HAL open science


# Extension temps réel d'AltaRica


Claire Pagetti



## ▶ To cite this version:

Claire Pagetti. Extension temps réel d'AltaRica. Génie logiciel [cs.SE]. Ecole Centrale de Nantes (ECN); Université de Nantes, 2004. Français. NNT : . tel-00006316

## HAL Id: tel-00006316
## https://theses.hal.science/tel-00006316

Submitted on 24 Jun 2004

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

**École Centrale de Nantes** **Université de Nantes**

École Doctorale
Sciences et Technologies
de l'Information et des Matériaux

Année 2004 N° BU : ED 0366 - 141

Thèse de Doctorat

Diplôme délivré conjointement par
L'École Centrale de Nantes et l'Université de Nantes

Spécialité : Automatique et Informatique Appliquée

Présentée et soutenue publiquement par :

Claire PAGETTI

le 20 avril 2004
à l'École Centrale de Nantes, Nantes

Titre

# Extension temps réel d'AltaRica

Jury

| | | |
|---|---|---|
| Président | André Arnold | (PR, Univ. Bordeaux 1, LaBRI) |
| Examinateur et co-encadrant | Franck Cassez | (CR, CNRS, IRCCyN) |
| Examinateur | Jean-Pierre Elloy | (PR, ECN, IRCCyN) |
| Examinateur et rapporteur | François Laroussinie | (MCF, ENS Cachan, LSV) |
| Examinateur et rapporteur | Antoine Rauzy | (CR (HDR), CNRS, IML) |
| Examinateur et directeur de thèse | Olivier Roux | (PR, ECN, IRCCyN) |
| Examinateur | Robert Valette | (DR, CNRS, LAAS) |
| Rapporteur | Béatrice Bérard | (MCF (HDR), ENS Cachan, LSV) |

Directeur de thèse : Olivier Roux
Co-encadrant : Franck Cassez
Laboratoire : IRCCyN (UMR CNRS 6597), 1, rue de la Noë - BP 92101
44321 Nantes cedex 3, FRANCE.
N° ED 0366 - 141

# Remerciements

Je remercie chaleureusement André Arnold d'avoir accepté de présider le jury de cette thèse. Je lui suis reconnaissante de s'être intéressé à mon approche sur le formalisme AltaRica et de m'avoir remis sur le droit chemin en tant que détenteur du "dogme AltaRica". Je suis très honorée de la participation de chacun des membres du jury de cette thèse. J'ai eu la chance de les croiser au cours de ma vie de doctorante et chacun à sa manière m'a apporté des éclaircissements sur diverses questions. Je remercie particulièrement Béatrice Bérard, François Laroussinie et Antoine Rauzy d'avoir accepté d'être rapporteurs de cette thèse, Jean-Pierre Elloy et Robert Valette d'avoir accepté d'examiner ce travail.

Je voudrais exprimer toute ma gratitude envers Olivier Roux et Franck Cassez pour m'avoir encadrée, encouragée et soutenue. Olivier Roux est un directeur de thèse exigent qui laisse à ses étudiants une grande liberté tout en les concentrant sur un but précis. Travailler avec Franck Cassez fut une expérience riche et passionnante, il m'a appris beaucoup de choses que je ne pourrai énumérer mais tout particulièrement la rigueur et l'objectivité. Je n'oublierai jamais nos après-midi travail dans le bureau 521.

Je suis très heureuse d'avoir rencontré et de m'être intégrée dans le groupe AltaRica. D'années en années, une réelle dynamique et une grande cohésion se sont créees, cela m'a également insufflé du courage. Je remercie tout particulièrement Alain Griffault pour son humanité et ses excellents conseils.

Cette thèse ne serait pas ce qu'elle est sans Aymeric Vincent, mon *frère AltaRica* pour reprendre ses termes. Je le remercie entre autre pour tous nos moments de joie et d'intense collaboration.

La vie du laboratoire joue sur la qualité du travail et j'ai eu la chance d'appartenir à l'Irccyn puis au Labri. Je remercie les équipes temps réel/Moves et MVTsi pour leur accueil amical. Merci notamment à Frédéric Herbreteau, Michaël Adélaïde et Grégoire Sutre pour leurs remarques judicieuses.

Je tiens à remercier mes co-bureaux : Anne Manoury, Aude Maitrot, Yannick Lenir, Richard Moot et Roland Barriot pour avoir supporté mes frasques mais surtout pour leur soutien et leurs conseils. Je remercie les informaticiens thésards ou docteurs qui n'ont pu empêcher des crash et formattages intempestifs de disques durs, merci à Seb, Didier, Mik et Guillaume. Je remercie les thésards sans qui les pauses, les soirées, Noirmoutier, Attack et les Stooges n'auraient pas eu le même sens. Merci à Kristell, Fred, Pierrick, Cédric, Hélène, Pierre M, Imad, Julien, Franck, Mehdi et les autres.

Enfin, j'aimerais exprimer ma reconnaissance à ceux qui me soutiennent au quotidien et principalement Axelle : je remercie ma famille de n'avoir jamais douté de moi et de m'avoir donné toute ma force. Je remercie mes amis (outre ceux déjà cités) de Suède, de Belgique, de France et de Navarre.

# Table des matières

**1 Introduction** **9**
1.1 Le contexte . . . . . 9
1.2 Le langage de modélisation AltaRica . . . . . 11
1.3 Contenu du document . . . . . 12
1.3.1 Introduction du temps dans AltaRica . . . . . 12
1.3.2 Introduction des priorités temporelles . . . . . 13
1.3.3 Traduction symbolique des modèles AltaRica temps réel . . . . . 14
1.3.4 Travaux existants . . . . . 14
1.4 Structure du document . . . . . 16

**I Les points de départ** **17**

**2 L'atelier logiciel AltaRica** **19**
2.1 Le modèle des automates à contraintes . . . . . 19
2.1.1 Des systèmes réels . . . . . 19
2.1.2 Préliminaires . . . . . 20
2.1.3 Notations . . . . . 20
2.1.4 Les systèmes de transitions : variations sur un même thème . . . . . 21
Les automates à variables . . . . . 21
2.1.5 Les automates à contraintes . . . . . 23
2.1.6 Les systèmes de transitions interfacés . . . . . 25
Les flux ou l'indéterminisme . . . . . 25
La bisimulation . . . . . 25
2.1.7 Les priorités sur les événements . . . . . 26
2.2 Le langage AltaRica . . . . . 28
2.2.1 La syntaxe AltaRica . . . . . 28
2.2.2 L'interface des composants . . . . . 29
2.3 La hiérarchie . . . . . 30
2.3.1 Les nœuds AltaRica . . . . . 31
Synchronisation particulière : le broadcast . . . . . 31
Sémantique des nœuds . . . . . 32
2.3.2 Réécriture d'un nœud AltaRica en composant AltaRica . . . . . 33
2.4 Présentation générale de l'atelier logiciel AltaRica . . . . . 35
2.4.1 Les langages AltaRica . . . . . 35
2.4.2 Des outils pour des modèles AltaRica . . . . . 37
2.4.3 Des études de cas avec AltaRica . . . . . 37

**3 Rappels sur les automates temporisés et sur les automates hybrides 39**
3.1 Les automates temporisés . . . 40
3.1.1 Le temps et les horloges . . . 40
3.1.2 Syntaxe et sémantique des automates temporisés . . . 42
3.1.3 Le modèle des automates temporisés : avantages et inconvénients . . . 44
3.1.4 Des outils pour les automates temporisés . . . 45
UPPAAL . . . 45
KRONOS . . . 46
CMC . . . 47
3.2 Les automates hybrides . . . 47
3.2.1 Syntaxe et sémantique des automates hybrides . . . 48
3.2.2 Composition d'automates hybrides . . . 49
3.2.3 Décidabilité dans les automates hybrides . . . 50
3.2.4 Les outils pour les automates hybrides . . . 51
HYTECH . . . 51

**II Le langage AltaRica temps réel 53**

**4 Le langage Timed AltaRica 55**
4.1 Une extension temporisée du langage AltaRica . . . 55
4.1.1 Extension syntaxique temporisée . . . 55
Les composants Timed AltaRica . . . 56
La grammaire de Timed AltaRica . . . 57
4.1.2 Systèmes de transitions temporisés interfacés . . . 57
Les valeurs continues de flux . . . 58
4.1.3 La bisimulation temporisée . . . 59
4.1.4 Sémantique des composants temporisés . . . 61
4.2 Les priorités temporelles . . . 63
4.2.1 La faible précondition . . . 64
4.2.2 L'urgence . . . 65
L'urgence dans Timed AltaRica . . . 65
Encodage syntaxique de l'urgence . . . 66
L'urgence dans les STTI . . . 69
4.2.3 Cas général : les priorités temporelles . . . 73
Les priorités temporelles dans Timed AltaRica . . . 74
Les priorités temporelles dans les STTI . . . 74
Résolution syntaxique des priorités temporelles . . . 77
4.3 La hiérarchie dans Timed AltaRica . . . 80
4.3.1 Les nœuds Timed AltaRica . . . 80
Syntaxe des nœuds temporisés . . . 80
Sémantique des nœuds Timed AltaRica . . . 81
Broadcast et priorité temporelle . . . 82
4.3.2 Modularité de Timed AltaRica . . . 83
Compositionnalité et réutilisabilité . . . 83
Réécriture d'un nœud temporisé en un composant temporisé . . . 84
4.4 Equivalence Timed AltaRica et automate temporisé . . . 87

4.4.1 Traduction d'un automate temporisé en un programme Timed AltaRica . . . 87
4.4.2 Traduction d'un modèle Timed AltaRica en un automate temporisé . . . . . 87
Algorithme sémantique : Timed AltaRica → automate temporisé . . . . . . . 88
Algorithme symbolique : Timed AltaRica → automate temporisé . . . . . . . 89
4.4.3 Résultats de décidabilité des modèles Timed AltaRica . . . . . . . . . . . . . 91

**5 Hybrid AltaRica 95**

5.1 Une extension hybride des composants AltaRica . . . . . . . . . . . . . . . . . . 95
5.1.1 Extension syntaxique hybride . . . . . . . . . . . . . . . . . . . . . . . . . . . 95
Les composants hybrides . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
La grammaire de Hybrid AltaRica . . . . . . . . . . . . . . . . . . . . . . . . 97
Interface des composants hybrides . . . . . . . . . . . . . . . . . . . . . . . . 97
5.1.2 Sémantique des composants hybrides . . . . . . . . . . . . . . . . . . . . . . . 98
Réduction des flux . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
5.2 Les nœuds hybrides . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103
5.2.1 Sémantique des nœuds hybrides . . . . . . . . . . . . . . . . . . . . . . . . . . 104
5.2.2 Réécriture d'un nœud hybride en un composant hybride . . . . . . . . . . . . 105
5.3 Hybrid AltaRica et les automates hybrides . . . . . . . . . . . . . . . . . . . . . . 108
5.3.1 Traduction d'un automate hybride en un modèle Hybrid AltaRica . . . . . . 108
5.3.2 Traduction d'un modèle Hybrid AltaRica en un automate hybride . . . . . . 108
5.3.3 Résultats de décidabilité des modèles Hybrid AltaRica . . . . . . . . . . . . . 111

**III De la théorie à la pratique 113**

**6 Un outil pour Timed AltaRica : Tarc 115**

6.1 Rappels sur le model-checking . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
6.2 Mec V . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
6.2.1 Fonctionnement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
6.2.2 Structure de données : les BDDs [Bry86] . . . . . . . . . . . . . . . . . . . . 116
6.2.3 Les modules de mise à plat . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
6.2.4 Mise à plat des modèles AltaRica . . . . . . . . . . . . . . . . . . . . . . . . 118
6.3 Tarc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
6.3.1 Fonctionnement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
6.3.2 Abstraction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
6.3.3 Mise à plat des modèles Timed AltaRica . . . . . . . . . . . . . . . . . . . . 122
6.3.4 Génération de l'automate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
6.3.5 Concrétisation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123

**7 Une étude de cas 127**

7.1 Spécification du système . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127
7.2 Modélisation en Timed AltaRica . . . . . . . . . . . . . . . . . . . . . . . . . . . 128
7.3 Vérification sur le système . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 130
7.4 Deuxième modélisation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

**8 Conclusion et perspectives 133**

**A Compléments sur AltaRica 137**

**B Exemple d'une réécriture** **139**

**C La grammaire Timed AltaRica** **143**

**D La grammaire AltaRica hybride** **151**

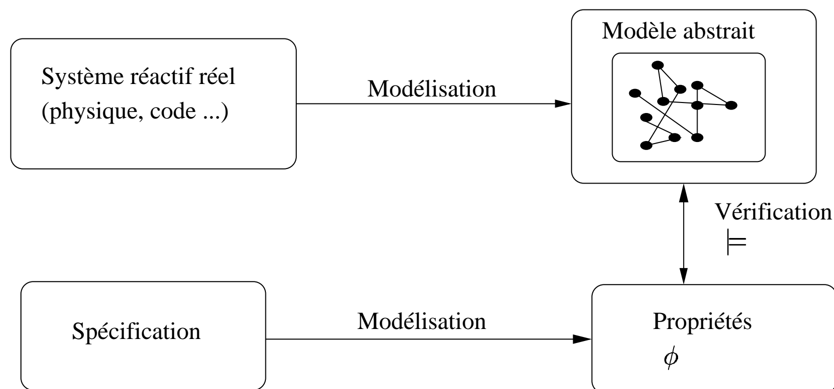
# Chapitre 1

# Introduction

## 1.1 Le contexte

Un *système réactif* [HP85, BG92] est un programme informatique qui répond constamment aux sollicitations de son environnement en produisant des actions sur celui-ci. Ces systèmes sont notamment utilisés pour *le contrôle de processus* [RW89, LHM98], c'est-à-dire qu'un système réactif supervise ou contrôle un procédé dynamique. Cette technique a par exemple été employée dans le contrôle de protocoles [RW92], la surveillance de réactions chimiques [KSF+99] ou la supervision du contrôle aérien [TPS98].

