(S,T) = succ*(S,T).

coreach : Cette expresssion calcule les prédécesseurs d'un ensemble d'états par un ensemble de chemins.
coreach(S,T) = pred*(S,T).

proj_state[i] : Projection d'une marque d'état sur le i^{eme} composant.

proj_trans[i] : Projection d'une marque de transition sur le i^{eme} composant.

2.1.3 Quelques propriétés simples usuelles

Exprimer les propriétes suivantes en terme de marque d'état ou de transition.

Question 2.1.1 *Le système risque t-il de se bloquer ?*

Question 2.1.2 *Tous les états du i^{eme} composant sont-ils utilisés ?*

Question 2.1.3 *Toutes les transitions du i^{eme} composant sont-elles utilisées ?*

Question 2.1.4 *Une marque de transition Y, lorqu'elle est possible dans le i^{eme} composant, est-elle toujours possible dans le système global ?*

2.2 Le model-checker MEC

2.2.1 Description fonctionnelle

Mec est un outil qui manipule :

- Les systèmes de transition avec marques.
- Les systèmes de synchronisation.
- Le produit de synchronisation.
- Les fonctions prédéfinies.
- Les fonctions définies par l'utilisateur.
- Des fonctions d'affichage et d'archivage.

2.2.2 Une séance type d'utilisation

Il est préférable d'enregistrer dans un premier temps dans des fichiers les modélisations des systèmes de transition et du système de synchronisation. Puis au prompt de l'outil mec :

1. load("NomDuFichierContenantSystemesTransition");
2. load("NomDuFichierContenantSystemeSynchronisation");
3. sync(NomSystemeSynchronisation, NomSystemeTransitionResultat);
4. dts(NomSystemeTransitionResultat);
5. dead := * - src(*);

6. wts(dead,*) ;
7. D'autres calculs de propriétés.
8. wts(PetiteMarqueEtat, *) ;
9. wts(src(PetiteMarqueTransition), PetiteMarqueTransition) ;
10. L'enregistrement de certains résultats par :
    – ats(NomFichierArchive, MarqueEtat, MarqueTransition, NomSystemeTransitionResultat) ;
    – l'enregistrement d'une session :
      (a) log(NomFichierArchive) ;
      (b) Liste de commandes
      (c) stoplog() ;
11. quit() ;

Il est évidememnt possible de mettre la suite des calculs dans un fichier, puis d'effectuer un "load" de ce fichier.

2.3 Modélisation par systèmes de transition et de synchronisation

2.3.1 Modélisation de comportements complexes

Question 2.3.1 *Calculer par le produit de synchronisation et avec l'outil mec les systèmes de transition des objets suivants :*

1. *Le protocole du bit alterné.*
2. *le circuit logique :*

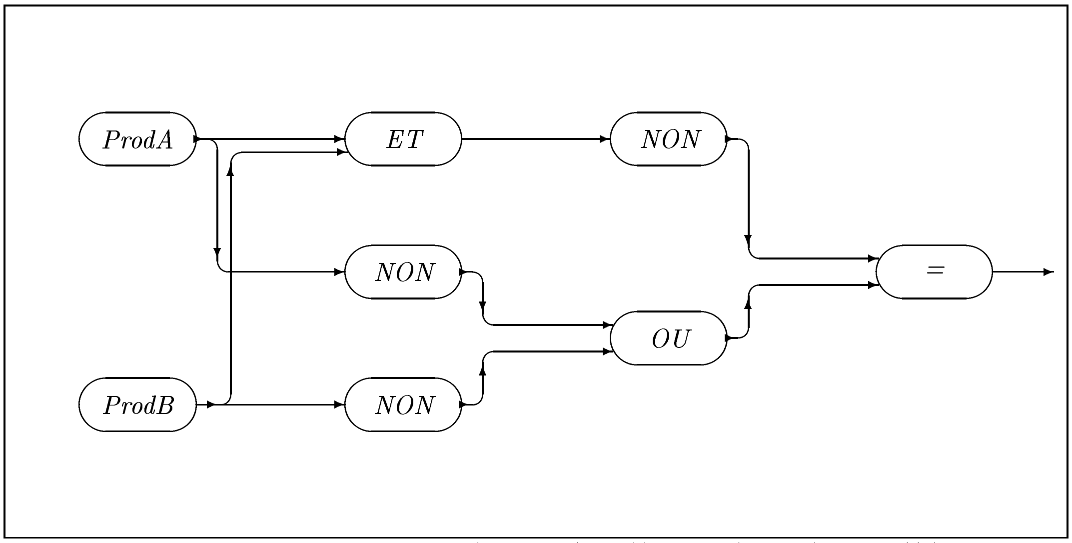

Figure 2.1: Le circuit NON (A ET B) = ((NON A) OU (NON B))(

3. *l'algorithme d'exclusion mutuelle de Peterson dans le cas de deux processus.*
4. *le jeu de Nim : Soit un nombre quelconque de rangées ayant chacune un nombre quelconque d'allumettes. Un joueur peut retirer autant d'allumettes qu'il le désire sur une même rangée. Les joueurs jouent à tour de rôle et le gagnant est celui qui retire la dernière allumette.*
5. *le jeu de Tic-Tac-Toe : Sur un damier, chaque joueur dispose à tour de rôle des jetons de couleur distincte jusqu'à ce que l'un d'entre eux en aligne trois.*

2.4 Vérification de propriétés de systèmes de transition

2.4.1 Des propriétés spécifiques

Question 2.4.1 *Exprimer les propriétes suivantes en terme de marque d'état ou de transition.*

1. *Problème de protocoles*
   - *le protocole est il correct avec un médium parfait.*
   - *le protocole resiste-il aux pertes.*
   - *le protocole resiste-il aux duplications.*
   - *le protocole resiste-il aux déséquencements.*
   - *le protocole resiste-il aux erreurs de transmissions.*
2. *Problème de circuits logiques*
   - *Deux circuits sont-ils équivalents.*
3. *Problème d'exclusion mutuelle*
   - *l'exclusion est-elle respectée.*
   - *l'absence de famine est-elle verifiée.*
   - *l'équité est-elle verifiée.*
4. *Problème de jeux*
   - *Existe t-il des stratégies gagnantes pour des jeux à plusieurs joueurs.*

Chapitre 3

Fonctions point-fixe et équité

3.1 Les fonctions définies par l'utilisateur

3.1.1 Rappels sur les monotonies

Un système d'équations au point fixe, lorsqu'il respecte certaines règles de monotonie, possède une solution. Cette règle syntaxique est implémentée dans Mec.
Les systèmes suivants qui définissent les états `pair, impair et nonimpair` vérifient la règle de monotonie :

```
function pair(S : state) return Z : state;
var pair : state;
    impair : state
begin
pair = S \/ tgt(rsrc(impair);
impair = tgt(rsrc(pair));
Z = pair
end.
function impair(S : state) return Z : state;
var pair : state;
    impair : state
begin
pair = S \/ tgt(rsrc(impair);
impair = tgt(rsrc(pair));
Z = impair
end.
function nonimpair(S : state) return Z : state;
var pair : state;
    nonimpair : _state
begin
pair = S \/ tgt(rsrc(* - nonimpair);
nonimpair = * - tgt(rsrc(pair));
Z = nonimpair
end.
```

Par contre, les exemples suivants ne respectent pas la règle.

```
function incorrecte1(S : state) return Z : state;
var pair : state;
    nonimpair : state
begin
pair = S \/ tgt(rsrc(* - nonimpair);
nonimpair = * - tgt(rsrc(pair));
```

```
Z = nonimpair
end.
function incorrecte2(S : state) return Z : state;
var pair : state ou _state;
    nonimpair : state ou _state
begin
pair = S \/ tgt(rsrc(* - nonimpair);
nonimpair = tgt(rsrc(pair));
Z = nonimpair
end.
```

