Notes de Cours INF 3610

Olivier Sirois

2018-10-10

Contents

Chapter 1

Chapitre 1 - Introduction

On peu définir la vérification formelle comme étant l'utilisation de techniques théorique pour prouver le fonctionnement d'un système. Selon Sommerville, la vérification est la réponse à la question:
Are we building the product right

Pour etre vérifié, un système doit satisfaire:

- Vérification formele complète : exploration de façon exhaustive de tous les états possible du système
- Vérification formelle incomplète : vérification des bornes

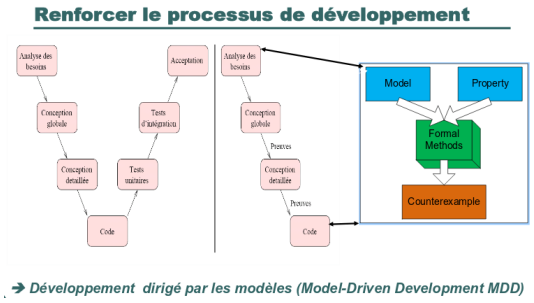


Figure 1.1: Processus de Développement

On vérifie formellement pour renforcer le processus de développement. Les techniques de vérification peuvent s'appliquer à différent niveau, car on regarde seulement le fonctionne du module sous test, pas tous le système.

On voit sur la photo la différence entre les deux méthodes de vérification, une est applicable à un gros système vs. l'autre qui ne peut être fait que sur des

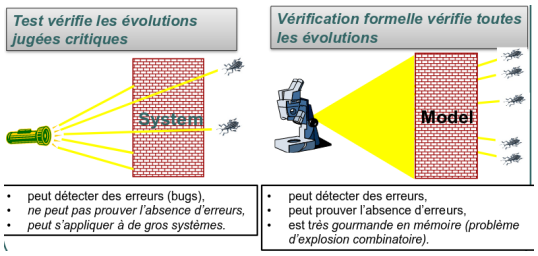


Figure 1.2: Différents types de tests

éléments de petites tailles.

Évidemment, certains accidents notables peuvent être attribuer à un manque de vérification. sa peut aller de AT&T à la NASA. (problème d'inversion).

Lorsqu'on fait affaire dans des systèmes critiques, soit:

- Transport (avionique, ferroviaire, spatial)
- Energie (nuclearire...)
- Medical (dosage radioactif)

En raison de la criticalités de ces applications, certains documents ont été mis en place pour la production de logiciel pour les applications aéronautiques.

Les normes ED-12C et DO-178C précisent ces contraintes.

1.0.1 Comment vérifier formellement

- Modéliser
- Spécifier
- Prouver

1.0.2 Comment spécifier un comportement

Le but de la modélisation est d'expirmer, au moyen d'un formalisme la manière dont le système se comporte. Le formalisme doit être assez expressif, repose sur une sémantique rigoureuse et doit offrire des possibilités d'analyse systématique. On peut résumé sa en une représentation simplifiée d'un système.

Dans le cas des systèmes temps réelles, le fonctionnement est assujetti à l'évolution dynamique de l'environnement et la réaction aux stimuli est soumise à des contraintes temporelles. c-a-d, il faut que la réponse à l'environnement arrive dans

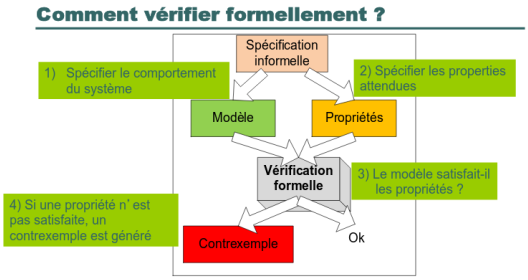


Figure 1.3: Flowchart représentant les étapes de la vérification formelle

un temps spécifier. Plusieurs modèle formels ont été développés puis adaptés aux systèmes temps réel. On peut les distingué par la séquentialité (concurrence), le non-déterminisme, la synchronisation, la communication, la compositionnalité, les contraintes temporelles. On a même certains language de spécifications fait pour pouvoir vérifier des sytèmes.

les classe de modèles proposé dans la litérature sont :

- langage de programmation et de spécialisation
- les modèles à base de transitions
- les modèles logiques
- les modèles algébriques

INSÉRÉ SLIDE 27

en générales, les modèles doivent spécifier les:

- actions :
  - Déplacement d'un ascenseur
  - fermeture de vanne
  - Écriture dans une base de donnée
- événement:
  - Dépassement d'une température
  - Réception d'un message
- contraintes temporelle
  - Trouver un vol pour une destination
  - envoi de message toutes les 100 secondes

- conditions d'activation
  - à respecter avant de déclencher des actions
  - s'il n'y a pas d'obstacle devant la cellule de l'ascenseur
  - fermer la porte de l'ascenseur
  - etc.
- situations anormale et leurs traitements
  - arrêt de l'ascenseur entre deux étages
  - appui de touche sans résultats
  - etc.
- les états significatifs
  - ascenseur à l'arrêt
  - ascenseur en mouvement
  - porte ouverte et ascenseur en mouvement

langage de prog -¿ séparer en deux partie:

- classique, JAVA, C, C++
- ceux basé sur un modele de transition, RT-LOTOS, PROMELA, ESTEREL, LUSTRE

1.0.3 JAVA

Java pathfinder, vérificateur (model-checker) pour la NASA. (babelfish).

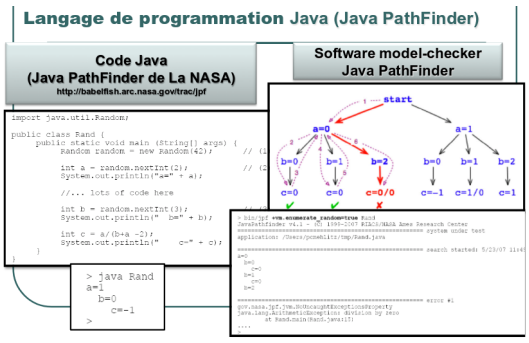


Figure 1.4: flowchart représentant le pathfinder java

1.0.4 LLBMC - low level bounded model checker

travail sur un code bas niveau (assembleur?). il fait du bounded model-checking dépendament du domaine spécifier.

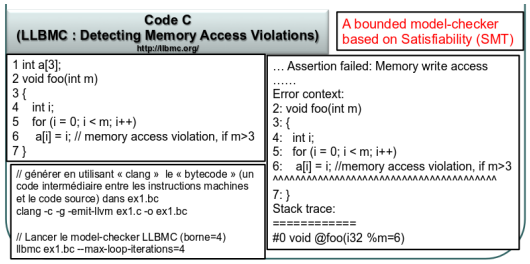


Figure 1.5: Exemple d'utilisation du LLBMC

1.0.5 Promela, model-checker SPIN

Promela (protocol/process meta language) est utilisé par le model checker SPIN. cré dynamiquement des processus concurrents, la communication entre processus via des variables partagées et des canaux de messages

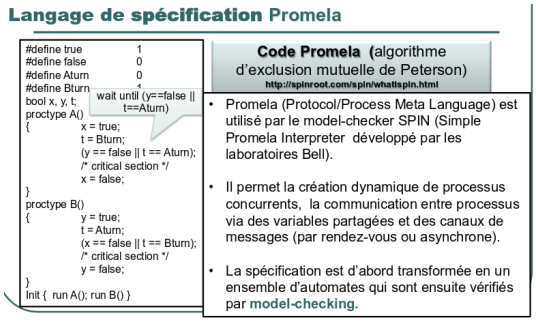


Figure 1.6: Exemple d'utilisation de Promela

1.0.6 LUSTRE, (boite a outils SCADE)

language synchrone a flots de donnees. Sa permet d'exprimer un systeme sour forme d'équations qui définissent l'évolution des valeurs de ses varables

Une variable x de LUSTRE représente une flot c-à-d une séquence infinie de valeurs ou x0 est la valeur initiale et xi est la valeur de x au i^{eme} cycle d'exécution.

on peut spécifier un programme aussi comme sur la slide 39

```
node MinMax(X : int)  returns (min, max : int);

let

min = X -> if (X < pre min) then X else pre min;

max = X -> if (X > pre max) then X else pre max;

tel
```

http://www-verimag.imag.fr/~raymond/edu/eng/lustre-a.pdf

Figure 1.7: Exemple d'utilisation de Lustre

1.0.7 Modele a base de transition

On simule une sorte de state machine mealy, ou on modélise les états discrèt par états ainsi que leur transitions par rapport à leurs actions

Ces modèles sont en générales graphique:

- Les Statecharts
- Les Ré seaux de Petri
- Les Automates

Les automates offrent un bon compromis entre la puissance de modélisation et la complexité de vérification

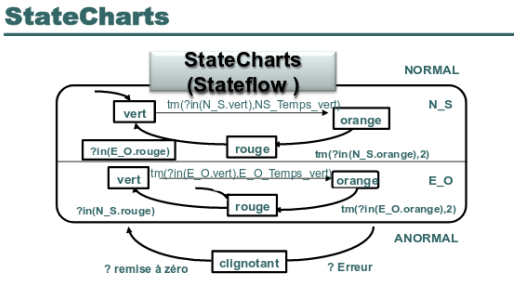


Figure 1.8: Exemple d'une statechart

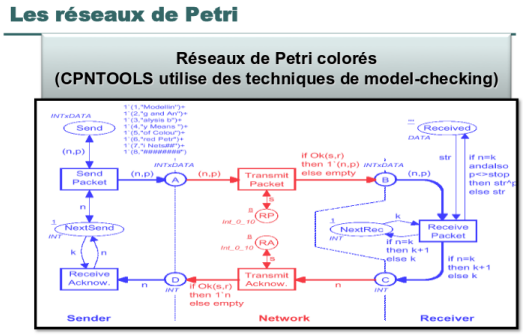


Figure 1.9: Exemple d'un réseaux de pétri

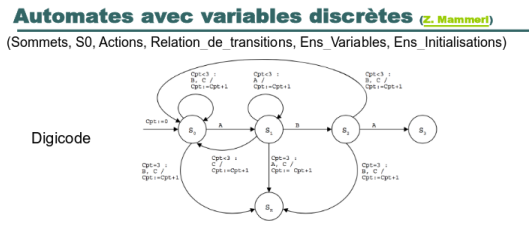


Figure 1.10: Exemple d'un automate avec variables discrèt

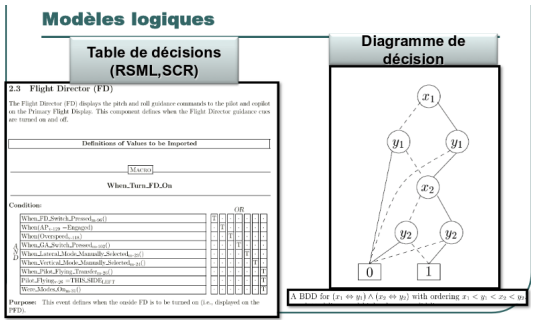


Figure 1.11: Exemple d'une représentation par modèle logique

1.0.8 Modeles Logiques

1.0.9 Modèles Algébrique

Description du comportement d'un système à l'aide d'une description algébrique. On combine des opérateurs et des actions.

- préfixage : $<;>: a; C$
- parallèle : $<||>: C_1||C_2$

- choix (non déterministe) $< [] > ou < + >: C_1[]C2$

l'algèbre des processus CCS et ATP sont dans ces catégories.

1.0.10 Comment spécifier une proprité

On peut utiliser la logique (propositionelle, prédicat), pour spécifier des chose. Sauf que c'étais insuffisant pour décrire des systèmes à comportement temporels. On a alors étendu ces notions à logique temporelle pour pouvoir vérifier formellement des systèmes de natures temporelle.

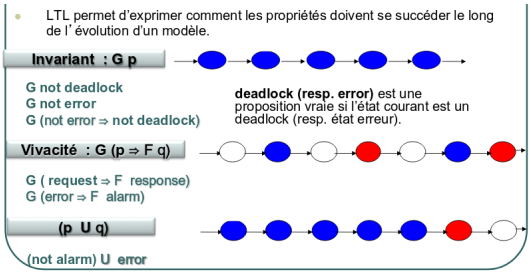


Figure 1.12: Exemple de différentes propriétés de LTL

1.0.11 Logique temporelle linéaire

LTL permet d'expliquer comme la logique évolue dans le temps

1.0.12 Logique temporelle arborescente (CTL)

Cette logique permet de quantifier les chemins d'exécutions

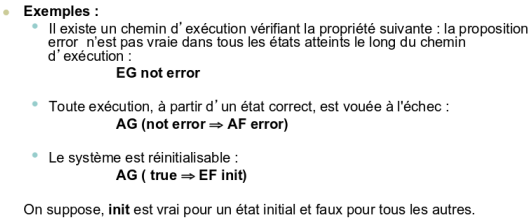


Figure 1.13: Exemple de différentes propriétés de CTL

1.0.13 Vérification effective

La vérification est dite décidable pour un ensemble de modèles M et une classe de propriétés P si et seulement si il existe un programme qui prend en trné un modèle quelconque de M et une propriété quelconque de P et détermine au bout d'un temps fini, si la propriété est satisfaite ou non par le modèle. Pour être automatisable, la vérification doit être décidable et aussi de complexité acceptable. On différencier entre deux grandes catégories de méthodes: les méthodes syntaxiques et les méthodes sémantiques.