Les systèmes réactifs sont souvent *critiques* [Lap85, Rus94], c'est-à-dire qu'un dysfonctionnement peut entraîner des conséquences graves. On veut donc assurer que le système réactif satisfait certaines propriétés. Comme le système réactif réel est trop complexe pour être étudié formellement tel quel, on l'abstrait en un *modèle* de ce système dans lequel on cherche à conserver tous les comportements utiles à la vérification des propriétés souhaitées. Afin de pouvoir vérifier ces propriétés avec un outil informatique, elles doivent elles aussi être exprimées de manière formelle et pour cela on utilise une logique, par exemple une *logique temporelle* [Pnu77, SBB+99, CGP99].




**Modélisation.** La *modélisation* est la phase d'abstraction d'un système en un modèle. Il s'agit de la première étape dans le processus de vérification. Il faut alors choisir les familles de modèles qui permettent de vérifier les propriétés étudiées. Les critères de choix sont l'*expressivité* (la famille de modèles exprime-t-elle toutes les caractéristiques importantes ?) ; la *décidabilité* et la *complexité* des algorithmes associés à cette famille (si un formalisme est *indécidable*, on ne pourra pas toujours obtenir de résultats exacts dessus) ; les outils disponibles pouvant être appliqués.

Un modèle élémentaire et universel est le *système de transitions,* en effet tout système peut être modélisé par un système de transitions. Un système de transitions est un triplet $(S, E, \rightarrow)$ où $S$ est un ensemble d'états, $E$ est un ensemble d'actions et $\rightarrow$ est un ensemble de transitions étiquetées par $E$. Un état décrit l'état général du système, une action est une opération indivisible du système et une transition est un changement d'état sous l'effet d'une action. Ce formalisme est facile à utiliser, seulement il paraît difficilement concevable de modéliser des systèmes très complexes. Une idée est de créer un ensemble d'opérateurs (comme par exemple le regroupement d'états sous forme de macro états [Har87] ou le raffinage d'états [FH00]) que l'on ajoute au formalisme des systèmes de transitions afin de faciliter la conception de comportements complexes. On obtient alors des *langages de haut niveau* [Har87, CCI88, RBP$^+$91, Der98, CWU98, FH00].

**Les modèles temps réel.** Il existe des familles de modèles particulièrs qui font apparaître le temps explicitement. Un système *temps réel* peut être défini comme suit : "toute activité de traitement d'information ou tout système qui doit répondre à un stimulus extérieur dans un délai déterminé" [You82]. L'évolution d'un tel système est double : des changements discrets et du temps qui passe. Les propriétés qualitatives, *l'événement $e$ est survenu avant $f$*, ne suffisent pas, il faut également pouvoir quantifier des durées, des périodes. Ces systèmes sont généralement à la fois critiques, puisque le non respect des échéances entraînent souvent des conséquences graves, réactifs et *distribués* [LL90], voire embarqués.

**Les modèles hybrides.** Les *systèmes hybrides* [Hen96] permettent de modéliser des systèmes temps réel dont certaines grandeurs évoluent continûment en fonction du temps et de manipuler *quantitativement* ces grandeurs. Le modèle hybride est un système de transitions étendu avec des *variables continues*, c'est-à-dire des variables qui suivent une loi d'évolution en chaque état définie par une équation différentielle. Les transitions sont alors de deux types :

- *discrètes* : il est possible de franchir une transition étiquetée par une action, c'est-à-dire que les variables continues satisfont une *contrainte* de franchissement, puis une fois l'action réalisée les variables continues sont *remises à jour* ;
- *continues* : cela correspond à un écoulement de temps et les variables progressent alors selon leur loi d'évolution. On peut ajouter des *invariants temporels* qui bornent le temps qui "a le droit" de passer.

Une sous famille de systèmes hybrides particulière est celle des *automates temporisés* [AD90, AD94]. Toutes les variables d'un automate temporisé suivent la même loi d'évolution dont la pente est 1 et on appelle ces variables des *horloges.*

**Vérification.** *Vérifier* [Pnu81, CWA$^+$96, CGL94, SBB$^+$99, CGP99] consiste à répondre à la question : *est-ce que le modèle satisfait une certaine propriété ?* Le *model-checking* [McM93, ACD93, CGL94, SBB$^+$99, CGP99] est une technique largement développée depuis des années : un modèle est vu comme un système de transitions et on cherche à prouver qu'il satisfait ($\models$) une certaine propriété ($\phi$). Cette vérification est réalisée par un outil appelé un *model-checker* [McM92, ABC94, AHM$^+$98]. Cette méthode formelle a fait ses preuves dans la détection d'erreurs, par exemple dans des protocoles audio [HS96, BGK$^+$02], des protocoles de gestion de cohérence de caches [PD96]...

**Vérifier les systèmes hybrides.** L'accessibilité n'est en général pas décidable pour un automate hybride sans hypothèse particulière. Il faut alors apporter des restrictions pour définir des sous familles pour lesquelles certaines propriétés sont décidables. Le model-checking s'étend pour

ces sous familles et de nombreux model-checker temporisés ou hybrides existent depuis quelques années [HHWT97, PL00, DOTY95, LL98b].

Dans ce document, nous allons nous intéresser à la partie de modélisation et dans une moindre mesure à la phase de vérification. Nous nous basons et nous étendons temporellement les travaux réalisés autour du langage de description hiérarchique de systèmes *AltaRica* [GLP+98, AGPR00, Poi00, PR99b].

## 1.2 Le langage de modélisation AltaRica

**Bref historique.** Le projet AltaRica, apparu en 1996, est le résultat d'une collaboration industrielle/universitaire allant dans le sens de l'amélioration de l'analyse de systèmes concurrents orientée sûreté de fonctionnement. Il découle de l'unification de travaux de recherche issus de domaines distincts. D'un côté, l'atelier de sûreté de fonctionnement *FIABEX* [HLM94] était dédié à l'évaluation d'architecture système et reposait sur la description comportementale fonctionnelle et dysfonctionnelle des systèmes. Le problème majeur était l'absence d'une sémantique formelle ce qui interdisait toute preuve de correction. D'un autre côté, le LABRI (Laboratoire Bordelais de Recherche en Informatique) développait l'outil *Toupie* [Rau95], un interpréteur de $\mu$-calcul. Ce langage utilise les contraintes pour caractériser les relations entre variables. Les recherches se sont alors portées sur le modèle formel des *automates à contraintes* [BR94, FV94]. Tous ces aspects ont été regroupés et ont donné naissance à l'atelier AltaRica. L'objectif, depuis 1996, est de développer un outil générique permettant l'analyse de systèmes incorporant :

- un langage de spécification de haut niveau AltaRica [GLP+98, Poi00, AGPR00] ;
- un model-checker Mec V [ABC94, Arn95, Vin03a, Vin03b] ;
- des outils de sûreté de fonctionnement (Aralia [ARA96], ...).

Aujourd'hui, le projet AltaRica s'est élargi à un niveau national et est le fruit de plusieurs collaborations : industrielles (la société d'ingénierie GFI, Dassault Aviation, TotalFinaElf, Schneider Electric, France Telecom, Thales systèmes aéroportés) et universitaires (LaBRI, IML, IRCCyN et ONERA).

**Principes de l'atelier AltaRica.** On résume les principes de l'atelier AltaRica dans la Figure 1.1 extraite de [PR99a]. Un modèle AltaRica est transformé en des formalismes de plus bas niveau, comme les systèmes de transitions, et peut ainsi être vérifié selon la méthode de model-checking explicitée précédemment.

**Opérateurs de modélisation AltaRica.** En tant que langage de haut niveau, le langage AltaRica possède plusieurs opérateurs de modélisation. Le langage repose sur le modèle des automates à contraintes [BR94, FV94] et spécifie un système comme un ensemble de variables contraintes par des formules et reliées par une relation de transitions étiquetées. Les interactions entre les sous modèles sont de deux types (représentées en trait continu et pointillé dans la Figure 1.1) : des *synchronisations* entre événements et des contraintes sur des *variables partagées*. Les variables partagées jouent un rôle important en AltaRica : une telle variable peut être remise à jour à tout instant. Les ensembles des événements et des variables partagées d'un modèle AltaRica sont appelés l'*interface* du modèle.

Un autre atout important du langage est la possibilité d'exprimer des *priorités* entre événements. Les priorités sont largement utilisées notamment pour résoudre le problème de l'*ordonnancement* [LL73, Der74] de processus dans un système multitâche.

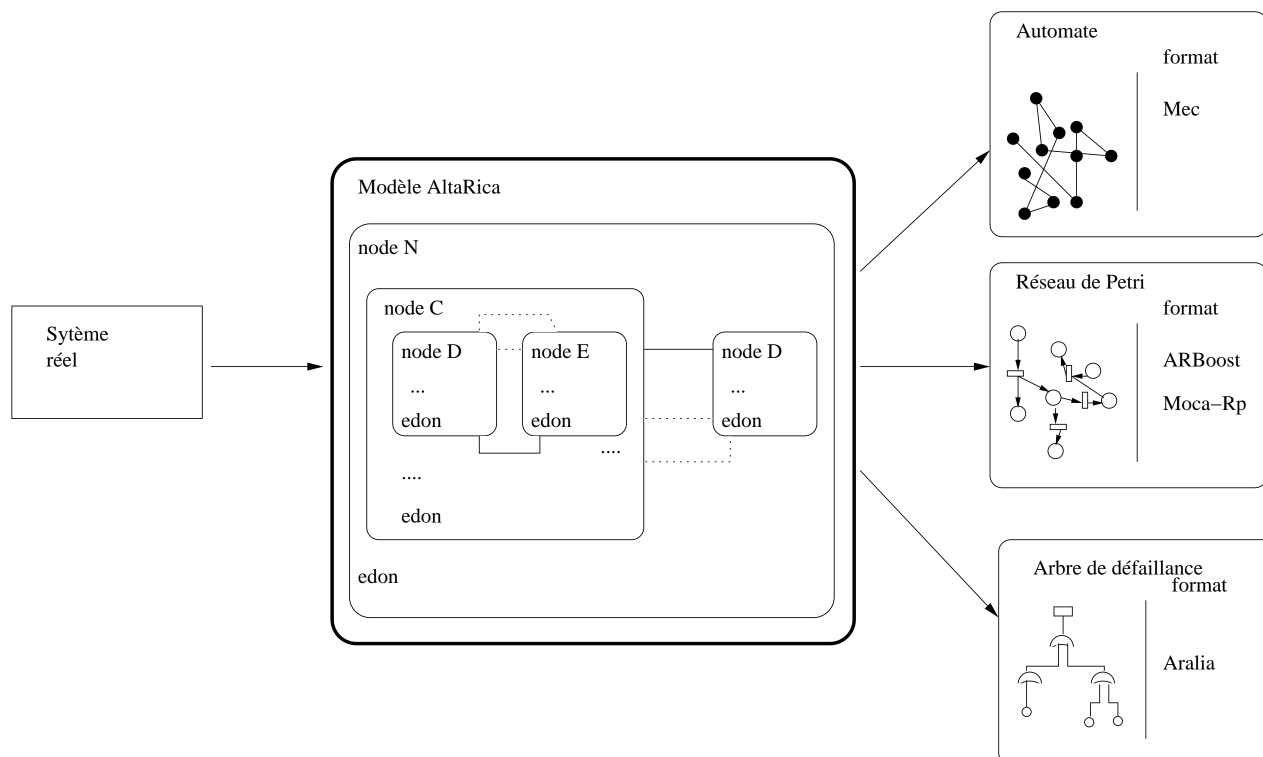



FIG. 1.1 – Atelier AltaRica [PR99a]

**Vérifier les modèles AltaRica.** Mec V est un model-checker dédié aux modèles AltaRica : il prend en entrée des modèles AltaRica et permet entre autre de mettre les modèles à plat et de vérifier des propriétés logiques sur leurs comportements. En particulier, il est possible de vérifier si deux modèles AltaRica ont les mêmes comportements.