3.1.2 Des exemples de fonctions

Question 3.1.1 *Definissez à l'aide de système d'équations au point fixe les fonctions suivantes :*

1. *La fonction* `inevitable` *qui calcule les états* `Z` *à partir desquels il est inévitable d'aboutir dans* `X` *si l'on n'utilise que des transitions appartenant à* `Y`.
2. *La fonction* `perte` *qui calcule les transitions appartenant à un chemin allant d'une émission à une autre sans réception.*
3. *La fonction* `duplique` *qui calcule les transitions appartenant à un chemin allant d'une émission à une autre avec duplication de réception.*
4. *La fonction* `last_lost2` *qui calcule les états gagnant dans un jeu à deux joueurs et dans lequel les états puits sont gagnant pour celui qui doit jouer.*
5. *La fonction* `last_wint2` *qui calcule les états gagnant dans un jeu à deux joueurs et dans lequel les états puits sont perdant pour celui qui doit jouer.*
6. *La fonction* `last_lost3` *qui calcule les états gagnant dans un jeu à trois joueurs et dans lequel les états puits sont gagnant pour celui qui doit jouer.*

Chapitre 4

Méthodologie d'utilisation de Mec

4.1 Méthodologie de modélisation, vérification et validation

4.1.1 Définitions

La **modélisation** consiste à définir un modèle abstrait.
La **validation** consiste à *montrer* que le modèle abstrait est un bon candidat pour représenter le monde réel.
La **vérification** consiste à *prouver* que le modèle abstrait *donc certainement le monde réel* vérifie les spécifications.

4.1.2 Processus de travail

1. Concevoir une premier modèle avec beaucoup d'abstractions.
2. Valider le modèle.
3. Itérer 1 et 2 en augmentant la liste des hypothèses,
   jusqu'à ce que la validation soit satisfaisante.
4. Vérifier le modèle
5. Itérer 1, 2, 3 et 4 en augmentant la liste des hypothèses,
   jusqu'à ce que la vérification soit satisfaisante
6. Itérer 1, 2, 3, 4 et 5 en :
   - raffinant le modèle.
   - augmentant la liste des propriétés vérifiées.

   jusqu'à ce que l'une des conditions suivantes soit vraie :
   - Le temps imparti pour la spécification est terminé.
   - Le modèle est une implémentation.
   - Le système de transition global est trop important pour la machine utilisée.

Dans cette démarche, l'introduction d'une nouvelle hypothèse et/ou contrainte dans la description informelle doit renseigner obligatoirement les points suivants :

1. Une description en langage informel de la nouvelle contrainte en la justifiant si possible.
2. Une description formelle des différentes expressions possibles de la contrainte.
3. Une indication s'il s'agit d'une contrainte matérielle ou logicielle, et qui devra la traiter, après en avoir éventuellemnt référé au donneur d'ordres.
4. Le choix utilisé pour la suite de l'étude.

L'intérêt d'une telle démarche est : d'une part éviter la vérification d'un modèle non validé auparavant ; d'autre part permettre une relecture aisée de la description formelle. En pratique, cela évite de considérer des modèles non implémentables, ou non réalisables comme corrects.

La validation du modèle

Des validations syntaxiques et/ou sémantiques

Ces calculs doivent être effectués en priorité car les erreurs détectées indiquent soit des erreurs d'édition lors de la saisie du modèle, soit des erreurs de conception grâves.

- Tous les états décrits sont-ils accessibles ?
- Toutes les transitions décrites sont-elles utilisées ?
- Tous les vecteurs de synchronisations décrits sont-il utilisés ?

L'idée intuitive de cette étape est la suivante : De même qu'un développeur de logiciel n'écrit pas dans un programme une fonction qui ne serait jamais appelée, le modélisateur ne décrit certainement pas des transitions, ou des intéractions qui ne servent jamais dans le produit de synchronisation. Cette vérification purement syntaxique est en quelque sorte un garde fou.

Les propriétés simples et complexes usuelles Les calculs suivants étant d'une part très généraux; d'autre part peu coûteux en temps, doivent être effectués pour toute modélisation. L'interprétation des résultats dépend à la fois de l'application, et de la modélisation.

1. le blocage.
2. le blocage retardé, (inévitable).
3. la possibilité de ne pas revenir dans l'état initial.
4. l'impossibilité de revenir dans l'état initial.
5. le respect des événements incontrôlables. L'environnement d'un système réactif est généralement composé d'événements incontrôlables. Par définition, l'occurrence d'un tel événement est imprévisible, il faut donc veiller à ce qu'à tout instant, le système soit capable de le recevoir (cf paragraphe **??**, page **??**).

La vérification du modèle

Dans cette partie, les propriétés vérifiées correspondent aux spécifications du cahier des charges. Elles sont très liées à la nature du problème, et c'est le donneur d'ordre qui définit l'ensemble des propriétés que doit satisfaire le modèle pour qu'il soit déclaré correct. Par exemple, un concepteur de protocoles souhaite que le protocole resiste aux pertes, aux duplications ou encore aux déséquencements des messages ; un concepteur d'algorithmes distribués désire une solution autostabilisante pour un problème d'élection.
Le régulateur de vitesse est un système de type contrôle-commande, les propriétes généralement souhaitées pour de tels systèmes sont :

- une commande peut-elle être sans effet ?
- une commande peut-elle engendrer des effets indésirés ?
- la réponse à une commande peut-elle être retardée indéfiniment ?

Deuxième partie

Problèmes

Chapitre 5

Etude d'un système réactif de type contrôle commande

5.1 Spécifications informelles d'un régulateur de vitesse

Cet exercice est inspiré de l'article [?].

5.1.1 Spécifications des besoins

La vitessse de croisière d'un véhicule est contrôlée par logiciel. Pour le contrôleur du sytème, nous distinguons quatre états :

OFF : Le contact n'est pas mis.

INACTIVE : Le contact est mis, mais la vitesse n'est pas controlée.

CRUISE : Le contact est mis, et la vitesse est effectivement controlée.

OVERRIDE : Le contact est mis, la vitesse est controlée, mais le contrôle est momentanément suspendu.

L'état initial du contrôleur est **OFF**.

Les changements d'états du contrôleur sont liés à des actions du conducteur, mais également à des événements générés par des capteurs. Toutes ces informations sont véhiculées au contrôleur par l'intermédiaire de signaux électriques.

Le système est donc de type réactif, les changements d'états sont synchronisés d'une part aux fronts montants et descendants des signaux ; d'autre part à la validité de conditions.

Nous introduisons, pour un signal donné, les notations suivantes :

toT : Le front montant.

toF : Le front descendant.

isT : Le signal reste haut.

isF : Le signal reste bas.

Le conducteur peut agir sur le contrôleur de différentes manières :

- En éteignant et démarrant le véhicule.
- En enclanchant une vitesse.
- En appuyant et relâchant la pédale d'accélérateur.
- En appuyant et relâchant la pédale de freins.
- En appuyant sur une des trois boutons à sa disposition :
  - Activation.

- Désactivation.
- Reprise.

Avec ces notations, la table suivante décrit les changements d'états du contrôleur.