1.0.14 Méthodes syntaxiques

Ce sont des preuves au sens mathématique du terme. Elles cherchent à déterminer si une propriété peut être obtenus.. difficile a automatiser

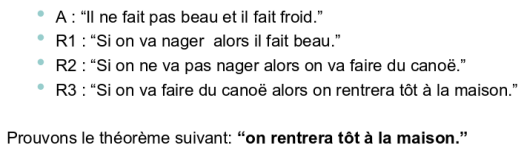


Figure 1.14: Exemple d'inférence avec la méthode syntaxique

Pour résoudre syntaxiquement, on utilise les méthodes vu dans le cours 8215 (logique prédicats/propositionelle)

1.0.15 Méthodes sémantiques

Dans les méthodes sémantiques, on se base sur l'exécution du modèle. L'approche populaire est le model-checking. Sa s'appuie sur deux formalisme:

- système de transition
- logique temporelle

1.0.16 Model-checking

Normalement automatique, et produit des contrexemple sous forme de traces qui sont utils à la compréhension des situations d'erreurs et àla correction. basé sur la sémantique d'entrelacement, avec sa, sa pourrait générer n! et plus de 2^n états

Gros problème d'explosion combinatoire avec sa..

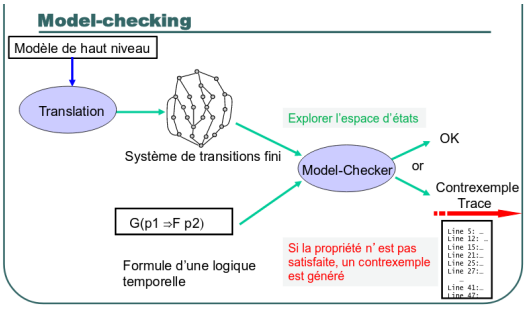


Figure 1.15: Exemple de float à haut niveau de Model-checking

Chapter 2

Cours 2 - Automate temporisé

2.1 Introduction

Systèmes temps réel Agit dans l'environnement..

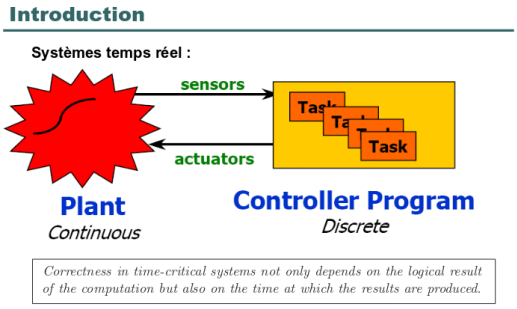


Figure 2.1: Concept de système temp réel

Normalement, le systèmes captes son environnement à l'aide de **capteurs**, fait des traitements (normalement dans un temps définis) et agit ensuite dans son environnement à l'aide d'**actuateurs**.

Le principe est que dans un système temps réel, on doit répondre dans un temps requis. La vérification de tels systèmes doivent alors prendre en considération les contraintes temporelles.

- Statecharts, réseaux de pétri temporisé, Automates temporisés

- Algèbre de processus temporisé
- language synchrone

Automates temporisé: offrent un très bon compromis entre la puissnace de modélisation et la complexité de vérification.

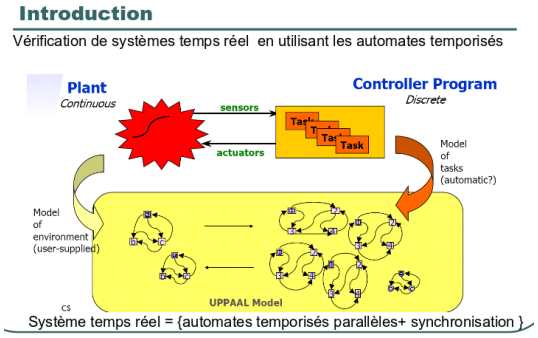


Figure 2.2: Système temps réel dans l'environnement

De plus, on ne peut pas se contenter de simplement modéliser le System under test, on doit aussi modéliser l'environnement et ses interactions avec le systèmes testé.