L'atelier AltaRica est une réponse à certains aspects du contrôle de processus. Des limites ont été rencontrées dans l'expressivité du modèle et la nécessité d'ajouter des opérateurs de modélisation *temps réel* est apparue.

## 1.3 Contenu du document

Les travaux que nous présentons s'inscrivent dans la modélisation de systèmes temporisés ou hybrides. Nous proposons en outre des moyens d'appliquer des méthodes de vérification formelle sur ces modèles.

### 1.3.1 Introduction du temps dans AltaRica

L'objectif de ce document est de proposer des extensions temps réel des modèles AltaRica. Nous nous intéresserons plus particulièrement à des versions temporisées et hybrides du langage AltaRica. Pour cela, nous nous inspirons des modèles AltaRica et des modèles des automates temporisés avec remise à jour [BDFP00] resp. des modèles AltaRica et des modèles des automates hybrides.

La syntaxe du langage AltaRica est enrichie de façon à manipuler des *variables continues* : on introduit les types ad hoc *analog* et *clock*, on définit les *contraintes linéaires* [Hen96] et les

*contraintes d'horloge* [AD90], les *mises à jour* [BDFP00], les *invariants temporels* [LPY95]. Les variables continues pourront aussi bien être internes que partagées.

Le pouvoir d'expression est différent selon qu'on se place dans le cas hybride ou temporisé. Néammoins leur sémantique sera exprimée à l'aide du même modèle : pour cela nous temporisons la sémantique des modèles AltaRica et nous définissons les *systèmes de transitions temporisés interfacés*. La différence par rapport à AltaRica réside dans l'ajout des transitions temporelles.

Dans les deux cas (temporisé ou hybride), on suppose qu'un système peut réaliser plusieurs actions à la suite en un temps nul. Cette hypothèse est très forte et engendre des comportements difficiles à étudier (*zénons* [HNSY92, HLMR02, JLSE99]) d'une part et peu réalistes d'autre part car il faut un minimum de temps pour effectuer physiquement une action. Néanmmoins, l'ensemble des systèmes vérifiant cette hypothèse contient l'ensemble des systèmes tels qu'il y a un délai minimal entre deux événements. Un modèle AltaRica temps réel est *asynchrone*, c'est-à-dire la durée entre deux événements peut être arbitrairement petite.

Les variables continues partagées, de même que les variables discrètes partagées, peuvent se remettre à jour à tout instant par une transition $\varepsilon$. Cette particularité va influer sur les résultats de décidabilité des modèles.

**Timed AltaRica.** Nous étudions dans un premier temps un extension temporisée d'AltaRica, appelée Timed AltaRica, car ce modèle est plus simple que le modèle hybride.

Nous étendons la hiérarchie d'AltaRica et les opérateurs de façon à construire un langage cohérent. On écrit un procédé de réécriture syntaxique d'un modèle hiérarchique en un modèle équivalent sans hiérarchie, c'est ce que nous appelons la *mise à plat*.

Le langage Timed AltaRica est aussi expressif que les automates temporisés avec mise à jour et transitions inobservables [BGP96]. La décidabilité de l'accessibilité dépend des gardes et des affectations des transitions, de même que pour les automates temporisés avec mise à jour [BDFP00], mais également des *horloges partagées*. Un récaputatif est donné dans le tableau 4.1 page 92.

**Hybrid AltaRica.** L'extension hybride est plus compliquée puisqu'elle requiert la prise en compte des pentes des variables. Les lois d'évolution ne sont spécifiées que pour les variables d'état et celles des variables de flux sont déduites de leur relation avec les variables d'états.

De même que pour Timed AltaRica, nous proposons une extension hybride du langage AltaRica dont nous exprimons la sémantique à l'aide des systèmes de transitions temporisés interfacés. A nouveau, nous obtenons un langage de modélisation hiérarchique de haut niveau.

L'expressivité d'Hybrid AltaRica est celle des automates hybrides linéaires avec conditions de flot rectangulaires [HM00]. La décidabilité des modèles dépend des initialisations sur les transitions [HKPV95, HM00]. Dans Hybrid AltaRica, les variables continues partagées posent des problèmes supplémentaires et la décidabilité est résumée dans le tableau 5.7 page 111.

### 1.3.2 Introduction des priorités temporelles

Nous étendons les priorités d'AltaRica de sorte qu'elles prennent en compte des quantités temporelles. En premier lieu, nous introduisons l'*urgence* [SY96, BST98, BS00]. L'urgence consiste à donner la priorité des pas discrets sur les pas continus. Ce genre de comportements est particulièrement intéressant dans les systèmes critiques : en effet en cas de situation extrême où il faut réagir, l'urgence assure que quelque chose se passera immédiatement.

Nous introduisons également les *priorités temporelles* [BST98, BS00, BGS00] dans Timed AltaRica. Une priorité temporelle permet de modéliser des comportements comme *l'événement e n'est*

*possible que si l'événement $f$ ne peut pas être réalisé dans les $k$ prochaines unités de temps.* Les priorités temporelles peuvent être utilisées dans la synthèse de contrôleur [AGP+99] pour l'ordonnancement temps réel [Alt01, AGS02].

Pour ces deux opérateurs, nous exprimons l'effet sémantique sur les systèmes de transitions temporisés interfacés et nous donnons une traduction syntaxique équivalente sur les modèles Timed AltaRica. On constate alors que ces deux priorités sont conceptuellement différentes : l'urgence modifie les invariants temporels et les priorités temporelles agissent sur les gardes des transitions. L'encodage des priorités est syntaxique et ne modifie en rien l'expressivité du modèle Timed AltaRica.

### 1.3.3 Traduction symbolique des modèles AltaRica temps réel

Pour les deux extensions temporelles étudiées, nous proposons un algorithme symbolique qui termine toujours et qui traduit un modèle AltaRica temps réel en un automate temporisé ou un automate hybride selon le cas. Le modèle est d'abord mis à plat en utilisant la définition de la hiérarchie dans les langages, puis le modèle équivalent sans hiérarchie est traduit en un automate temporisé ou un automate hybride. Ce processus est implanté dans un prototype qui génère pour tout modèle temporisé un automate temporisé écrit dans le format d'un model-checker temporisé. On peut alors vérifier des propriétés sur le système initial.

### 1.3.4 Travaux existants

D'autres travaux ont été réalisés parallèlement aux extensions temps réel d'AltaRica pour construire des langages de haut niveau temporisés et/ou hybrides permettant de faire des vérifications.

Le langage modulaire de spécification hybride le plus élaboré est **Charon** [AGH+00, ADE+01]. Le langage est aussi expressif que la famille des automates hybrides linéaires. La description d'un système se fait en deux temps : d'abord on décrit l'architecture du système puis on modélise le comportement par des composants hiérarchiques (le modèle inhérent aux composants est une machine à états finie). Dans un état, on peut exprimer des invariants et l'évolution des variables est donnée par une équation différentielle ou des contraintes algébriques, on peut exprimer des invariants. Le langage permet :

- l'instanciation de variables, de composants ;
- de masquer des variables, des composants ;
- de composer des composants.

Le langage propose en outre des opérateurs particuliers comme : les exceptions (certaines transitions sont toujours possibles quel que soit le contexte), les restaurations (retourner au dernier état précédant un état de sortie), les transitions gardées, le non déterminisme.

Actuellement, CHARON propose une interface graphique pour décrire les systèmes et un simulateur.

Les langages AltaRica temps réel et CHARON ne proposent pas les mêmes opérateurs de modélisation : les synchronisations sont plus variées (cf. extensions particulières de produit synchronisé) et surtout, on a la possibilité d'exprimer des priorités temporelles dans Timed AltaRica. Enfin, on peut faire des vérifications sur les modèles Timed AltaRica, ce qui n'est pas (encore) le cas pour CHARON.

Hormis Charon, les autres langages de haut niveau temporisés sont moins expressifs que Timed AltaRica. On peut néanmoins citer **Timed Argos** [JMO93, JM95], une extension temporelle d'Argos. Un constructeur de délai est ajouté : ainsi un état temporisé est un état avec un délai à

respecter. Une fois dans l'état temporisé, le système doit absolument quitter cet état avant la fin du délai, soit en faisant une transition spécifiée soit en tirant la transition spéciale *timeout*. Ce qui revient à rajouter une horloge qui est remise à jour à chaque transition et chaque état temporisé respecte un invariant temporel sur cette horloge. De plus, le temps est discret. Timed Argos est strictement moins expressif que les automates temporisés.

**HyCharts** [GSB98] est une extension hybride des Statecharts. Il s'agit donc d'un formalisme visuel réactif, hiérarchique et modulaire pour spécifier des systèmes hybrides. Un système est vu comme un réseau de composants qui communiquent en temps synchrone. Comme pour Argos, on suppose que la sortie est calculée instantanément en fonction de la donnée. Le modèle est celui des *machines hybrides* : pas discrets, pas continus et boucle *feedback* (le système se souvient de l'entrée et de la sortie à l'instant précédent).

**SDL** [CCI88] (Specification and Description Language) est un formalisme de spécification de haut niveau conçu pour la description de systèmes de télécommunications temps réel et réactifs. On spécifie un système en SDL et tout ce qui ne fait pas partie du système est appelé environnement. Un système est vu comme un ensemble d'automates communicant entre eux de manière asynchrone et avec l'environnement avec des files d'attente.

Le problème actuel de SDL est qu'il ne manipule pas le temps quantitatif. En effet, la notion de temps est extérieure à la sémantique d'un système [BGM+01] : aucune hypothèse ne peut être formulée sur le temps et une transition est tirée au bout d'un certain laps de temps dans un état. La notion de timer est importante, un timer mesure un certain laps de temps et quand il y a timeout, il y a envoi d'un message et le processus le reçoit et le prend en compte dans un délai indéterminé. Les transitions d'une spécification SDL sont toutes *lazy*. Actuellement, le temps n'est pas correctement modélisé mais des tentatives sont réalisées dans [BGM+01] pour mettre une sémantique au temps et rajouter des primitives dans le langage pour manipuler le temps. Le modèle est celui des automates à deadline [SY96] mais on ne connaît pas l'expressivité.

**Shift** [DGV96, GK00] est un langage de programmation orienté objet pour décrire des réseaux dynamiques de systèmes hybrides afin de les simuler. Ainsi, les composants peuvent être créés, interconnectés et détruits à tout moment. Le modèle est celui des machines à états finis auquel on ajoute des équations différentielles. Les transitions sont gardées et on peut synchroniser les composants. Un composant est un type dont les sous types héritent des attributs. Un programme Shift est compilé en un programme en C. Cet outil est destiné au contrôle de processus et ne met en œuvre que la simulation.

**UML** [BRJ98] (Unified Modeling Language) est une méthode de modélisation orientée objet, originellement développée à la Rational Software Corporation. UML rassemble plusieurs formalismes graphiques orienté objet permettant d'appréhender un problème et de le modéliser. Cette méthode est devenue un standard, seulement de nombreuses limites empêchent son utilisation, notamment pour les systèmes temps réel [BHH+97, DP99a]. L'absence de sémantique formelle décrivant le comportement des modèles est une carence majeure de la méthode : aucune vérification formelle n'est envisageable, non plus qu'aucune simulation. Pour aborder le temps réel, on rencontre encore d'autres lacunes [DP99a] : il est impossible d'avoir une vue globale de la dynamique du système, le parallélisme pour les systèmes de grande taille est difficile à gérer, il n'existe aucune représentation simple des dépendances temporelles, enfin dans sa version actuelle, UML ne peut exprimer de relations quantitatives temporelles. Plusieurs études [DP99b, FM02, DM99] sont menées pour remédier à ces problèmes, mais aucun consensus n'est encore établi.

## 1.4 Structure du document

La **partie** I de ce document consiste à présenter les fondations et les bases sur lesquelles reposent les travaux d'extension des modèles AltaRica. Dans le chapitre 2, nous présentons la syntaxe et la sémantique du langage AltaRica, il s'agit d'un résumé succinct de la thèse de Gérald POINT [Poi00]. Dans le chapitre 3, nous rappelons les notions sur les automates temporisés et les automates hybrides dont nous avons besoin dans la suite.