Les changements d'états du contrôleur								
Etat	Contact	Roule	Vitesse élevéé	Freinage	Activation	Désactivation	reprise	Etat
OFF	toT							INACTIVE
INACTIVE	toF							OFF
INACTIVE	isT	isT		isF	toT			CRUISE
CRUISE	toF							OFF
CRUISE		toF						INACTIVE
CRUISE			toT					INACTIVE
CRUISE				toT				OVERRIDE
CRUISE						toT		OVERRIDE
OVERRIDE	toF							OFF
OVERRIDE		toF						INACTIVE
OVERRIDE	isT	isT		isF	toT			CRUISE
OVERRIDE	isT	isT		isF			toT	CRUISE

5.1.2 Spécifications de sureté

Le contrôleur précédemment décrit ne sera implémenté que lorsque les propriétés suivantes auront étées prouvées :

1. Lorsque le contrôleur est dans l'état **OFF**, le contact n'est pas mis.
2. Lorsque le contrôleur n'est pas dans l'état **OFF**, le contact est mis.
3. Lorsque le contrôleur est dans l'état **INACTIVE**, le contact est mis et soit la voiture est arrêtée, soit le contrôleur n'est pas activé.
4. Lorsque le contrôleur est dans l'état **CRUISE**, le contact est mis, la voiture roule, et le contrôleur n'appuye pas sur la pédale de frein.
5. Lorsque le contrôleur est dans l'état **OVERRIDE**, le contact est mis et la voiture roule.

De plus, afin de pouvoir commercialiser ce contrôleur, toute propriété prouvée et garantissant un *meilleur* fonctionnement est intéressante.

Question 5.1.1 *Vous devez modéliser ce cahier des charges, en interprétant le moins possible, mais en ajoutant suffisement de contraintes afin que les propriétés de sureté soit vérifiées.*

Chapitre 6

Etude d'un système réactif temps-réel

6.1 Spécifications informelles d'un contrôleur de chaudière

Cet exercice est inspiré de l'article [?].

6.1.1 Spécifications des besoins

Le niveau d'eau dans la cuve d'une chaudière est contrôlée par logiciel. Pour le contrôleur du sytème, nous distinguons quatre états :

OPERATING : Le système fonctionne.

STANDBY : Le système est arrêté.

SHUTDOWN : Le niveau d'eau a dépassé un des extrémas (mini ou maxi), ce qui peut provoqué un arrêt si le niveau d'eau ne repasse pas dans des limites plus contraignantes que les extrémas (Extréma ± hystérésis) avant 200 ms.

TEST : Le syteme est en test.

L'état initial du contrôleur est **STANDBY**.

Les changements d'états du contrôleur sont liés à des actions effectuées par l'opérateur, à des événements générés par des capteurs, mais également à des conditions temporelles. Toutes ces informations sont véhiculées au contrôleur par l'intermédiaire de signaux électriques.

Le système est donc de type réactif et temps réel, les changements d'états sont synchronisés d'une part aux fronts montants et descendants des signaux ; d'autre part à la validité de conditions.

Nous introduisons, pour un signal donné, les notations suivantes :

toT : Le front montant.

toF : Le front descendant.

isT : Le signal reste haut.

isF : Le signal reste bas.

L'opérateur peut agir sur le contrôleur grâce aux deux boutons, *SelfTest* et *Reset*, à sa disposition. Suivant le type d'action, les événements générés sont :

SelfTestPressed L'opérateur appuie sur le bouton *SelfTest*.

SelfTestInterval L'opérateur appuie sur le bouton *SelfTest* depuis au moins 500 ms.

ResetInterval L'opérateur appuie sur le bouton *Reset* depuis au moins 3 s.

Le contrôleur réagit également aux signaux et/ou conditions suivantes :

WithinLimits Le niveau de l'eau vérifie ; $LimiteMini < niveau < LimiteMaxi$.

InsideHysRange Le niveau de l'eau vérifie ; $(LimiteMini + 0.5cm) < niveau < (LimiteMaxi - 0.5cm)$.

TestInterval Le système est en mode TEST depuis au moins 14 s.

ShutdownLockTime Le système est en mode shutdown depuis au moins 200 ms.

Avec ces notations, la table suivante décrit les changements d'états du contrôleur.

Les changements d'états du contrôleur								
Etat	Inside HysRange	Within Limits	SelfTest Pressed	SelfTest Interval	Test Interval	Reset Interval	Shutdown LockTime	Etat
STANDBY	isT	isT				toT		OPERATING
STANDBY			isT	toT				TEST
OPERATING	isF	toF	isF	isF				SHUTDOWN
OPERATING			isT	toT				TEST
SHUTDOWN	toT	isT	isF	isF			isF	OPERATING
SHUTDOWN			isF	isF			toT	STANDBY
SHUTDOWN			isT	toT				TEST
TEST					toT			STANDBY

6.1.2 Spécifications de sureté

Le contrôleur précédemment décrit ne sera implémenté que lorsque les propriétés suivantes auront étées prouvées :

1. Lorsque le contrôleur n'est pas dans l'état **TEST** et que l'opérateur appuie sur le bouton *SelfTest* depuis au moins 500 ms, le contrôleur passe dans l'état **TEST**.
2. Lorsque le contrôleur est dans l'état **STANDBY**, l'opérateur n'appuie pas sur le bouton *SelfTest* depuis au moins 500 ms.
3. Lorsque le contrôleur est dans l'état **OPERATING**, l'opérateur n'appuie pas sur le bouton *SelfTest* depuis au moins 500 ms et soit le niveau est correct, soit l'opérateur appuie sur le bouton *SelfTest*.
4. Lorsque le contrôleur est dans l'état **SHUTDOWN**, l'opérateur n'appuie pas sur le bouton *SelfTest* depuis au moins 500 ms et le niveau de l'eau ne vérifie pas ; $(LimiteMini + 0.5cm) < niveau < (LimiteMaxi - 0.5cm)$ et le système n'est pas en mode shutdown depuis au moins 200 ms et l'opérateur n'appuie pas sur le bouton *SelfTest*.

De plus, afin de pouvoir commercialiser ce contrôleur, toute propriété prouvée et garantissant un *meilleur* fonctionnement est intéressante.

Question 6.1.1 *Vous devez modéliser ce cahier des charges, en interprétant le moins possible, mais en ajoutant suffisemment de contraintes afin que les propriétés de sureté soit vérifiées.*

Chapitre 7

Synthèse d'un scheduleur

7.1 Une application répartie temps réel

7.1.1 Le dispositif physique

Une application utilise, pour des raisons économiques, deux processeurs de bas-niveau et très spécialisés, en lieu et place d'un processeur unique possédant l'ensemble des caractéristiques nécessaires.
Cette contrainte vous oblige à concevoir et à développer votre application comme répartie (ou distribuée) sur ces deux processeurs. La spécialisation des processeurs (l'un pour les entrées-sorties, l'autre pour les calculs) entrainera logiquement la répartition des éxécutions des différentes tâches à réalisées.
La description physique du matériel peut donc se résumer comme suit :

- Deux sites ayant chacun :
  - un proceseur,
  - une mémoire,
- Des liaisons entre ces deux sites pouvant être assimilées à deux buffers parfaits et unidirectionnels de taille un.
- Les informations qui transitent sur ces liaisons sont contrôlées par un protocole de communication qui pourra au choix être supposé soit :
  1. half-duplex,
  2. full-duplex,

7.1.2 Le besoin logiciel

L'application doit être non seulement maintenable, mais doit également servir de base (ou de modèle) à un ensemble d'applications similaires. En fait, il s'agit d'une famille d'applications utilisant sensiblement les mêmes ressources matériels, mais dont les fonctionnalités peuvent différées.
Cette contrainte vous oblige à concevoir cette application en plusieurs couches :

1. une couche protocole, essentiellement une couche de gestion de flux de données entre les deux sites,
2. une couche noyau,
3. une couche application très modulaire et facilement interchangeable.

7.1.3 Les choix du chef de projet

Le découpage fonctionnel des différentes couches auquel à pense le concepteur est le suivant :

- la couche application est constituée d'un ensemble de tâches (à la ADA) qui communique par rendez-vous, que les tâches soient ou non sur le même site.