2.2 Automates

un automate est un tuple A = ¡ L, Act, E, I0¿ ou:

- L est un ensemble fini et non vide de sommets (locations)
- Act est un ensemble fini d'actions pour le sommet
- E ∈ L x Act x L est un ensemble fini de transition possible (transitions)
- I0 est 'état initiale

d'un automate, on peut extraite un évolution. Qui est une séquence alterné d'états et d'actions. Il est possible de déduire les évolutions possible du modèles. On appelle trace d'exécution la séquence d'action exécutées selon une évolution. ex: action1 -¿ action2 -¿ action3.

Avec deux automates, leurs composition est le produit scalaire entre les deux automates:

- $A_1||A_2 =< L_1xL_2, Act_1 \cup Act_2, E, (I_{01}, I_{01}) >$ ou

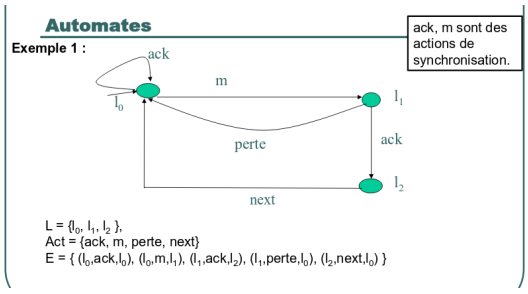


Figure 2.3: Exemple initiale d'automate

- E $\subseteq (L_1 x L_2) x (Act_1 \cup Act_2) x (L_1 x L_2)$ et
- $((I_1, I_2), a, (I'_1, I'_2)) \in E$ ssi
  - a $\in Act_1 - Act_2 \wedge (I_1, a, I'_1) \in E_1 \wedge I_2 = I'_2 \vee$
  - a $\in Act_2 - Act_1 \wedge (I_2, a, I'_2) \in E_2 \wedge I_1 = I'_1 \vee$
  - a $\in Act_1 \cap Act_2 \wedge (I_1, a, I'_1) \in E_1 \wedge (I_2, a, I'_2) \in E_2$

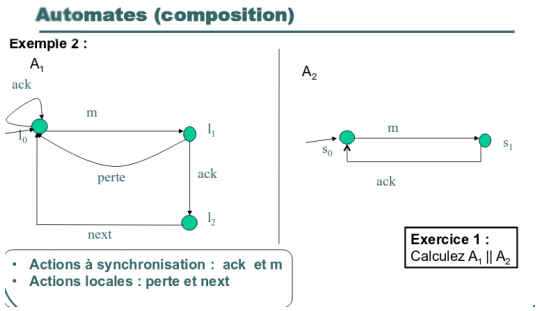


Figure 2.4: Exemple d'une composition de deux automates

représentent le produit entre deux automates. Il prend en compte des actions des deux automates (synchronisation) versus les actions locals à chaque automates.

2.3 Automates Temporisé

On peutrajouter des contraints temporelles. Lorsqu'un message m est émis, m est déclaré perdu si aucun acquitetement n'est recu avant les 5 unités de temps qui suivent l'émission et un acquittement ne peut être recu qu'après 1 unité de temps.

Le modèle d'automates temporisé est une extension du modèle d'automates traditionelle qui prend en considération ces contraintes temporelles.

On distingue deux sémantiques pour le domaine temporel. on peut le faire en temps discret ou en temps continue. Normalement on vérifie les systèmes en temps continue parce qu'il n'y a aucune fréquence d'échantillonage qui peut théoriquement attraper toute les différentes possibilité d'arrivé d'événement.

On associe normalement des horloges (qui sont les seules variables continus). Les horloges évoluent de facon uniforme avec le temps. Elle peut être mise à zéro au franchissement d'un arc pour commencer à compter le temps à partir d'un événement bien précis.