La **partie** II regroupe l'ensemble des résultats théoriques exhibés en réalisant deux extensions temps réel du langage AltaRica. Nous proposons dans le chapitre 4 une étude exhaustive de l'extension temporisée du langage, Timed AltaRica. L'extension hybride est formellement étudiée dans le chapitre 5.

Enfin, la **partie** III présente les résultats pratiques consécutifs au travaux théoriques précédents. Nous présentons l'outil Tarc dans le chapitre 6 implémentant des algorithmes rencontrés dans le chapitre 4. Après des rappels sur le model-checking, nous montrerons comment transposer des propriétés logiques d'un modèle étendu AltaRica vers les model-checker que nous utilisons. Enfin, le chapitre 7 est dévolu à une étude de cas détaillée.

# Première partie

# Les points de départ

# Chapitre 2

# L'atelier logiciel AltaRica

L'objectif de ce chapitre est d'introduire le formalisme AltaRica [GLP+98, AGPR00, Poi00]. Une fois rappelées les notations et définitions générales utilisées tout au long du manuscrit, nous résumons les recherches menées autour d'AltaRica depuis quelques années. Nous présentons ensuite le modèle inhérent à tous les modèles AltaRica : les automates à contraintes [BR94]. Enfin, nous rappelons les étapes de la construction syntaxique et sémantique du langage de description AltaRica [Poi00].

## 2.1 Le modèle des automates à contraintes

### 2.1.1 Des systèmes réels

Dans la Figure 2.1, on représente un système d'extraction de pétrole sur une plateforme pétrolière. Cet exemple est une simplification d'étude de disponibilité de production [].

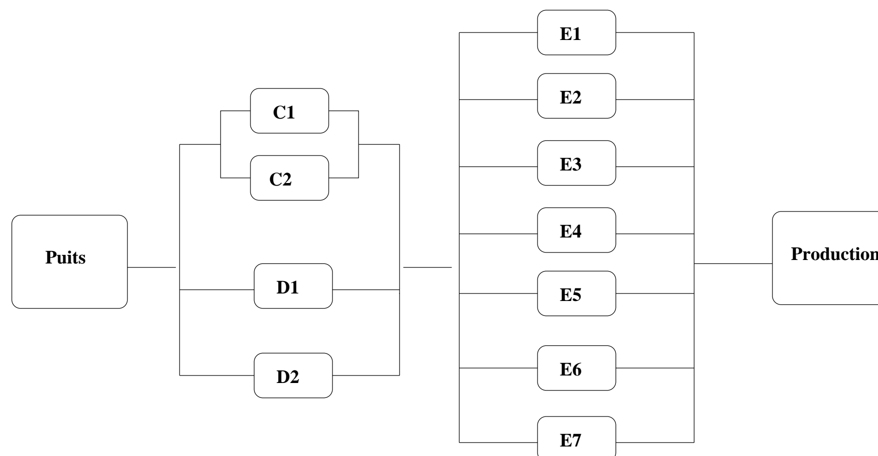
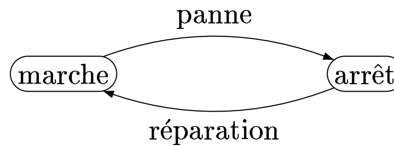



FIG. 2.1 – Schéma d'un système d'extraction de pétrole

Les composants (excepté $C_2$) ont tous deux comportements possibles : soit ils fonctionnent soit ils sont en panne. $C_2$ possède un état supplémentaire de repos. Ainsi, ces composants peuvent être représenté sous forme d'états/transitions.




Ces différents composants sont mis en séries, les premiers composants ($C_i$ et $D_i$) font un premier traitement puis le deuxième est réalisé par les $E_i$. Ainsi, pour spécifier, le système complet il faut pouvoir faire le produit des composants.

Si $C_1$ tombe en panne, le composant redondant $C_2$ prend le relais et lorsqu'un composant ($C_2$, $D_i$) tombe en panne, cela a un effet sur la quantité produite. En revanche, lorsque plusieurs composants $E_i$ sont hors d'état de marche, cela a un effet à la fois sur la production mais également sur la vitesse d'extraction au puits.

Lorsque la vitesse d'extraction doit être modifiée, il faut pouvoir propager l'information sur les états des composants. Pour cela, l'idée consiste à introduire des variables partagées appelées *variables de flux*.

Pour réparer les composants, on fait appel à un service technique. On peut à nouveau modéliser un technicien par un automate et on doit pouvoir synchroniser le composant à réparer et le technicien.

D'autres opérateurs

### 2.1.2 Préliminaires

Pour exprimer formellement des modèles, il est nécessaire de se placer dans un cadre mathématique. Les éléments du modèle seront, entre autre, explicités à l'aide de *termes* et de *formules* d'un *langage du premier ordre* [CL93]. Nous définissons dans la section 2.1.3 le langage du premier ordre utilisé pour définir les modèles AltaRica. Puis dans la section 2.1.4 nous rappelons certaines définitions liées aux systèmes de transitions qui seront nécessaires à la construction de nos modèles.

### 2.1.3 Notations

Les éléments de logique ont été largement étudiés et la théorie qui nous intéresse se trouve notamment dans le livre [CL93]. Nous ne rappelons pas ici les définitions classiques des langages.

Dans l'ensemble des articles sur AltaRica [GLP+98, PR99b, PR99a, AGPR00, Poi00] et Timed AltaRica [CPR02, Pag03b], les auteurs considèrent le langage du premier ordre suivant : $\mathcal{L}_Z = \{Z, \mathcal{D}, \mathcal{F}, \mathcal{R}\}$ où :

1. $Z$ est un ensemble dénombrable de variables ;
2. $\mathcal{D}$ est l'ensemble des symboles de constantes et correspond au domaine des variables $Z$. On retrouve cet ensemble dans les articles mentionnés plus haut sous le nom de *domaine de l'automate à contraintes* . Si le domaine est infini, on peut prendre $\mathcal{D} = \mathbb{Z}$, sinon il s'agit d'un intervalle d'entiers ;
3. les seules fonctions considérées sont l'addition et la soustraction. Alors $\mathcal{F} = Z \cup \{+, -\}$ ;
4. les symboles de relation sont les opérateurs de comparaison et les deux prédicats particuliers d'arité 0 *vrai* et *faux*. Alors $\mathcal{R} = \{tt, ff\} \cup Z \cup \{=, <, >, \leq, \geq\}$.

L'ensemble des termes (ou *expressions*) $\mathcal{T}(\mathcal{L}_Z)$ du langage est le plus petit ensemble des mots de $\mathcal{L}_Z$ contenant $Z \cup \mathcal{D}$ stable pour les opérations arithmétiques $+, -$. Ainsi $x + 3$ ou $5x$ sont des termes.

L'ensemble des formules $\mathcal{F}(\mathcal{L}_Z)$ est le fondement même du travail puisqu'il permettra de "formuler" des propriétés, des contraintes ou des conditions. Dans notre langage $\mathcal{L}_Z$, les formules atomiques

sont engendrées par les prédicats de $\mathcal{R}$ et sont de la forme $x+y=z-3$, $x \leq y+20$, $tt$. Les formules sont générées à partir de ces formules élémentaires, des connecteurs et des quantificateurs. Ainsi $\exists x.\ x+y=z-3 \vee x \leq 20$ est une formule.

Une variable $z$ dans une formule $F$ est *libre* si elle n'est sous l'emprise d'aucun quantificateur. On dit qu'elle est *liée* sinon. On note pour toute formule ou tout terme $f$, $vlib(f)$ l'ensemble de ses variables libres.

Il est parfois utile de réécrire une formule sous une forme équivalente plus exploitable dans certains contextes. On rappelle que toute formule est logiquement équivalente à au moins une formule sous forme normale disjonctive [CL93].

Enfin, pour notre travail sémantique, il est nécessaire de définir une réalisation de notre langage et d'interpréter les termes et les formules. On choisit comme base de réalisation $\mathcal{D}$, les symboles de fonction sont translatés dans dans cette base, i.e. $[\![m_1 \sim m_2]\!] = [\![m_1]\!] \sim [\![m_2]\!]$ avec $\sim \in \{+,-\}$. La réalisation d'une formule est un sous ensemble de $\mathcal{D}^Z$ avec $[\![tt]\!] = \mathcal{D}^Z$ et $[\![ff]\!] = \emptyset$. Par exemple, $[\![x+3 \leq 4]\!] = [\![x \leq 1]\!] = ]-\infty, 1] \times \mathcal{D}^{Z \setminus \{x\}}$. La réalisation d'un terme est une fonction de $\mathcal{D}^Z \rightarrow \mathcal{D}$. Ainsi $[\![x_i + x_j + 3]\!]$ est la fonction $(n_1, \cdots, n_i, \cdots, n_j, \cdots, n_p) \rightarrow n_i + n_j + 3$.

### 2.1.4 Les systèmes de transitions : variations sur un même thème

On considère un système de transitions $(S, E, \rightarrow)$ avec $S$ un ensemble d'états, $E$ un ensemble d'actions et $\rightarrow \subseteq S \times E \times S$ une relation de transitions.

On peut déterminer un *ensemble d'états initiaux* $S_0 \subseteq S$. Alors tout comportement $s_0 \rightarrow^{\alpha_1} s_1 \rightarrow^{\alpha_2} \cdots \in \Sigma$ commence à partir d'un état initial, c'est à dire $s_0 \in S_0$. De même, on peut déterminer un *ensemble d'états finaux* $F \subseteq S$. Alors tout comportement fini $s_0 \rightarrow^{\alpha_1} s_1 \cdots \rightarrow^{\alpha_n} s_n$ termine dans un de ces états, c'est à dire $s_n \in F$ et tout comportement infini doit vérifier une certaine contrainte (par exemple passer infiniment souvent par l'ensemble $F$ dans le cas des systèmes de transitions de Büchi).

Un état $s_n$ est *accessible* s'il est atteignable par un comportement c'est à dire $\exists s_0 \rightarrow^{\alpha_1} \cdots \rightarrow^{\alpha_n} s_n \in \Sigma$.

On peut ajouter une *action particulière* $\boldsymbol{\varepsilon}$ interne au système et non observable : i.e. $\forall s \in S, (s, \varepsilon, s) \in \rightarrow$ et $\forall s' \in S, (s, \varepsilon, s') \in \rightarrow \Longrightarrow s' = s$. L'expressivité, en terme de comportements, reste identique si le système est muni de la $\varepsilon$-transition ou non. Dans la suite, on notera l'ensemble des événements $E = E_+ \cup \{\varepsilon\}$ tel que $\varepsilon \notin E_+$.

Si l'ensemble des états $S$ est fini, on parle alors d'*automate* et on peut le représenter graphiquement.

Les systèmes de transitions se composent entre eux et sont stables par composition. Soient $\mathcal{S}_i = (S_i, E_i, \rightarrow_i)$, $i = 1, \cdots, n$, $n$ systèmes de transitions, soit $I \subseteq E_1 \times \cdots \times E_n$ une *contrainte de synchronisation*, on appelle *produit synchronisé à la Arnold-Nivat* [Arn92, Arn94] des $\mathcal{S}_i$ par rapport à $I$ le système de transitions $\mathcal{S} = (S, E, \rightarrow)$ avec $S = S_1 \times \cdots \times S_n$, $E = E_1 \times \cdots \times E_n$ et $(\langle s_1, \cdots, s_n \rangle, \langle e_1, \cdots, e_n \rangle, \langle s'_1, \cdots, s'_n \rangle) \in \rightarrow \Longleftrightarrow \langle e_1, \cdots, e_n \rangle \in I \wedge \forall i \in [1, n], (s_i, e_i, s'_i) \in \rightarrow^i$. Si la contrainte $I = E_1 \times \cdots \times E_n$, on parle de *produit libre* des systèmes de transitions.

#### Les automates à variables

On peut introduire un ensemble fini de variables discrètes $Z$ dans le modèle des systèmes de transitions. De telles variables facilitent la modélisation des systèmes réels. Une *valuation* est une application $Z \rightarrow \mathcal{D}$ qui associe à chaque variable une valeur dans le domaine de définition. On note l'ensemble des valuations $\mathcal{D}^Z$. Quand on applique une valuation à un terme, on obtient un élément

constant de $\mathcal{D}$. Quand on applique une valuation à une formule, on obtient une valeur de vérité, c'est à dire le prédicat vrai ou faux.