– le noyau gère les demandes de rendez-vous, il est en fait un séquenceur des rendez-vous. Il utilise la couche protocole lorsque le rendez-vous concerne deux tâches situées sur des sites différents.
– la couche protocole assure le comportement d'un médium parfait vis à vis des requêtes qui lui sont transmises.

Le concepteur impose, afin de pouvoir respecter ses objectifs :

1. les spécifications formelles,
2. l'outil MEC.

7.2 Synthèse d'un séquenceur à Rendez-vous

La but du TD-TP est de fournir aux développeurs de la couche "noyau" des modèles de description comportemental des rendez-vous.

7.2.1 Le cahier des charges

Description fonctionnelle

Deux types de rendez-vous sont à considérer :

– les rendez-vous dits *forts*, la tâche appelante est bloquée pendant que la tâche appelée exécute le corps du rendez-vous ;
– les rendez-vous dits *faibles*, la tâche appelante dépose une information dans une boâte aux lettres, puis continue son travail. La tâche appelée exécute le corps du rendez-vous ultérieurement.

Description pratique

Du fait que les tâches peuvent être sur des processeurs différents, il y a quatre implémentations distinctes :

– les rendez-vous forts locaux ;
– les rendez-vous forts distants ;
– les rendez-vous faibles locaux ;
– les rendez-vous faibles distants ;

7.2.2 Descriptions incomplètes des rendez-vous

Le comportement simplifié d'un rendez-vous fort local est :

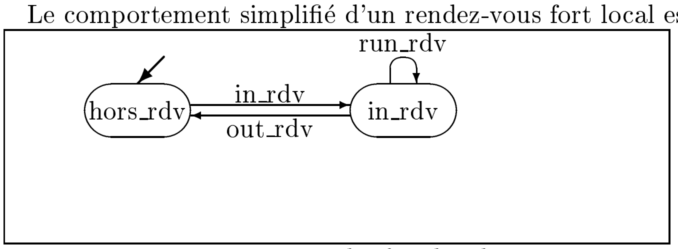

Figure 7.1: RdV fort local

Par contre, il est très difficile de décrire directement les trois autres cas, On se heurte en effet rapidement à des problèmes de formalisation concernant :

– Le comportement général,
– Le contrôle,
– La modélisation.

Question 7.2.1 *Utiliser le modèle Arnold-Nivat et l'outil MEC pour construire le comportement des rendez-vous, comme produit de synchronisation de systèmes plus simples, afin de valider le comportement du séquenceur vis à vis des couches inférieure (protocole) et supérieure (application). Vous pouvez faire les suppositions suivantes :*

1. *La couche inférieure est représentée par deux médiums parfaits ;*
2. *les rendez-vous sont étiquétées de manière unique. Cet identificateur permet de déterminer la tâche appelante, la tâche appelée et le type de rendez-vous ;*
3. *la liste des rendez-vous est connue à la compilation.*
4. *Pour chaque rendez-vous, le comportement global du séquenceur doit tenir compte :*
   - *des signaux venant de l'application (demandes de service) ;*
   - *des signaux venant du protocole.*

Troisième partie

Examens

Chapitre 8

DESS 95/96

Abstract

This article releases an industrial experiment of using formal methods to analyze and to debug a system that was shown erroneous by testing. After presenting the industrial context of the experiment, it details the modelling process and the interpretation of the results, in conjunction with the designers of the system.

8.1 Introduction

8.1.1 The EUROTRI project

The EUROTRI project was aimed by Schlumberger-Industries to define, conceive, design and develop a static electricity meter with large abilities of tariff programming and distant measuring. As short time of development and low cost of production (ie : use of components at frontier of their technical limits) were the challenges, it has been decided at the very beginning of the project to use formal methods in most steps of it, from feasability study to reception testing. For different reasons, including readability of transition systems by electronic engineers and availability of a powerful model-checker, the Arnold-Nivat model and the MEC tool have been selected, in conjunction with rate monotonic analysis. Another part of the project, the study of the physical layer of the two processors, at the interface of hardware and software, has been released in [?]. The all work has been presented as a thesis by Jean-Pierre Radoux [?].

fineffacе

8.2 The OUCABO system

Dealing with an electricity meter, it is a basic feature to iteratively measure N numerical values (such as voltage, intensity and phase) and to process them in the flow. At a given time t, one among the N analog signals is input in the measuring software, by the mean of a 1-in-n multiplexer, driven by an external (external to the software) counter. So we have two counters referring to the same value : an *external* (hardware) counter, selecting the input signal, an *internal* (software) counter, processing the input signal, both of them being controled by the same clock.

The first problem we have to deal with is that parasites can impact the links between the common clock and both devices, and thus desynchronize both counters, making the software process erroneously input values.

This can be handled either by identifying each sample transmitted to the software or by periodically synchronizing both counters. The first solution was not relevant for technical reasons, and it has been designed that an output port would be used by the software counter to reset the hardware counter.

In such an embedded system as the electricity meter, various processes have to be activated to ensure the fiability of the apparatus. One of them is the OUCABO system. This system requires a device to be periodically activated and tested. *The second problem is that the solution to first problem left no output port* *available to activate this secondary device, nor input port to test it.* The situation on the output port has been handled using the leftmost carry bit of the hardware counter to activate the device : if the software counter does not reset the hardware counter in due time, there occurs an arithmetic overflow and the device is activated.

8.3 Hard and soft counters

The system contains a hardware counter modulo N activated by ticks of a clock and able to be reset to 0 by an input signal. If N ticks occur without any reset, a carry bit is set. Each tick of the clock generates an interruption : the interruption routine increments a counter, called **channel** and initialized to 0, when its value is strictly less than $N-1$, and resets it to 0 and sends a reset signal to the hardware counter when it has value $N-1$. The code of this interruption routine is thus
if channel $< N-1$ **then increment(channel) else send(reset) ; channel :=0 ; endif**

8.3.1 Modeling the hardware counter

The first thing to do is to give a definite value to N. We assume that N has a lesser value than the actual value, say 4, and it will not be difficult to convince oneself that is a quite admissible simplification with respect to the verifications to be done. The hardware counter can have its carry bit set or not. In order to make things easier, we model activity of the carry bit during $3 \times N$ ticks.

Therefore, the hardware counter can be modeled by a transition with thirteen states. A state of the counter is encoded by a two-digit number : the second digit (0,1,2, or 3) represents the value of the counter and the first digit counts the number of carries since its last reset. Two actions can change the state of the counter :

- a tick of the clock, denoted by `t`,
- a reset signal, denoted by `reset`.

We also use the null action denoted by `e` to express that something else may occurs that has no effect on the counter.

In MEC, such a transition system is written

```
transition_system counter;

00 |-e-> 00, reset-> 00, t-> 01;
01 |-e-> 01, reset-> 00, t-> 02;
02 |-e-> 02, reset-> 00, t-> 03;
03 |-e-> 03, reset-> 00, t-> 10;
10 |-e-> 10, reset-> 00, t-> 11;
11 |-e-> 11, reset-> 00, t-> 12;
12 |-e-> 12, reset-> 00, t-> 13;
13 |-e-> 13, reset-> 00, t-> 20;
20 |-e-> 20, reset-> 00, t-> 21;
21 |-e-> 21, reset-> 00, t-> 22;
22 |-e-> 22, reset-> 00, t-> 23;
23 |-e-> 23, reset-> 00, t-> 30;

<initial={00}>.
```

8.3.2 Modeling the software counter

The software counter (i.e., the code above), is similar to the hardware counter : its states are the values of **channel** and when a tick occurs (action denoted by `t`) this value changes. However, when this value is 3, a reset signal is immediately sent to the hardware counter before the value becomes 0 (action denoted by `reset`). Therefore, we get :

```
transition_system channel_v1;

0 |-e-> 0, t->1;
1 |-e-> 1, t->2;
2 |-e-> 2, t->3;
3 |-e-> 3, t->30;
30|-reset->0;

<initial={0}>.
```

Remark that when the channel receives a tick in state 3, it goes into an intermediate state 30. In this state it is not allowed to wait (there is no “e-loop”), it has to immediately execute `reset` and to go into state 0.