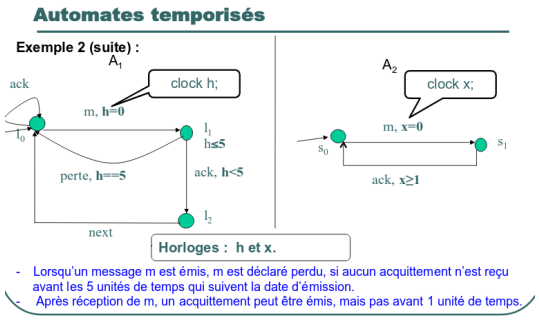


Figure 2.5: L'automate composé sous forme d'automate temporisés

Il suffit d'imposer des contraintes sur les horloges pour modéliser les aspect temporelles des automates.

Noter que d'avoir un horloge sur un état signifie qu'on a une contrainte pour rester dans un état. l'absence d'horloge signifie qu'il n'y a aucune limites de temps.

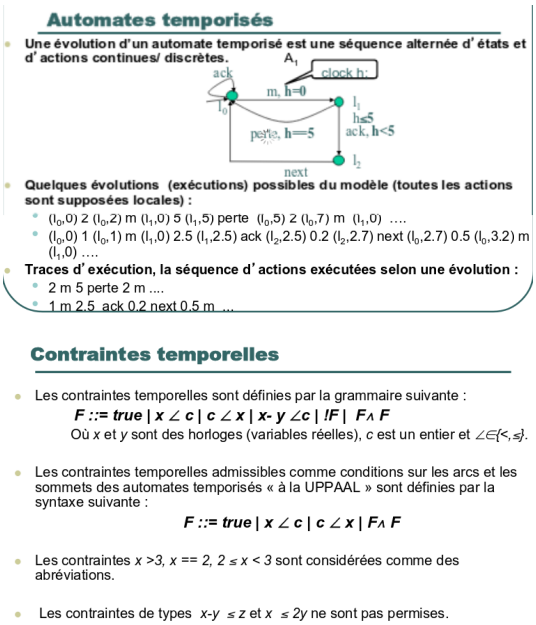
Une évolution d'un automate temporisé est une séquence alterné d'états et d'actions continues-discrètes.

On peut définir une contrainte temporelle par la grammaire de la slide 16

ou x et y sont des horloge, c est un entier et $\angle \in \{\leq, \leq\}$

2.3.1 Contraintes temporelles

Soit un ensemble fini d'horloge



une valuation v de H est une fonction qui associe à chaque horloge de H une valeur réelle positive ou nulle. R^+.

soit F une contrainte temporelle sur H et v une valuation de H, dh $\in R^+$ et $H' \in H$

- f(v) représente la valeur de vérité de f pour v. v satisfait la condition f ssi f(v) est vraie.
- f est dite consistante satisfiable ssi il existe une valuation de H qui satisfait f.

Soit H un ensemble fini d'horloge, f une contrainte temporelle sur H, v une valuation de H, dh $\in R^+$ et $H' \subseteq H$

- on désigne la valuation v+dh la valuation définie par :
  $\forall$x $\in$ H, (v + dh)(x) = v(x) + dh
- v[H'= 0] est la valuation obtenue en remplaçant la valeur de chaque horloge de H' par 0.
- on désigne par f[H'=0] la formule obtenue en éliminant par substitution les horloges de H' et en ajoutant ensuite la contrainte x=0, pour toute horloge x de H'.

Exemple

- **f1 est-elle consistante?**
  Le contraintes sont rencontrés dans f1 et f2. Alors c'est consistant
  v(x) = v(y) = 2
  v(x) = v(y) = v(2) = 2
- **Donnez la représentation graphique**
  x est compris parmis les bornes. On doit faire les contraintes vertical/horizontal pour x et y. On rajoute ensuite les contraintes diagonale.

2.3.2 Définition et représentation graphique

Pour une représentation graphique, on modifie une condition d'un automate

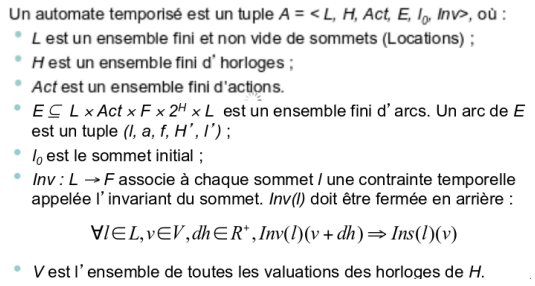


Figure 2.6: Définition et représentation graphique

On défini Inv comme étant l'invariant. Elle associe une contrainte temporelle applée L à chaque sommet. Inv(l) doit être fermée en arrière.

$\forall \in$ L, v $\in$V, dH $\in$...