On peut alors définir des *transitions gardées* : le franchissement d'une transition n'est possible que si une condition sur les variables, appelée *garde*, est vérifiée. Formellement, une *garde* $g$ est une formule sur $Z$, i.e. $g \in \mathcal{F}(\mathcal{L}_Z)$. Ainsi, une garde est vérifiée pour une valuation des variables $v$ si la formule est satisfaite : $g(v) = tt$, ou encore $v \models g$ (sinon $g(v) = ff$).

Les valeurs des variables peuvent être modifiées lors d'une transition par une *affectation*. Une variable prendra comme valeur une nouvelle expression sur $Z$. Une affectation est une application de $Z$ dans $\mathcal{T}(\mathcal{L}_Z)$.

Un état d'un automate à variables correspond à une paire $(s, v)$ où $s \in S$ et $v$ est une valuation de chacune des variables. De fait, cette notation est un raccourci pour exprimer le produit synchronisé de plusieurs automates puisqu'une variable est elle-même un automate. Ensuite, la contrainte de synchronisation retranscrit le lien entre l'action, la garde et l'affectation.

On donne le système de transitions associé à une variable entière dans la Figure 2.2. Ce système de transitions $(Q, E, \to)$ est infini et est donné par $E = \{k | k \in \mathbb{Z}\}$, $Q = \mathbb{N}$ et $(i, k, j) \in \to$ si, et seulement si $i + k = j$.

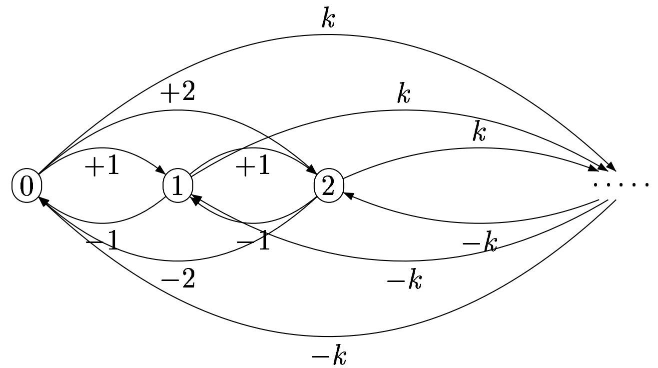



FIG. 2.2 – Représentation d'une variable entière sous forme d'automate

**Exemple 1** *On considère l'automate de la Figure 2.3. Il correspond au produit synchronisé de la variable $k \in [0, 1]$ et de l'automate de la Figure 2.3 sans les conditions sur $k$. La contrainte est $I = \{\langle +1, e\rangle, \langle -1, f\rangle\}$.*

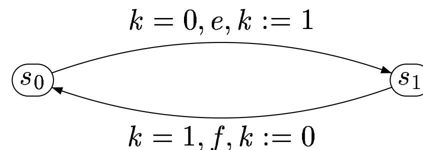



FIG. 2.3 – Exemple d'un automate à variables

Nous avons rappelé des généralités et des notations qui seront régulièrement mentionnées dans la suite. Nous pouvons à présent introduire le modèle des automates à contraintes.

### 2.1.5 Les automates à contraintes

Au milieu des années 90 dans la mouvance des automates temporisés, plusieurs recherches ont porté sur les problèmes de spécification et de vérification de modèles comportant un nombre éventuellement infini de paramètres entiers. L'introduction de paramètres entiers revient à représenter de manière plus concise, et parfois même finie, un système de transitions comme nous l'avons vu pour les automates à variables dans la partie 2.1.4, page 21. La taille réelle du système dépend donc du nombre de variables et de la taille de celles-ci.

Dans [FV94], Laurent FRIBOURG et Marcos VELOSO-PEIXOTO ont défini les *automates généralisés avec contraintes arithmétiques.* Cette approche est contemporaine des travaux réalisés sur le modèle des automates à contraintes introduits par Srécko BRLECK et Antoine RAUZY dans [BR94]. Ces deux formalismes sont équivalents même s'ils diffèrent quelque peu. Un automate généralisé avec contraintes arithmétiques est défini par un ensemble d'états qui coïncident avec des valuations des variables. Il n'y a pas de contrainte globale, puisqu'elle est implicite : les valeurs admissibles correspondent à l'ensemble des états, et de même il n'y a pas d'affectation puisque le but d'une transition est sa nouvelle valuation. En revanche, il y a des gardes qui contraignent les actions en fonction de leur source. Dans un automate à contraintes, on n'exprime pas explicitement les états et on manipule les variables avec des formules. Un automate généralisé avec contraintes arithmétiques est à mi-chemin entre un automate à contraintes et sa sémantique.

Les résultats [FV94] obtenus sur les automates généralisés avec contraintes arithmétiques s'appliquent avec une légère adaptation aux automates à contraintes. On peut ainsi réaliser les opérations de base des automates : intersection et mélange de deux automates généralisés avec contraintes arithmétiques, complétion et déterminisation d'un automate généralisé avec contraintes arithmétiques. Enfin, le problème de l'inclusion de langage pour deux automates généralisés avec contraintes arithmétiques est décidable lorsque les formules ne contiennent que les symboles $=$, $\neq$, $<$.

Nous présentons une version plus tardive des automates à contraintes où les variables de *flux* ont été introduites. Ces variables particulières interviennent en prévision de construction de modèles hiérarchiques. Les variables de flux correspondent à des variables visibles par plusieurs automates à contraintes dont les valeurs peuvent être contraintes par une formule globale et modifiées à tout instant par une transition $\epsilon$. Ainsi un automate à contraintes anticipe le comportement potentiel d'un autre automate à contraintes auquel il est synchronisé. Ces variables sont nécessaires puisqu'elles influent sur les comportements de l'automate à contraintes, notamment dans la sémantique de la transition $\epsilon$. Nous définissons formellement un automate à contraintes.

On considère un ensemble dénombrable de variables discrètes $Z$. On a vu dans la partie 2.1.4 qu'une valuation est une application $Z \to \mathcal{D}$ et qu'on note l'ensemble des valuations $\mathcal{D}^Z$.

**Définition 1 (Automate à contraintes - version interfacée [Poi00])** *Un* automate à contraintes *(ou* composant*) est un quintuplet $\mathcal{A} = \langle V_S, V_F, E, A, M \rangle$ où :*

1. *$V_S$ et $V_F$ sont deux ensembles disjoints de variables respectivement d'*état *et de* flux *; on note $Z = V_S \cup V_F$ ;*
2. *$E = E_+ \cup \{\varepsilon\}$ est un ensemble d'événements avec $\epsilon \notin E_+$ ;*
3. *$A \in \mathcal{F}(\mathcal{L}_Z)$ est une contrainte particulière sur les variables appelée* assertion *telle que $vlib(A) \subseteq Z$ ;*
4. *$M \subseteq \mathcal{F}(\mathcal{L}_Z) \times E \times \mathcal{T}(\mathcal{L}_Z)^{V_S}$ est une relation de macro-transitions. $(g, e, a) \in M$ est telle que :*

*(a)* $g \in \mathcal{F}(\mathcal{L}_Z)$ *est la* garde,

*(b)* $e \in E$ *est l'événement,*

*(c)* $a : V_S \to \mathcal{T}(\mathcal{L}_Z)$ *est une application qui associe à chaque variable d'état un terme* $a(s)$ *qui dépend de l'origine de la transition.*

*On peut éventuellement préciser une contrainte initiale* $I \in \mathcal{F}(\mathcal{L}_Z)$.

La sémantique d'un automate à contraintes $\mathcal{A} = (V_S, V_F, E, A, M)$ est donnée par le système de transitions $(Q, E, \to)$ tel que : $Q = \{(s,f) \in \mathcal{D}^{V_S \cup V_F} | \ (s,f) \in [\![A]\!]\}$ et $(s,f,e,s',f') \in \to \Longleftrightarrow (s,f,e,s') \in M \ \wedge \ (s',f') \in [\![A]\!]$. Ainsi, un état de l'automate à contraintes est une valuation de toutes les variables et cette valuation doit satisfaire la contrainte $A$. Un état $(s,f)$ est dorénavant appelé *configuration* pour distinguer variables d'état et états. Une transition $(g,e,a)$ est *valide* dans l'état $(s,f) \in \mathcal{D}^{V_S \cup V_F}$ si, et seulement si la garde est vraie $g(s,f) = tt$ et si l'état d'arrivée vérifie l'assertion, i.e. $\exists f' \in F, (a(s,f), f') \in [\![A]\!]$.

**Exemple 2 (Jeu de Marienbad - I)** *On va modéliser pendant la présentation du langage AltaRica un jeu de Nim dit de Marienbad (ou encore Fan Tan ou jeu des allumettes). Cet exemple est extrait de [Gri02]. Ce jeu se joue à deux :*

**disposition :** *on dispose plusieurs paquets d'allumettes sur une table. Initialement, il y a* 4 *tas contenant respectivement* 1, 3, 5 *et* 7 *allumettes ;*

**règles :** *chaque joueur ramasse à tour de rôle autant d'allumettes qu'il le désire (au moins une) dans un des paquets,*

**but du jeu :** *le but est de ne pas ramasser la dernière allumette.*

*Dans ce premier exemple, nous modélisons un paquet d'allumettes : il y a au plus* 7 *allumettes et pour un nombre donné* $n$ *d'allumettes on peut en retirer* $p \leq n$. *On construit le composant* Paquet *: il s'agit de l'automate à contraintes* $\mathcal{P}aquet = (V_S, V_F, E, A, M)$ *avec* $V_S = \{n\}$, $V_F = \{N\}$, $E = \{joue\}$, $A = 0 \leq n \leq 7 \wedge n = N$ *et les transitions sont* $(n > i, joue, n := n - i - 1)$ *avec* $i = 0$ *à* 6. *L'assertion nous donne le domaine* $\mathcal{D} = [0,7]$ *et un état de contrôle correspond à une valuation de* $(n, N)$. *On obtient donc* 8 *états possibles représentés dans l'automate sémantique de l'automate à contraintes fini* $\mathcal{P}aquet$ *donné à la Figure 2.4. L'étiquette* joue *est la même pour toutes transitions et nous les avons omises dans la Figure.*

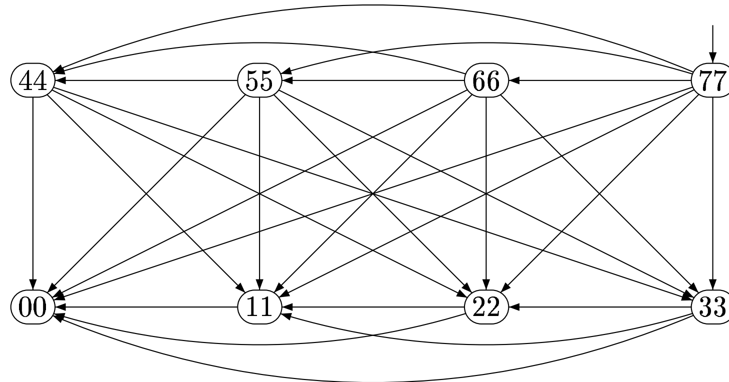



FIG. 2.4 – Représentation de l'automate à contraintes fini $\mathcal{P}aquet$

### 2.1.6 Les systèmes de transitions interfacés

Pour simplifier l'expression sémantique des automates à contraintes, Gérald Point a introduit dans sa thèse [Poi00] des systèmes de transitions particuliers : les *systèmes de transitions interfacés.*

**Définition 2 (Système de transitions interfacé [Poi00])** *Un* système de transitions interfacé *(STI) est un quintuplet* $\mathcal{A} = \langle E, F, S, \pi, T \rangle$ *avec :*

1. $E = E_+ \cup \{\varepsilon\}$ *est un ensemble fini d'événements tel que* $\varepsilon \notin E_+$ *;*
2. $F$ *est un ensemble de valeurs de flux ;*
3. $S$ *est une ensemble de valeurs d'états ;*
4. $\pi : S \to 2^F$ *est une application qui associe à chaque variable d'état* $q \in S$ *toutes les valeurs de flux admissibles en* $q$*. On suppose que* $\forall q \in S,\ \pi(q) \neq \emptyset$ *;*
5. $T \subseteq S \times F \times E \times S$ *est la relation de transitions, elle vérifie :*
   (a) $(s, f, e, s') \in T \Rightarrow f \in \pi(s)$
   (b) $\forall s \in S, f \in \pi(s),\ (s, f, \varepsilon, s) \in T$

*Une* configuration *d'un STI est une paire* $(s, f) \in S \times F$ *telle que* $f \in \pi(s)$*.*

Un STI va exactement incorporer les caractéristiques d'un automate à contraintes fini $\mathcal{A} = (V_S, V_F,\ E, A, M)$. En effet, $F_{STI} = \mathcal{D}^{V_F}$, $S_{STI} = \{s \in \mathcal{D}^{V_S} | \exists f \in F_{STI}, f \in \pi(s)\ \equiv (s, f) \in [\![A]\!]\}$. Pour illustration, la représentation graphique dans l'exemple 2 correspond exactement au STI sémantique de l'automate à contraintes. On notera le STI sémantique d'un automate à contraintes $[\![A]\!]$.