8.3.3 Synchronization constraint

We consider the system composed of the counter and the channel. When a tick occurs, both components react simultaneously by executing a transition labeled by the action `t`. When the interruption routine sends a reset signal by `reset`, the counter simultaneously receives this signal and executes `reset` too. Finally both components can stay idle together. The system together with this synchronization constraint is described by :

```
synchronization_system s1 <width = 2; list = (counter, channel_v1)>;
(t     .t     );
(reset.reset);
(e     .e     ).
```

The synchronized product of this system is :

```
transition_system r1 <width = 2> ;
e(00.0) |- (e.e) -> e(00.0) , (t.t) -> e(01.1) ;
e(01.1) |- (e.e) -> e(01.1) , (t.t) -> e(02.2) ;
e(02.2) |- (e.e) -> e(02.2) , (t.t) -> e(03.3) ;
e(03.3) |- (e.e) -> e(03.3) , (t.t) -> e(10.30);
e(10.30)|- (reset.reset) -> e(00.0);
 < initial = { e(00.0) } >.
```

and one can see that, excepted when the channel is in its intermediate state 30, the values of the counter and of the channel are the same.

8.4 Introduction of a mode

Actually, for some reasons relating to the activity of the secondary device via the carry bit, the above system can be in two modes, special and non special. The mode becomes special when an external and uncontrollable event, called `start`, is generated by the environment of the system ; we assume, to simpplify the model, that once in the special mode, the system cannot come back to the normal mode. The only difference between the two modes is that no reset signal is emitted when in special mode, and thus the carry bit will be set to activate the secondary device :
if channel $< N-1$ **then increment(channel)**
else if mode is not special then send(reset) ; endif channel :=0 ; endif
We model the mode by the transition system

```
transition_system mode_v1;
not_special |-e-> not_special,
              start-> special,
              is_not_special-> not_special;
special |-e-> special,
```

```
          is_special-> special;
<initial={not_special}>.
```

The action `start` switches the mode from non special to special.
The actions `is_not_special` and `is_special` allows to know what the mode of the system is.

We now have a system with three components : `counter`, `channel_v1`, `mode_v1`.

Question 8.4.1 Donner les vecteurs de synchronisation décrivant les interactions de ces trois composants. Quels sont les nombres d'états et de transitions du produit synchronisé obtenu ?

Question 8.4.2 Montrer que `start` est bien une action incontrôlable.

8.5 Activation of a secondary device

When the mode is special, the channel 1 of the multiplexer is used to read a value produced by the device that has been activated by the carry bit of the counter. We use an action "read" to explicit the fact that a value is read on this channel coming from the device.

The interruption routine becomes
case
$0 <$ **channel** $< N - 1 \Rightarrow$ **increment(channel) ;**
channel $= 0 \Rightarrow$ **channel :=1 ; if mode is special then read endif ;**
channel $= N - 1 \Rightarrow$ **if mode is not special then send(reset) endif ; channel :=0 ;**
end
that is modeled by slightly modifying the previous model : adding a new intermediate state `01` between `0` and `1`.

```
transition_system channel_v2;

0 |-e-> 0, t->01;
01|-read-> 1;
1 |-e-> 1, t->2;
2 |-e-> 2, t->3;
3 |-e-> 3, t->30;
30|-reset->0;

<initial={0}>.
```

Indeed when is special mode the secondary device is activated twice and read twice. We introduce a transition system that counts how many times the device has been read.

```
transition_system device;
0 |-e-> 0, read-> 1;
1 |-e-> 1, read-> 2;
2 |-e-> 2;
<initial = {0}>.
```

As for the reset signal, the "read" action will be taken into account depending on the mode is special or not.

Question 8.5.1 Donner les vecteurs de synchronisation décrivant les interactions de ces quatre composants. Quels sont les nombres d'états et de transitions du produit synchronisé obtenu ?

8.6 Fixing a bug

To behave correctly the system must meet the following property P :

> Just after any reading of the device, the number of activations of the device (i.e., the first digit of the state of the hardware counter) must be equal to the number of readings (i.e., the state of `device`).

The designers of the code were almost sure that this property was true. Experiments showed that indeed it was not, and we introduced the modeling above just to understand how and why it happened.

8.6.1 Finding the cause of the errors

First, we look at the states reached after any reading of the device.

Question 8.6.1 Expliquer comment calculer cet ensemble d'états. Quels sont ceux qui satisfont la contrainte ci-dessus et ceux qui ne la satisfont pas.

Then one guesses that the problem is probably caused by the mode switching from non special to special while the channel has value 0.

Question 8.6.2 Comment peut-on s'assurer que cette hypothèse est correcte ?

8.6.2 The correction

A solution for avoiding the problem is to force the system to switch to special mode when the channel has not value 0. But there is no possible control on the start event. On the other hand, it is possible to control the time when the mode switches to special. Thus, we "desynchronize" the two actions : when the start signal is received, the mode goes into a special state informing the system that this event has happened, and the system goes from this state to the special mode under control of the interruption routine by the action named `switch`. Remark that this action can be executed only when the mode is in state `request`.

```
transition_system mode_v2;
not_special |-e-> not_special,
          start-> request,
          is_not_special-> not_special;
request |-e-> request,
          switch-> special
          is_request-> request;
special |-e-> special,
          is_special-> special;
<initial={not_special}>.
```

When setting the channel from 0 to 1 the interruption routine makes the device to be read if the mode is special and do nothing if the mode is not special. If the mode is in the intermediate state `request` it switches the mode to `special`. That new value of the mode will have effect at the next `read`, as if the start event has occurred when the channel has value 1, 2, or 3 ! Of course, this cause a delay of four additionnal ticks before switching to the special mode, but engineers found it quite acceptable.

Question 8.6.3 Ecrire le système de transitions modélisant cette nouvelle routine d'interruption. Donner les vecteurs de synchronisation décrivant les interactions de ce nouveau système. Vérifier que la propriété P (page 34) est maintenant satisfaite.

Question 8.6.4 Ecrire le code de cette nouvelle routine.

Chapitre 9

DESS 96/97

9.1 Introduction

Cet exercice est inspiré de la thèse de François Laroussinie [?]. Il consiste, en partant d'une spécification informelle d'un ascenseur, d'une part à traduire cette spécification en terme de propriétés logiques, d'autre part à construire un modèle d'ascenseur qui satisfait cet ensemble de propriétés.

9.2 Spécifications informelles d'un ascenseur

L'ascenseur que nous allons étudier dessert n étages. La cabine comporte n boutons lumineux (ie lorsqu'un bouton est allumé, il témoigne de l'existence d'une requête pour cet étage) qui permettent de choisir la (ou les) destination(s). De plus, à chaque étage se trouve un bouton de même type qui permet d'appeler l'ascenseur. Lorsque la cabine s'arrête à un étage, la porte s'ouvre alors automatiquement.

Le comportement de cet ascenseur est contrôlé par logiciel. Les changements d'états du contrôleur sont liés aux actions sur les boutons lumineux. Le donneur d'ordre choisira le constructeur qui pourra lui prouver que les comportements suivants sont *"vrais"*.