Le principe est que certaine formulation d'invariant peuvent donner des comportements inattendus.

Un état du modèle est défini par un couple composé d'un sommet I et d'une valuation v des horloges de H qui satisfait Inv(I), i.e.: Inv(l)(v) est vraie.

on peut seulement prendre une transition si la valuation de sa transition est vraie.

2.3.3 Sémantique opérationelle

Un système de transitions est un tuple T ¡Q, Q_0, Et, Rt¿

- Q est un ensemble non vide d'états
- Et est un ensemble d'étiquettes
- Rt $\subseteq$ Q x Et x Q est une relation de transitions entre les états
- Q0 $\subseteq$ Q est un ensemble d'états initiaux

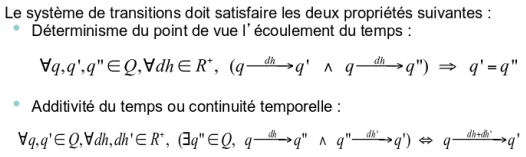


Figure 2.7: Propriétés d'un système de transition

Le système de transitions d'un automate temporisé

A = ¡L, H, Act, E, I0, Inv¿

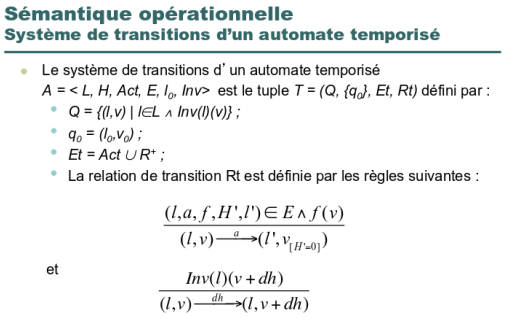


Figure 2.8: d'autres propriétés d'un système de transition

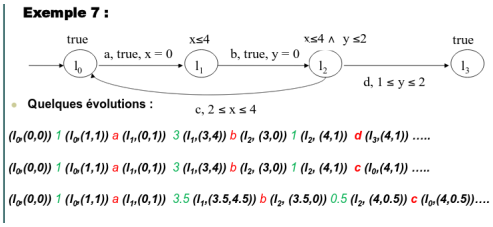


Figure 2.9: Exemple d'Évolution

Note, il n'existe pas d'évolution qui supporte ABA vu que l'invariant de l1 éjecte en dehors de l1 à x≥2

2.3.4 Automates bien temporisés

Une évolution inif est à temps divergent si la somme des temps de séjours dans ses locations est infinis, convergent quand ses finis.

Temps convergent et zénon

la suite peu avoir une infinité d'actions, sauf que l'invariant de on ne supporte pas une valeur d'horloge en haut de 2.

Pour un non-zénon, dans tout cycle élémentaire de transitions discrète au moins une horloge est remise à zéro et comparé à une orne inférieur strictement positive.

un état a un verrou temporel (timelock) si il n'a aucune évolution à temps divergent

dans la deuxième image, aucune possibilité d'été pour une valeur de x supérieur à 2

Un automate temporisé est à l'état bloqué si aucune action de Act ne peut être exécuté à partir de cet état (état bloqué).

un automate est bien temporisé si: pour chaque état accessible q (l,v) il existe une action a de Act ou un réel positif dh tel que

et il est non zénon (toutes ses évolutions sont non zénon).

2.4 Composition d'automates temporisé

On fait comme au début du cours, on fait l'union des actions, horloge et états.. c'est gros.

Les composition d'automates se généralise à un ensemble d'automate..

Pour synchroniser deux automates, il faut tout simplement mettre la même actions.

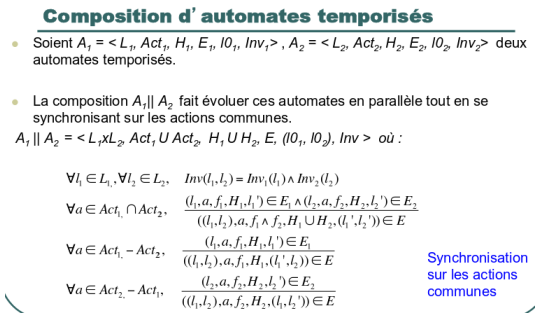


Figure 2.10: Propriété d'une composition d'automates temporisé

2.5 Automate temporisé à la UPPAAL

2.6 Automates hybride, à chronomêtre, à couts