#### Les flux ou l'indéterminisme

Les valeurs de flux dans un STI, de même que les variables de flux d'un composant AltaRica, peuvent être modifiées à tout instant par une transition $\epsilon$ : ce comportement est exprimé par la condition $\forall s \in S, f \in \pi(s),\ (s, f, \varepsilon, s) \in T$.

De plus, lorsqu'une transition est tirée, i.e. $(s, f, e, s') \in T$, l'ensemble des configurations atteintes est l'ensemble des $(s', f')$ telles que la valeur de flux $f' \in \pi(s')$. Les STI sont indéterministes d'après ces deux conditions.

**Exemple 3** *On considère le STI* $\mathcal{S} = \langle E, F, S, \pi, T \rangle$ *avec* $E_+ = \{e\}$*,* $S = \{0, 1\}$*,* $F = \{0, \cdots, p\}$*,* $\pi(0) = \{f \in F | f \text{ est pair}\}$*,* $\pi(1) = \{f \in F | f \text{ est impair}\}$ *et* $T = \{(0, f, e, 1)\}$*. On représente l'indéterminisme du STI dans les Figures 2.5 et 2.6 : on énumère les transitions possibles depuis une configuration* $(0, 2i)$*. On note* $R(p, 2)$ *le reste de la division euclidienne de* $p$ *par* $2$*.*

#### La bisimulation

Pouvoir comparer deux modèles est essentiel : si l'on veut réutiliser un composant ou substituer un composant par un autre, il est nécessaire d'assurer que le comportement sera celui attendu. La *simulation* assure que tout comportement du modèle initial sera dans le nouveau composant. la *bisimulation* certifie que les deux composants ont exactement les même comportements. On définit la relation de bisimulation pour les STI.

**Définition 3 (Bisimulation interfacée [Poi00])** *Soient* $\mathcal{A}_1 = \langle E, F, S_1, \pi_1, T_1 \rangle$ *et* $\mathcal{A}_2 = \langle E, F, S_2, \pi_2, T_2 \rangle$ *deux STI. Une relation de bisimulation interfacée pour* $\mathcal{A}_1$ *et* $\mathcal{A}_2$ *est une relation* $R \subseteq S_1 \times S_2$ *donnée par :*

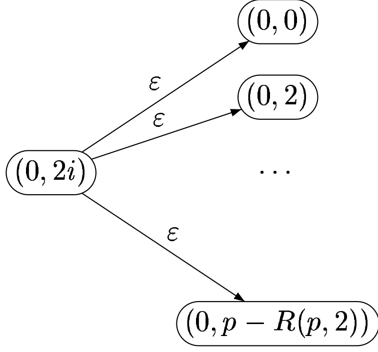



FIG. 2.5 – $(0, 2i, \varepsilon, 0) \in T$

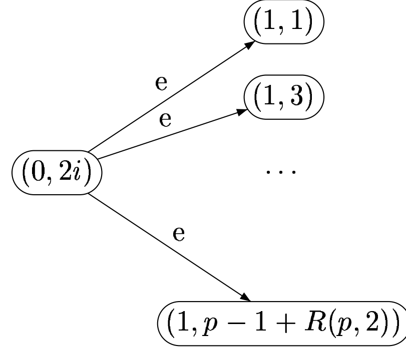



FIG. 2.6 – $(0, 2i, e, 1) \in T$

1. $\forall s_1 \in S_1, \exists s_2 \in S_2, (s_1, s_2) \in R$ *et* $\forall s_2 \in S_2, \exists s_1 \in S_1, (s_1, s_2) \in R$
2. $\forall (s_1, s_2) \in R,\ \pi_1(s_1) = \pi_2(s_2)$
3. $\forall (s_1, f, e, s'_1) \in T_1,\ \forall s_2 \in S_2$ *tel que* $(s_1, s_2) \in R$ *alors* $\exists (s_2, f, e, s'_2) \in T_2, (s'_1, s'_2) \in R$
4. $\forall (s_2, f, e, s'_2) \in T_2,\ \forall s_1 \in S_1$ *tel que* $(s_1, s_2) \in R$ *alors* $\exists (s_1, f, e, s'_1) \in T_1, (s'_1, s'_2) \in R.$

*Deux STI sont bisimilaires inerfacés si, et seulement si il existe une relation de bisimulation interfacée entre eux.*

Deux composants AltaRica $\mathcal{A}_i, i = 1, 2$ sont bisimilaires si, et seulement si leurs sémantiques le sont, i.e. si $[\![\mathcal{A}_i]\!], i = 1, 2$ sont bisimilaires interfacés.

### 2.1.7 Les priorités sur les événements

Spécifier des *priorités* entre événements est un concept classique de modélisation. Les priorités permettent de réguler l'activité du système : éviter les problèmes d'interblocage (interruption, préemption) ou améliorer les délais d'attente. On définit un ordre partiel sur les actions et si en un état, deux actions sont tirables, seule la plus prioritaire est possible ou les deux si les actions sont incomparables. On distingue deux types de priorités : les *statiques*, elles sont fixées au départ et restent inchangées, et les *dynamiques*, elles sont réévaluées régulièrement puisque fonctions de l'état du système.

On peut spécifier un ordre partiel sur les événements dans AltaRica et ces priorités sont statiques. Intuitivement, on considère un composant $\mathcal{A}$ et $<$ une relation de priorité n'ayant que deux éléments comparables $e < e'$. Pour toute configuration $(s, f)$ où des transitions étiquetées par $e$ et par $e'$ sont tirables, comme dans la Figure 2.7, alors toutes les transitions étiquetées par $e$ sont éliminées comme on le constate dans la Figure 2.8.

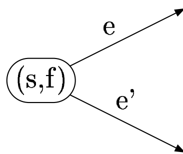



FIG. 2.7 – $\mathcal{A}$ sans priorité

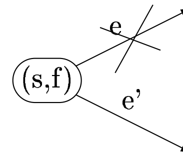



FIG. 2.8 – $\mathcal{A}$ avec priorité $e < e'$

**Définition 4 (Relation de priorité [Poi00])** *Une relation de priorité sur un ensemble d'événements $E$ est une relation partielle d'ordre strict sur $E_+$ $(= E \setminus \{\varepsilon\})$.*

On peut exprimer l'effet syntaxique d'une relation de priorité sur un composant. En effet, un composant avec priorité $\mathcal{A} = (V_S, V_F, E, A, M, <)$ est le composant $\mathcal{A} \restriction < = (V_S, V_F, E, A, M \restriction <)$ dont la relation de transitions est contrainte par la relation de priorité $<$. Une transition $(g, e, a) \in M \restriction <$ est tirable dans un état $(s, f)$ si, et seulement si $(g, e, a)$ est valide en $(s, f)$ (c'est-à-dire qu'elle vérifie la post condition $\pi(a(s, f)) \neq \emptyset$) et $e$ est maximal pour $<$ en $(s, f)$, i.e. $\forall (g', e', a')$ valide en $(s, f)$, on n'a pas $e < e'$. On peut alors exprimer syntaxiquement cette réduction, les priorités introduisent implicitement une quantification existentielle sur les gardes [Poi00] : $(G, e, a) \in M \restriction <$ si, et seulement si $\exists (g, e, a) \in M,\ G = g \wedge \bigwedge_{e<e'} \neg\big(\bigvee_{(g',e',a')} g' \wedge\ \forall s,\ \exists f,\ g'(s, f) = 1 \Rightarrow (g', e', a') \text{ valide en } (s, f)\big)$. Quand l'automate à contraintes est fini, les quantificateurs existentiels sont éliminés puisqu'il suffit d'énumérer les possibilités.

De même on définit l'effet d'une relation de priorité sur un STI :

**Définition 5 (Opérateur de restriction de priorité)** *Soit $\mathcal{A} = (S, F, E, A, T)$ un STI et $<$ une relation de priorité, on définit l'opérateur de restriction de priorité $\restriction$ pour la relation de transitions $T$ et la relation de priorité $<$ par : $(s, f, e, s') \in T \restriction < \iff \big((s, f, e, s') \in T \wedge \forall s'' \in S, \forall e' \in E, (s, f, e', s'') \in T \Rightarrow e \not< e'\big)$. On note $\mathcal{A} \restriction < = \langle E, F, S, \pi, T \restriction < \rangle$.*

La bisimulation interfacée est stable par opérateur de restriction de priorité. La relation de priorité agit de la même manière sur deux STI ayant le même comportement.

**Théorème 1 (Stabilité de la bisimulation interfacée par priorité [Poi00])** *Soient deux STI $\mathcal{A}_i = (S_i, F, E, A_i, T_i), i = 1, 2$ et $<$ une relation de priorité sur $E$. Si $\mathcal{A}_1$ et $\mathcal{A}_2$ sont bisimilaires interfacés, alors $\mathcal{A}_1 \restriction <$ et $\mathcal{A}_2 \restriction <$ aussi.*

L'encodage syntaxique d'une relation de priorité sur un composant coïncide avec l'effet de cette relation sur sa sémantique.

**Proposition 1 ([Poi00])** *Soit $\mathcal{A} = (V_S, V_F, E, A, M)$ un composant et $<$ une relation de priorité sur $E$, alors $[\![\mathcal{A}]\!] \restriction < = [\![\mathcal{A} \restriction <]\!]$.*

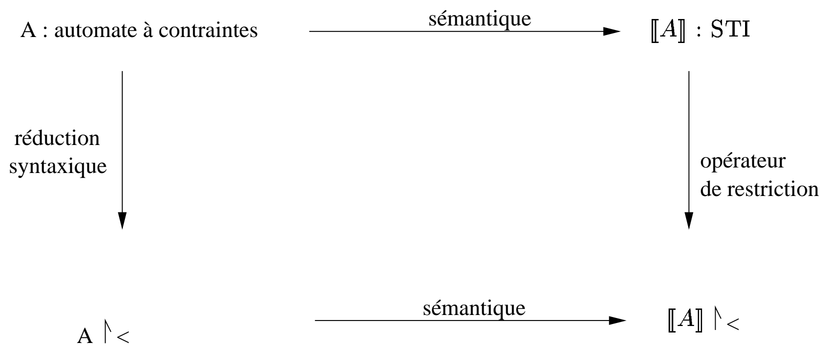



FIG. 2.9 – Diagramme commutatif sémantique/priorité

L'ensemble de ces propriétés entraînent l'existence du diagramme commutatif de la Figure 2.9. Ainsi, étant donné un automate à contraintes $\mathcal{A}$ et une relation de priorité $<$, calculer la sémantique de $\mathcal{A}$ puis réduire le STI $[\![\mathcal{A}]\!]$ en utilisant l'opérateur de restriction de priorité est équivalent à

résoudre syntaxiquement la relation de priorité sur $\mathcal{A}$ et donner la sémantique du nouvel automate à contraintes $\mathcal{A}\lceil<$. On peut donc indifféremment suivre l'une ou l'autre des procédures.

## 2.2 Le langage AltaRica

Nous avons introduit la syntaxe d'AltaRica (classique) dans la setcion 2.2.1 page 28. Nous précisons dans la section 2.11 des points sur cette syntaxe concrète. La grammaire se trouve dans [Vin03a] ou en Annexe C en omettant les ajouts temporels apparaissant en caractères gras. Nous introduisons également les *systèmes de transitions interfacés* : cette nouvelle variation des systèmes de transitions permet d'exprimer naturellement la sémantique des automates à contraintes et par conséquent du langage AltaRica. Dans la section 2.3, nous présentons les aspects hiérarchiques du langage et les opérateurs de modélisation proposés par AltaRica.