1. – Quand un bouton est enfoncé, le voyant correspondant s'allume.
   – Quand la requête correspondante est satisfaite, le voyant s'éteint.
2. A chaque étage, la porte n'est jamais ouverte si la cabine n'est pas là.
3. Chaque requête doit être satisfaite "un jour".
4. L'ascenseur ne dessert que les étages pour lesquels existe une requête.
5. Lorsqu'il n'y a pas de requête, la cabine reste à l'étage où elle se trouve.
6. Lorsque la cabine se déplace, elle doit s'arrêter aux étages par lesquels elle passe si une requête les concerne.
7. Lorsqu'il existe plusieurs requêtes, l'ascenseur doit traiter prioritairement celle(s) permettant de continuer dans la même direction que le dernier déplacement.

9.3 Le premier modèle

Le premier modèle est obtenu par description des trois objets *physiques* utilisés : le bouton, la porte et la cabine, et de leurs interactions "naturelles".

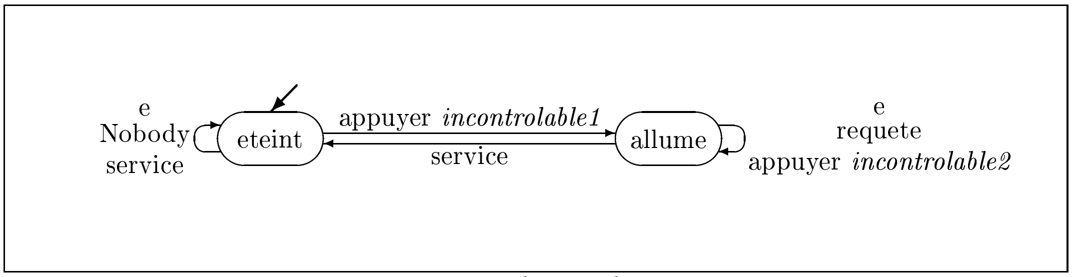

Figure 9.1: Un bouton lumineux.

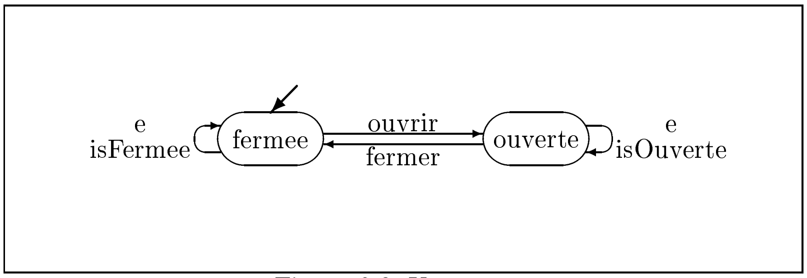

Figure 9.2: Une porte.

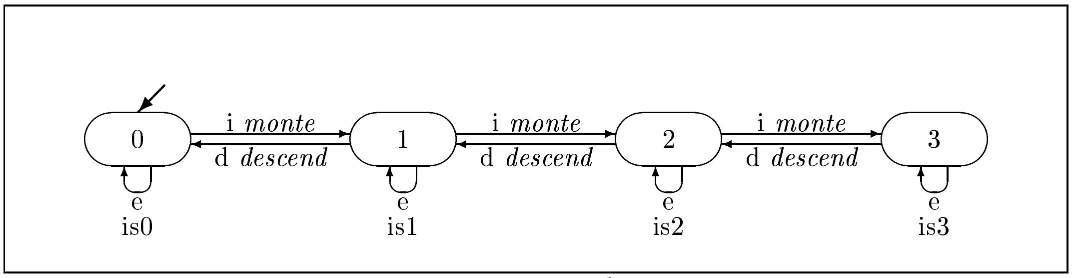

Figure 9.3: La cabine.

bouton	porte	bouton	porte	bouton	porte	bouton	porte	cabine1	bouton	bouton	bouton	bouton
appuyer	e	e	e	e	e	e	e	e	e	e	e	e
e	e	appuyer	e	e	e	e	e	e	e	e	e	e
e	e	e	e	appuyer	e	e	e	e	e	e	e	e
e	e	e	e	e	e	appuyer	e	e	e	e	e	e
e	e	e	e	e	e	e	e	e	appuyer	e	e	e
e	e	e	e	e	e	e	e	e	e	appuyer	e	e
e	e	e	e	e	e	e	e	e	e	e	appuyer	e
e	e	e	e	e	e	e	e	e	e	e	e	appuyer
service	fermer	e	e	e	e	e	e	is0	service	e	e	e
e	e	service	fermer	e	e	e	e	is1	e	service	e	e
e	e	e	e	service	fermer	e	e	is2	e	e	service	e
e	e	e	e	e	e	service	fermer	is3	e	e	e	service
requete	ouvrir	e	e	e	e	e	e	is0	e	e	e	e
e	ouvrir	e	e	e	e	e	e	is0	requete	e	e	e
e	e	requete	ouvrir	e	e	e	e	is1	e	e	e	e
e	e	e	ouvrir	e	e	e	e	is1	e	requete	e	e
e	e	e	e	requete	ouvrir	e	e	is2	e	e	e	e
e	e	e	e	e	ouvrir	e	e	is2	e	e	requete	e
e	e	e	e	e	e	requete	ouvrir	is3	e	e	e	e
e	e	e	e	e	e	e	ouvrir	is3	e	e	e	requete
e	isFermee	e	isFermee	e	isFermee	e	isFermee	i	e	e	e	e
e	isFermee	e	isFermee	e	isFermee	e	isFermee	d	e	e	e	e

Question 9.3.1 *L'ascenseur modélisé dessert 4 étages. Pensez-vous que 4 soit nécessaire et/ou suffisant*

pour pouvoir convaincre le donneur d'ordre ?

Question 9.3.2 *L'environnement (les personnes) n'a pas été modélisé. Pourquoi ?*

Question 9.3.3 *Calculer le comportement global de ce premier modèle. Combien d'états et de transitions comprend t-il ?*

9.4 La validation du premier modèle

Pour convaincre le donneur d'ordre que les objets manipulés définissent bien un ascenseur, les propriétés suivantes sont calculées.

```
dead := *-src(*);
boucle := loop(*,*);
deadlock := inevitable(*,dead);
proj_state(1,tous,*);
proj_trans(1,toutes,*);
dts(bouton);
wts(src(*-toutes),*-toutes);
src_incont1 := src(incontrolable1);
src_incont2 := src(incontrolable2);
dts(nom);
proj_state(2,tous,*);
proj_trans(2,toutes,*);
dts(porte);
wts(src(*-toutes),*-toutes);
dts(nom);
proj_state(9,tous,*);
proj_trans(9,toutes,*);
dts(cabine);
wts(src(*-toutes),*-toutes);
dts(nom);
proj_state(10,tous,*);
proj_trans(10,toutes,*);
dts(bouton);
dts(nom);
PbAppuyer1 := src_incont1[1] - src(incontrolable1[1]);
PbAppuyer2 := src_incont2[1] - src(incontrolable2[1]);
```