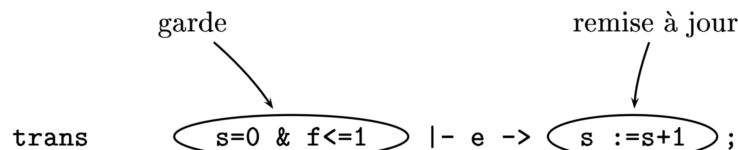
### 2.2.1 La syntaxe AltaRica

Le langage AltaRica, à l'instar des automates à variables, est un langage de variables. Un modèle AltaRica est décrit de la sorte :

**Déclarations :** les variables partagées ou de *flux* ; les variables internes ou d'*état* ; les événements sont déclarés après les mots clés respectivement *flow*, *state* et *event*. Les variables sont typées. On peut éventuellement exprimer des relations de priorités entre eux. Informellement, une priorité contraint davantage les comportements du composant : si un événement $e_1$ est moins prioritaire qu'un événement $e_2$, noté $e_1 < e_2$, alors aucune transition étiquetée $e_1$ ne peut être tirée en une configuration où une transition $e_2$ est tirable. La notion de priorité sera étudiée formellement dans le paragraphe 2.1.7 page 26. L'ordre des déclarations n'a pas d'importance. Par exemple, on déclare deux variables dans $[0, 2]$, l'une partagée et l'autre interne, ainsi qu'un événement $e$ de la sorte :

```
flow f: [0,2];
state s : [0,2];
event e
```

**Transitions :** il s'agit de *macro-transitions* puisque certaines variables sont testées et d'autres remises à jour lors du franchissement d'une transition. Par exemple, on spécifie une transition étiquetée par $e$ qui incrément la valeur de $s$ par 1 si $s = 0$ et $f \leq 1$ de la sorte :




**Contraintes :** les variables manipulées peuvent être contraintes par des formules, soit initiales soit globales. Par exemple, si on impose qu'initialement $s$ soit nul et que $s = 2$ si, et seulement si $f = 2$, on écrit dans le modèle :

```
init s:=0;
assert s=2 <=> f=2
```

On donne la syntaxe générale d'un modèle élémentaire AltaRica, appelé composant, dans la Figure 2.10. Un composant est spécifié entre les mots clés *node* et *edon*.

**Exemple 4** *avec la syntaxe AltaRica introduite dans la section 2.2.1.*

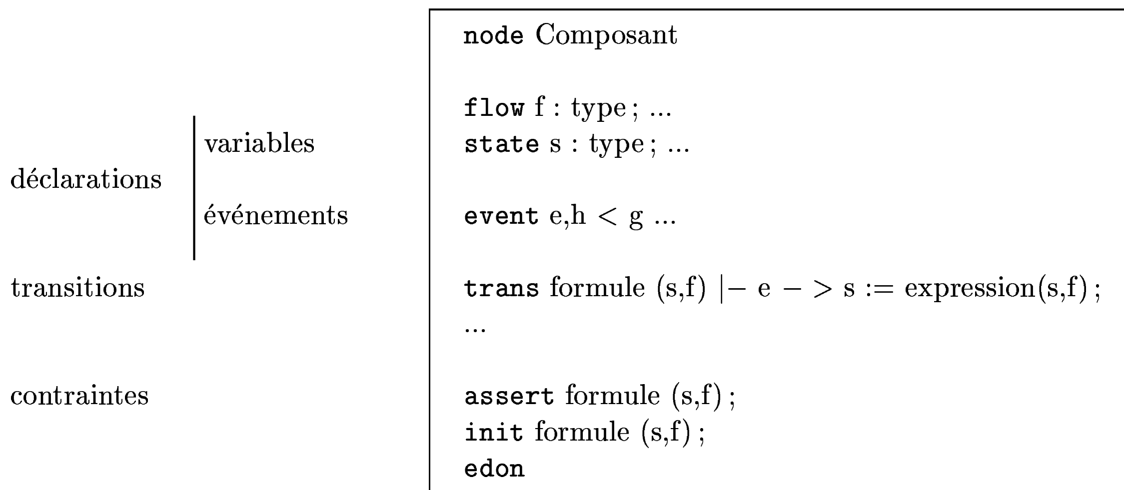



FIG. 2.10 – Syntaxe d'un composant AltaRica

```
node Paquet
 flow  N :[0,7]
 state  n :[0,7]
 event  joue
 trans n>0 |- joue -> n:=n-1;
      n>1 |- joue -> n:=n-2;
      n>2 |- joue -> n:=n-3;
      n>3 |- joue -> n:=n-4;
      n>4 |- joue -> n:=n-5;
      n>5 |- joue -> n:=n-6;
      n>6 |- joue -> n:=n-7;
 init n=7
 assert N=n
edon
```

Cette syntaxe a été proposée à partir de [GLP+98] pour représenter les automates à contraintes [BR94]. Nous utiliserons cette syntaxe dans la suite. Nous pouvons à présent définir le modèle inhérent à toute spécification AltaRica, les automates à contraintes.

### 2.2.2 L'interface des composants

L'interface d'un composant correspond aux éléments visibles du composant par l'extérieur. Cette interface est constituée des variables de flux et des événements du composant. Le *nœud parent* ou encore *nœud hiérarchique supérieur* (on reviendra ultérieurement sur cette notion, voir partie 2.3 page 30) est le seul y avoir accès. Le nœud parent peut alors contraindre les variables de flux et synchroniser les événements. A l'inverse, les variables d'état sont internes au composant et aucun autre composant ne peut y accéder.

Dans la nouvelle version d'AltaRica [Vin03a], un attribut de visibilité optionnel peut être ajouté aux variables et aux événements. Les trois attributs sont : `public`, `private` ou `parent`. La syntaxe devient alors

```
champ f [: type] : attribut ...
```

avec champ = flow, state ou event et type n'est utilisé que pour les variables. Tout ce qui est `private` est interne au composant, ainsi une variable de flux `private` est locale et non lisible ni

modifiable depuis l'extérieur et un événement `private` ne peut être synchronisé, c'est une transition $\varepsilon$ renommée. A l'inverse, les éléments `public` sont accessibles depuis toutes les couches de la hiérarchie. Une variable `public` peut être lue par tout composant du système et contrainte par les différentes assertions. Elle ne sera seulement modifiée que par le composant qui la déclare en tant que variable d'état `public`. Un événement `public` peut être spécifié dans un vecteur de synchronisation à n'importe quel niveau supérieur de la hiérarchie. Une fois utilisé, il devient `private` pour les niveaux encore supérieurs. Enfin, `parent` signifie que l'élément n'est accessible que par le nœud parent. Par défaut, l'attribut d'une variable de flux ou d'un événement est `parent` et celui d'une variable d'état est `private`.

Nous avons présenté les composants élémentaires du langage AltaRica ainsi que le passage du modèle des automates à contraintes au langage. Une deuxième caractéristique fondamentale est la possiblité de modéliser des systèmes hiérarchiquement.

## 2.3 La hiérarchie

AltaRica est un langage hiérarchique : ainsi un modèle AltaRica, dont le nom générique est *nœud* possède des sous-nœuds qui interagissent au moyen de leurs interfaces, c'est-à-dire par des synchronisations d'événements et par des contraintes sur des variables visibles. On peut représenter intuitivement un nœud comme dans la Figure 2.11. Le *Nœud* a un comportement local (correspondant à l'automate), des sous-nœuds, *N1* et *N2*, des contraintes touchant les variables visibles (dans l'assertion) et des vecteurs de synchronisation portant sur les transitions. Ainsi le vecteur $< e, N1.e?, N2.e >$ exprime que l'événement local $e$ est synchronisé avec l'événement $e$ du nœud *N2* et possiblement avec l'événement $e$ du nœud *N1*. On dit que *Nœud* est le parent des nœuds *N1* et *N2*.

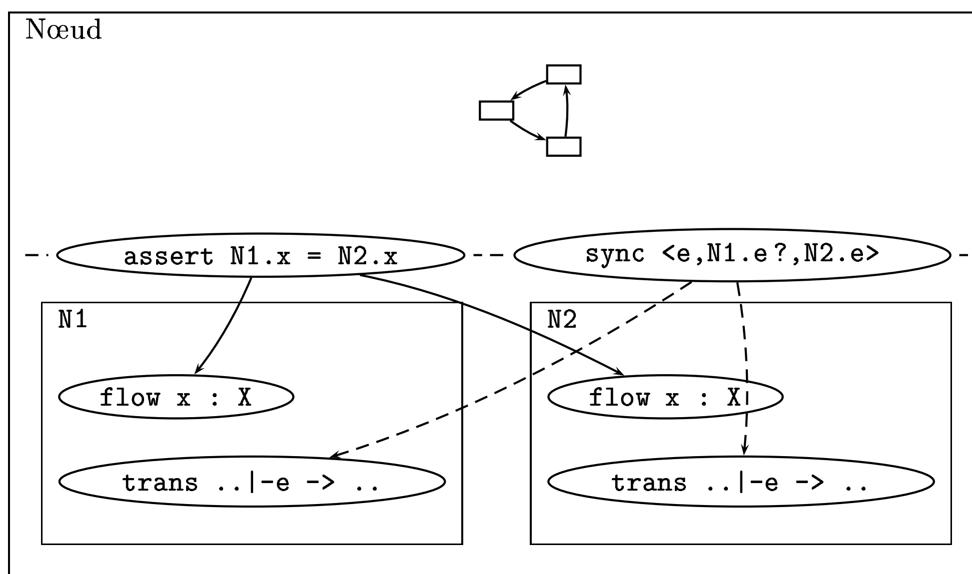



Fig. 2.11 – Interface des composants

| | | |
|---|---|---|
| | | `node` Nœud |
| déclarations | variables | `flow` f : type ; ... <br> `state` s : type ; ... |
| | événements | `event` e,h < g ... |
| | sous-nœuds | `sub` C : Composant ; |
| transitions | locales | `trans` formule(s,f,**C.f**) |– e – > s := expression(s,f,**C.f**) ; <br> ... |
| | synchro | `sync` <e,C.h ...>, <br> <h,C.e ? ...> |
| contraintes | | `assert` formule (s,f,**C.f**) ; <br> `init` formule (s,f,**C.f**,,**C.s**) ; <br> `edon` |

FIG. 2.12 – Syntaxe d'un nœud AltaRica

### 2.3.1 Les nœuds AltaRica

Un nœud est un composant étendu avec des opérateurs de modélisation hiérarchiques. La syntaxe générale d'un nœud est donnée dans la Figure 2.12 et les différences d'avec un composant ont été soulignées par l'emploi de caractères gras. Ainsi :

**sous-nœuds :** dans les déclarations, on spécifie les sous-nœuds et on donne leur “type” en tant que sous-nœud ;

**synchronisations :** les synchronisations d'événements sont spécifiées après le champ *sync*. Elles sont de deux types : la première est une extension du produit synchronisé *à la Arnold-Nivat* pour les automates à contraintes et la deuxième, les *broadcast*, symbolisée par l'utilisation de ?, est un mélange de produit synchronisé *à la Arnold-Nivat* et d'ordre partiel sur les vecteurs d'événements. Cette notion est plus amplement développée ci après (page 31) ;

**coordination de flux :** on peut appliquer des contraintes globales sur des variables visibles de sous-nœuds ce qui entraîne que l'assertion peut toucher les variables de flux des sous-nœuds. De même, les gardes des transitions peuvent prendre en compte les valeurs de flux des sous composants.

#### Synchronisation particulière : le broadcast

Cet artifice syntaxique a déjà été évoqué dans la partie 2.4, page 35, on ne spécifie plus un vecteur de synchronisation mais un ensemble de vecteurs. Ce mécanisme symbolise le comportement *parmi les actions spécifiées, on synchronise le nombre maximal de celles tirables dans l'état du système.* On note le nom d'un événement suivi d'un ?, un événement qui n'entrera pas forcément dans la synchronisation. Etant donnés un nœud $\mathcal{N}$ et ses $n$ sous-nœuds $\mathcal{N}_i$, les événements de $\mathcal{N}$ sont notés $E$ et ceux de $\mathcal{N}_i$, $E_i$, on considère un ensemble de vecteurs de broadcast $I \subseteq E^? \times E_1^? \times \cdots \times E_n^?$.

Un vecteur de $I$ est de la forme $e \in I \Rightarrow e = \langle e_0^{\sim}, e_1^{\sim}, \cdots, e_n^{\sim} \rangle$ avec $\sim \in \{?, \emptyset\}$. Soit un vecteur de broadcast $e$, $e$ génère un ensemble de vecteurs de synchronisation $Vec(e) \subseteq E \times E_0 \times \cdots \times E_n$ , appelés *vecteurs d'instanciation.* Soit $u = \langle u_1, \cdots, u_n \rangle \in E \times E_0 \times \cdots \times E_n$, $u$ est un vecteur engendré par $e$ si, et seulement si $\forall i,\ e_i^{\sim} = e_i \Rightarrow u_i = e_i$ et $e_i^{\sim} = e_i? \Rightarrow u_i = e_i$ ou $e_i = \varepsilon$. On ordonne $Vec(e)$ par la relation d'ordre partiel : $u, v \in Vect(e), u \sqsubset v \Longleftrightarrow \{u_i | u_i \neq \varepsilon\} \subset \{v_i | v_i \neq \varepsilon\}$. Sémantiquement, une transition étiquetée $e = \langle e_0, e_1, \cdots, e_n \rangle$ est tirable en une configuration $(s, f)$ si, et seulement si

- soit $e$ est un vecteur de synchronisation spécifié tirable $(s, f)$ et alors $Vec(e) = \{e\}$,
- soit $e$ est un vecteur d'instanciation, i.e. $\exists e'$, tel que $e \in Vec(e')$, tirable en $(s, f)$ et $e$ est maximal en cette configuration.