Question 9.4.1 *Le premier modèle est-il valide ?*

Question 9.4.2 *Les actions incontrôlables sont-elles correctement modélisées ?*

9.5 La vérification du premier modèle

Dans le but de vérifier si les spécifications sont satisfaites ou non par ce modèle, les pré-calculs suivants sont effectués.

```
AppelEtage0 := incontrolable1[1];
AppelEtage1 := incontrolable1[3];
AppelEtage2 := incontrolable1[5];
AppelEtage3 := incontrolable1[7];
AppelCabine0 := incontrolable1[10];
AppelCabine1 := incontrolable1[11];
AppelCabine2 := incontrolable1[12];
AppelCabine3 := incontrolable1[13];
```

```
Appel0 := AppelEtage0 \/ AppelCabine0;
Appel1 := AppelEtage1 \/ AppelCabine1;
Appel2 := AppelEtage2 \/ AppelCabine2;
Appel3 := AppelEtage3 \/ AppelCabine3;
Appels := (Appel0 \/ Appel1) \/ (Appel2 \/ Appel3);
RappelEtage0 := incontrolable2[1];
RappelEtage1 := incontrolable2[3];
RappelEtage2 := incontrolable2[5];
RappelEtage3 := incontrolable2[7];
RappelCabine0 := incontrolable2[10];
RappelCabine1 := incontrolable2[11];
RappelCabine2 := incontrolable2[12];
RappelCabine3 := incontrolable2[13];
Ouvrir0 := !label[2] = "ouvrir";
Ouvrir1 := !label[4] = "ouvrir";
Ouvrir2 := !label[6] = "ouvrir";
Ouvrir3 := !label[8] = "ouvrir";
Service0 := !label[2] = "fermer";
Service1 := !label[4] = "fermer";
Service2 := !label[6] = "fermer";
Service3 := !label[8] = "fermer";
AllumeEtage0 := !state[1] = "allume";
AllumeEtage1 := !state[3] = "allume";
AllumeEtage2 := !state[5] = "allume";
AllumeEtage3 := !state[7] = "allume";
AllumeCabine0 := !state[10] = "allume";
AllumeCabine1 := !state[11] = "allume";
AllumeCabine2 := !state[12] = "allume";
AllumeCabine3 := !state[13] = "allume";
EteintEtage0 := !state[1] = "eteint";
EteintEtage1 := !state[3] = "eteint";
EteintEtage2 := !state[5] = "eteint";
EteintEtage3 := !state[7] = "eteint";
EteintCabine0 := !state[10] = "eteint";
EteintCabine1 := !state[11] = "eteint";
EteintCabine2 := !state[12] = "eteint";
EteintCabine3 := !state[13] = "eteint";
PorteOuverte0 := !state[2] = 'ouverte';
PorteOuverte1 := !state[4] = 'ouverte';
PorteOuverte2 := !state[6] = 'ouverte';
PorteOuverte3 := !state[8] = 'ouverte';
Cabine0 := !state[9] = '0';
Cabine1 := !state[9] = '1';
Cabine2 := !state[9] = '2';
Cabine3 := !state[9] = '3';
CabineNon0 := !state[9] # '0';
CabineNon1 := !state[9] # '1';
CabineNon2 := !state[9] # '2';
CabineNon3 := !state[9] # '3';
Monte := monte[9];
Descend := descend[9];
NoRequete := !state # "*allume*";
ApresAppuyer0 := reach(tgt(Appel0),* - (Appel0 \/ Service0));
```

```
AvantService0 := coreach(src(Service0),* - (Appel0 \/ Service0));
ApresAppuyer1 := reach(tgt(Appel1),* - (Appel1 \/ Service1));
AvantService1 := coreach(src(Service1),* - (Appel1 \/ Service1));
```

Question 9.5.1 *A l'aide de ces marques, donner les expressions des propriétés 1, 2, 4, 5 et 6.*

Les propriétés 3 et 7 sont des propriétés dites de *vivacité*. Pour les vérifier, nous calculons :

```
propriete3A0 := loop(*, rsrc(ApresAppuyer0) /\ rtgt(AvantService0));
propriete3A1 := loop(*, rsrc(ApresAppuyer1) /\ rtgt(AvantService1));
propriete3B0 := loop(Monte \/ Descend, rsrc(ApresAppuyer0) /\ rtgt(AvantService0));
propriete3B1 := loop(Monte \/ Descend, rsrc(ApresAppuyer1) /\ rtgt(AvantService1));
ApresMonte := reach(tgt(Monte),* - Appels);
ApresMonteDescend := reach(tgt(Descend) /\ ApresMonte, * - Appels);
propriete7 := ApresMonteDescend /\ src(Monte);
```

Question 9.5.2 *Expliquer en quoi diffèrent les propriétés 3A et 3B. Donnez en une description informelle (en français), puis dites laquelle des deux est certainement celle que le donneur d'ordre désire.*

Question 9.5.3 *Expliquer le calcul de la propriété 7. Correspond t-il exactement à la spécification informelle ?*

Question 9.5.4 *Montrer (par calcul avec Mec) que le premier modèle satisfait les propriétes 1, 2 et 4 mais ne satisfait pas les propriétés 3A, 3B, 5, 6 et 7.*

9.6 Le deuxième modèle

Une analyse des résultats du premier modèle montre un manque d'information sur les déplacements de la cabine. Celle-ci est donc remplacée par le système représenté par la figure 9.4.

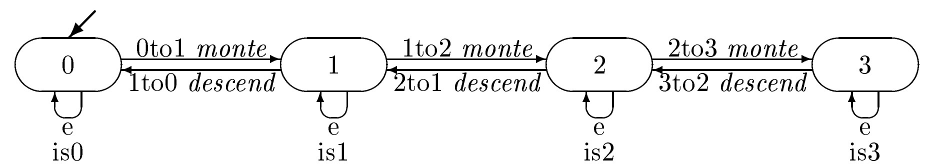

Figure 9.4: Une nouvelle cabine.

Question 9.6.1 *Donner le nouveau système de synchronisation.*

Question 9.6.2 *Montrer (par calcul avec Mec) que le second modèle satisfait les propriétes 1, 2, 4, 5 et 6 mais ne satisfait pas les propriétés 3A, 3B et 7.*

9.7 Le troisième modèle

Les propriétés 3 et 7 ne sont pas satisfaites, car il est difficile dans ce modèle de garantir qu'un événement se produira dans le futur. Il est donc nécessaire d'introduire un nouveau composant qui va contrôler les déplacements de la cabine.

Question 9.7.1 *Donner le système de transition d'un contrôleur (si possible indépendant du nombre d'étages desservis), et le système de synchronisation associé pour lesquels toutes les propriétés (sauf 3A) sont satisfaites.*

Chapitre 10

DESS 97/98

10.1 Introduction

Cet exercice est inspiré d'un cas test classique en sûreté de fonctionnement. Il consiste, en partant d'une spécification informelle d'un système de régulation de température, d'une part à traduire le fonctionnement nominal de la spécification en un un modèle de système valide et correct vis à vis d'un événement redouté ; d'autre part à ajouter les pannes décrites dans la spécification informelle afin d'exhiber les jeux de pannes minimaux qui conduisent inévitablement vers l'événement redouté.

10.2 HNO3 : un système de régulation de température

Il s'agit d'un système qui maintient la température d'un flux d'acide nitrique dans une plage acceptable. Le système est bien-connu, car il a déjà été choisi à plusieurs reprises dans la littérature pour illustrer l'aptitude de certaines méthodes à determiner l'ensemble de scénarios qui conduisent à l'occurence d'un événement redouté donné :

ER : Température de l'acide nitrique trop élévé en sortie du système.

Le système est représenté à la figure 10.1. Toutes les liaisons entre les composants sont supposés *parfaites*, alors que ces derniers peuvent connaätre des défaillances, mais uniquement celles qui sont listées dans la table 10.1. Aucun composant n'est réparable.

La mission du système consiste à refroidir un flux d'acide nitrique et à maintenir sa température dans une plage acceptable. Ceci est obtenu par l'action de l'une des trois séquences (interruptibles) décrites ci-après dans l'ordre chronologique de leur intervention :

- automatique : TS → TC → AP → V5 → HE
- manuelle : TS → TI → OP → V5 → HE
- manuelle : TS → TI → OP → V1 → HE

L'opérateur n'a qu'une seule source d'information : TI. Il n'agit pas tant que la température T6 lui apparaät égale à la température de consigne Tc, ou lui apparaät y être ramenée par la régulation automatique. Dans le cas contraire, il déconnecte l'automatisme de la commande de V5 par le contrôleur pneumatique AP, afin de pouvoir agir sur V5, puis sur V1 si ses actions s'avèrent inefficaces.

De même, toute défaillance de la pompe doit être détectée par FD, qui commande immédiatement la fermeture de V1.

A partir de l'état initial, deux évolutions possibles de la température de l'acide nitrique doivent être envisagées :

- un accroissement transitoire, dT, de cette température, trop bref pour pouvoir conduire, seul, à l'événement redouté.
- un accroissement persistant, DT, de cette température, qui, s'il n'est pas compensé par les régulations, conduit à l'événement.

Composants	Fonctions	Modes de défaillances
V1	Vanne de régulation du débit d'acide nitrique et de sécurité. Sa fonction de régulation est réalisée par action manuelle de l'opérateur. Sa fonction de sécurité résulte de FD ou de l'action de l'opérateur.	V1S : bloquée en position, V1 ne réagit pas aux sollicitations de FD ni à celles de l'opérateur.
HE	Echangeur. Il est supposé "parfait".	Aucune, si ce n'est que c'est ici que se produit l'explosion.
TS	Capteur de température. Il mesure la température T3 du flux de HNO3 en sortie de l'échangeur et transmet l'information T6 à TC et TI.	TSS : bloqué en position. TSFL : le capteur envoie une information érronée par défaut T6 $<$ T3 réelle.
TC	Contrôleur de température. Il compare la valeur de T6 à celle de Tc et, quand il détecte une dérive, modifie en conséquen,ce le débit de V5 par l'intermédiaire d'une commande pneumatique AP.	TCS : bloqué en position, TC n'envoie aucun ordre à V5.
V5	Vanne de régulation, contrôlée par AP. Elle contrôle elle-même le débit d'eau de refroidissement destiné à HE. Elle peut être actionnée manuellement par l'opérateur après que celui-ci ai déconnecté l'automatisme. Elle se ferme par manque d'air comprimé.	V5S : bloquée en position, V5 ne réagit ni aux sollicitations de TC via AP, ni à celles de l'opérateur. V5R : suite à une erreur de remontage après une opération de maintenance, V5 réagit à l'inverse des ordre reçus.
AP	Air comprimé, commande l'ouverture de V5.	LAP : perte d'air comprimé.
MP	Alimentation principale en eau de refroidissement.	LMP : perte de l'alimentation principale.
ES	Alimentation de secours en eau de refroidissement.	NES : perte de l'alimentation de secours.
P	Pompe, elle assure le débit nécessaire en eau de refroidissement.	PF : pompe défaillante (arretée).
TI	Indicateur de température. Il constitue la seule source d'information sur la température de sortie de HNO3, pour l'opérateur.	TIS : bloqué en position.
FD	Détecteur de panne de la pompe P. Lorsqu'il détecte une panne de P, il envoie un signal de fermeture à V1.	FDB : bloqué, il n'envoie aucun signal à V1.
OP	Opérateur. Il supplée les défaillances (ou supposées telles) de la régulation automatique en agissant sur V5 et, si nécessaire sur V1.	OPA : opérateur absent.

TAB. 10.1: Les composants

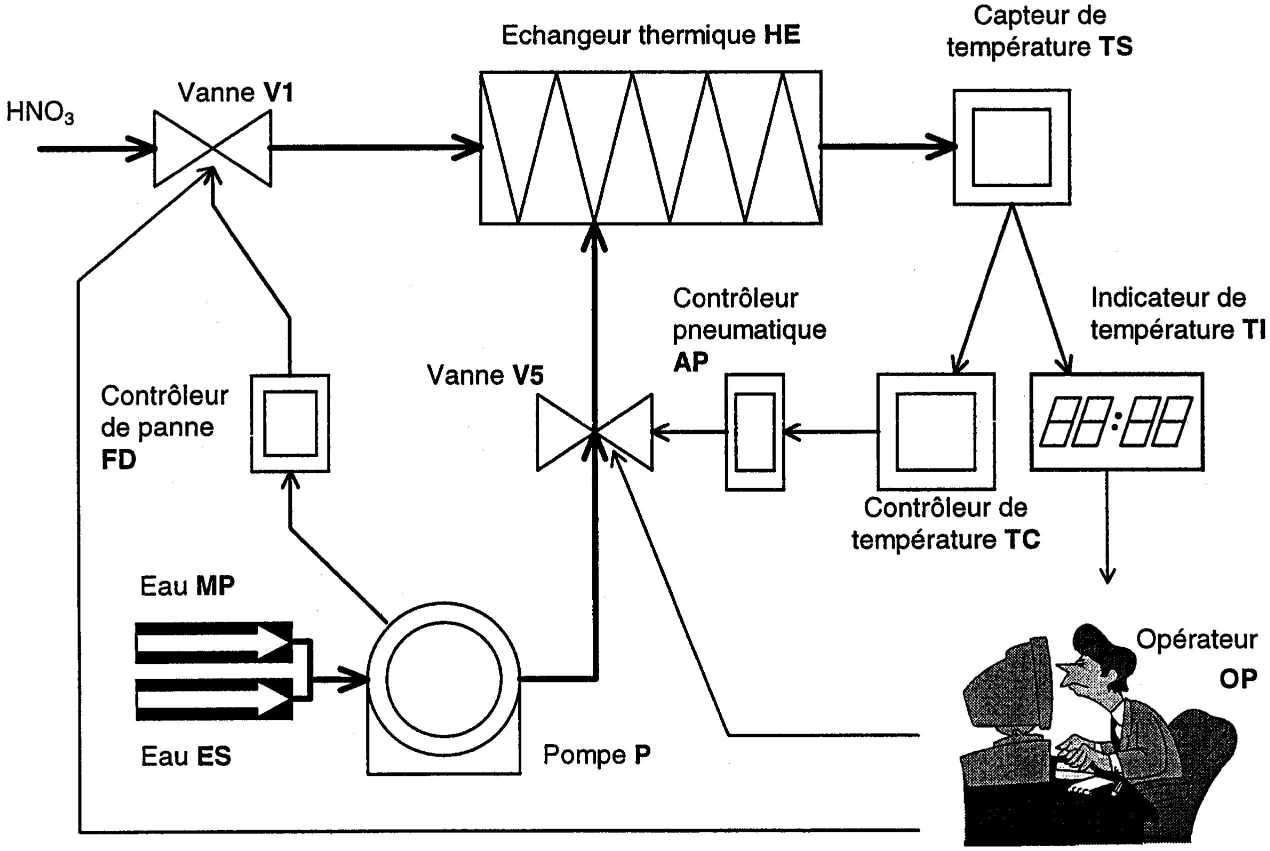

FIG. 10.1: Description du système

Le travail demandé comprend quelques questions et un travail de modélisation.

Question 10.2.1 *Le modèle hno3_v11 vous parait-il valide ?*
Si oui, modifiez simplement le calcul portant sur les l'événements incontrôlables, afin que les résultats soient toujours l'ensemble vide.

Question 10.2.2 *Expliquez la différence entre hno3_v11 et hno3_v12 ?*

Question 10.2.3 *Le modèle hno3_v21 vous parait-il valide ?*

Question 10.2.4 *Pour chacun des modèles, dites si l'événement redouté ER est :*

- *impossible,*
- *impossible si l'opérateur fait "correctement" son travail.*
- *possible si l'opérateur fait non "correctement" son travail. Vous donnerez une trace d'éxécution comme justificatif.*
- *inévitable. Vous donnerez la combinaison minimale de pannes conduisant à cette situation.*