On peut également ajouter des contraintes sur les vecteurs d'instanciation :

```
<e0∼ 0,e1∼ 1,···,en∼ 2> ∼ p
```

avec $\sim_i \in \{\emptyset, ?\}$, $\sim \in \{=, <, >, \leq, \geq\}$ et $p \in \mathbb{N}$. Alors, les vecteurs d'instanciation $u = \langle e'_0, \cdots, e'_n \rangle$ admissibles sont tels que $card\{i|\ \sim_i = ? \wedge e'_i = e_i\} \sim p$. L'ordre est alors maintenu pour les vecteurs admissibles et dans ce cas il peut y avoir plusieurs éléments maximaux.

L'expressivité des nœuds n'est pas modifiée par les broadcasts, en effet ils n'agissent que sur la validité des actions pour des ensembles de valuations. On peut réécrire le même système en énumérant toutes les possibilités et en spécifiant les propriétés induites par $\sqsubset$.

#### Sémantique des nœuds

Formellement, un nœud est un automate à contraintes modifié muni de sous-composants :

**Définition 6 (Nœud- syntaxe abstraite [Poi00])** *Un* nœud *est un uplet* $\mathcal{N} = \langle V_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ *avec :*

1. $V_F$ *est un ensemble de variables de flux ;*
2. $E = E_+ \cup \{\varepsilon\}$ *est un ensemble fini d'événements où* $\varepsilon \notin E_+$ *;*
3. $<$ *est une relation de priorité sur* $E$ *;*
4. *pour tout* $i \in [1, n]$, $\mathcal{N}_i$ *est un composant ou un nœud,* $F_i$ *est l'ensemble des variables flux de* $\mathcal{N}_i$ *et* $E_i$ *est l'ensemble des événements. On suppose* $\forall i \neq j \in [1, n], V_{F_i} \cap V_{F_j} = \emptyset$ *;*
5. $\mathcal{N}_0$ *est un composant spécial appelé le* contrôleur, *il s'agit du comportement local du nœud. L'ensemble des événements de ce nœud est* $E_0 = E$ *ordonné par la relation vide. L'ensemble des variables de flux de* $\mathcal{N}_0$ *est* $F_0 = F \cup F_1 \cup \cdots \cup F_n$,
6. $\tilde{V} \subseteq E_0 \times E_1 \times \cdots \times E_n$ *est une expansion de l'ensemble des vecteurs de synchronisation avec la relation de priorité* $<_{\tilde{V}}$ *(cette relation exprime l'ordre induit par les broadcast).*

**Remarque 1** *Dans le nœud, un ensemble* $I$ *de vecteurs est spécifié alors on note* $\tilde{V} = \bigcup_{a \in I} Vec(a)$. *On peut alors exprimer la relation* $<_{\tilde{V}}$ *engendrée par les vecteurs de broadcast. Soient* $e, e' \in \tilde{V}$, $e$ *et* $e'$ *sont comparables pour* $<_{\tilde{V}}$ *si, et seulement si* $\exists a \in I$, *tel que* $e, e' \in Vec(a)$. *Dans ce cas,* $e <_{\tilde{V}} e'$ *si, et seulement si* $e \sqsubset e'$.

**Définition 7 (Nœud- sémantique [Poi00])** *Soit un nœud* $\mathcal{N} = \langle V_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$. *On suppose que pour tout* $i \in [0, n]$, $[\![\mathcal{N}_i]\!] = \langle E_i, F_i, S_i, \pi_i, T_i \rangle$. *Alors la sémantique de* $\mathcal{N}$ *est le STI* $[\![\mathcal{N}]\!] = \langle E, F, S, \pi, T \rangle$ *défini par :*

1. $F = \mathcal{D}^{V_F}$ *;*

2. *pour tout* $s_i \in S_i$, $i \in [0,n]$, $\pi(s_0, s_1, \cdots, s_n) = \{f \in F | \exists (f, f_1, \cdots, f_n) \in \pi_0(s_0), \forall i \in [1,n], f_i \in \pi_i(s_i)\}$,
3. $S = \{s \in S_0 \times \cdots \times S_n | \pi(s) \neq \emptyset\}$,
4. $T \subseteq S \times F \times E \times S$ *est défini par :*
   (a) *on considère* $<_0$ *la relation de priorité engendrée par* $<$ *et définie par :* $(e_0, e_1, \cdots, e_n) <_0 (e'_0, e'_1, \cdots, e'_n) \iff e_0 < e'_0$,
   (b) *soit* $T_{\mathcal{N}} \subseteq S \times F \times \tilde{V} \times S$ *l'ensemble des transitions définies par : si* $e = (e_0, e_1, \cdots, e_n) \in \tilde{V}$,

$$\langle (s_0, \cdots, s_n), f, e, (s'_0, \cdots, s'_n) \rangle \in T_{\mathcal{N}} \iff \begin{cases} \exists f_0 = (f, f_1, \cdots, f_n) \in \pi_0(s_0), \\ \forall i \in [0,n], (s_i, f_i, e_i, s'_i) \in T_i \wedge f_i \in \pi_i(s_i) \end{cases}$$

   (c) *alors* $T = \left(T_{\mathcal{N}} \upharpoonright <_{\tilde{V}}\right) \upharpoonright <_0$.

**Exemple 5 (Jeu de Marienbad - III)** *On donne la spécification complète du jeu [Gri02]. Un composant Arbitre, Figure 2.13, alterne les joueurs A et B. L'action joueurA (resp. joueurB) correspond à A (resp. B) joue son tour. La condition initiale assure que A est le premier joueur. Le nœud Marienbad, Figure 2.14, modélise le jeu : les* 5 *sous composants correspondent à l'arbitre et aux* 4 *paquets d'allumettes. Les vecteurs de broadcast assurent qu'un joueur ne prendra des allumettes que dans un paquet : en effet on rajoute une contrainte*

```
<R.joueurA, L1.joue?, L2.joue?, L3.joue?, L4.joue?> = 1
```

*ce qui force les vecteurs d'instanciation admissibles de ne réaliser qu'un* $L_i$*.joue, c'est-à-dire une action sur un seul paquet.*

```
node Arbitre
  state AvaJouer : bool
  event joueurA, joueurB
  trans
      AvaJouer  |- joueurA ->
             AvaJouer := false;
      ~AvaJouer |- joueurB ->
             AvaJouer := true;
  init AvaJouer := true;
edon
```

FIG. 2.13 – Modélisation de l'arbitre

```
node Marienbad
  sub R : Arbitre;
       L1, L2, L3, L4 : Paquet;
  sync
    <R.joueurA, L1.joue?, L2.joue?,
        L3.joue?,L4.joue?> = 1;
    <R.joueurB, L1.joue?, L2.joue?,
        L3.joue?,L4.joue?> = 1;
  init L1.n := 1, L2.n := 3,
       L3.n := 5, L4.n := 7;
edon
```

FIG. 2.14 – Modélisation du jeu de Marienbad

*Le perdant est alors le joueur* $C \in \{A, B\}$ *qui jouera la dernière transition.*

### 2.3.2 Réécriture d'un nœud AltaRica en composant AltaRica

Le théorème central concernant la hiérarchie est l'algorithme de *mise à plat* d'un nœud AltaRica en un composant AltaRica. La mise à plat consiste à transformer un modèle AltaRica hiérarchique (comportant des sous-nœuds) en un modèle élémentaire (un composant sans sous composant).

**Théorème 2 (Réécriture d'un nœud en composant [Poi00])** *Soit un nœud* $\mathcal{N} = \langle V_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$, *on peut réécrire ce nœud sous forme d'un composant* $\mathcal{C}_{\mathcal{N}}$ *selon le procédé décrit dans la Définition 35 page 137 donnée en Annexe. Alors les sémantiques sont égales, i.e.* $[\![\mathcal{N}]\!] = [\![\mathcal{C}_{\mathcal{N}}]\!]$.

Cet algorithme est implanté dans Mec V [Vin03a], en effet l'outil met à plat tous les modèles avant de vérifier des propriétés. Nous développons cette implantation dans la partie 6.2.4.

**Exemple 6 (Jeu de Marienbad - fin)** *On utilise l'AltaTool [Poi03]* a-chek *pour mettre à plat le nœud Marienbad et on donne la représentation obtenue par l'outil dans la Figure 2.15. Le système comporte* 752 *états ce qui explique le manque de lisibilité.*

*Le produit libre des automates avec les mêmes conditions initiales génère* 768 *états. On illustre une utilisation possible de* Mec V *en recherchant les* 16 *états non atteignables. On peut expliquer pourquoi ces états ne sont pas atteignables :*

1. $(R.AvaJouer, L1.n = 1, L2.n = 3, L3.n = 5, L4.n = 7)$ *: A joue en premier et doit prendre au moins une allumette, donc B ne pourra pas jouer lorsque tous les paquets sont complets,*
2. $(R.AvaJouer, L1.n = 0, L2.n = 3, L3.n = 5, L4.n = 7)$, $(R.AvaJouer, L1.n = 1, L2.n = 2, L3.n = 5, L4.n = 7)$, $(R.AvaJouer, L1.n = 1, L2.n = 3, L3.n = 4, L4.n = 7)$ *et* $(R.Ava-Jouer, L1.n = 1, L2.n = 3, L3.n = 5, L4.n = 6)$ *correspondent aux configurations où A et B ont joué successivement et c'est encore au tour de A. Au moins* 2 *allumettes ont disparu, donc A ne peut jouer lorsqu'une seule une allumette manque.*
3. $(R.AvaJouer, L1.n = 0, L2.n = 2, L3.n = 5, L4.n = 7)$, $(R.AvaJouer, L1.n = 0, L2.n = 3, L3.n = 4, L4.n = 7)$, $(R.AvaJouer, L1.n = 1, L2.n = 2, L3.n = 4, L4.n = 7)$, $(R.Ava-Jouer, L1.n = 0, L2.n = 3, L3.n = 5, L4.n = 6)$ $(R.AvaJouer, L1.n = 1, L2.n = 2, L3.n = 5, L4.n = 6)$, $(R.AvaJouer, L1.n = 1, L2.n = 3, L3.n = 4, L4.n = 6)$ *: dans ces configurations, il manque une allumette dans deux paquets différents. On est alors assuré que* 2 *tours exactement ont eu lieu et c'est obligatoirement à A de jouer.*
4. $(R.AvaJouer, L1.n = 0, L2.n = 2, L3.n = 4, L4.n = 7)$, $(R.AvaJouer, L1.n = 0, L2.n = 2, L3.n = 5, L4.n = 6)$, $(R.AvaJouer, L1.n = 0, L2.n = 3, L3.n = 4, L4.n = 6)$, $(R.Ava-Jouer, L1.n = 1, L2.n = 2, L3.n = 4, L4.n = 6)$ *: c'est un peu le même contexte,* 3 *allumettes ont été retirées du jeu et dans* 3 *paquets différents, c'est nécessairement à B de jouer.*
5. $(R.AvaJouer, L1.n = 0, L2.n = 2, L3.n = 4, L4.n = 6)$ *: cette dernière configuration non atteignable correspond à la disparition de* 4 *allumettes dans* 4 *tas différents et c'est à A de jouer.*

*On peut en déduire le nombre d'états non accessibles pour un jeu de Nim général à n paquets, il faut retirer au nombre d'états du produit libre les états suivants :*

| *Tour* | *Configurations* | *Nombre* |
|---|---|---|
| *B* | *tous les paquets sont pleins* | $C_n^0$ |
| *A* | *on enlève une allumette dans* $paquet_i$ | $C_n^1$ |
| *B* | *on enlève une allumette dans* $paquet_i$ *et* $paquet_j$ | $C_n^2$ |
| | ... | |
| | *on enlève une allumette dans tous les paquets* | $C_n^n$ |

*donc le nombre de ces états inaccessibles est* $\Sigma_i C_n^i = 2^n$*. Réciproquement, si une configuration* $(R.AvaJouer, L1.n, L2.n, L3.n, L4.n)$ *possède un paquet* $L_i$ *contenant* 2 *allumettes de moins qu'initialement ou plus, cela peut aussi bien être à A que B de jouer. On suppose par exemple* $L_4.n = p$ *et par symétrie on obtient le résultat pour tous les paquets. A et B jouent jusqu'à la configuration* $(R.AvaJouer, L1.n, L2.n, L3.n, 7)$*. Ensuite, si A vient de jouer pour atteindre cette nouvelle configuration, il retire exactement p allumettes dans L4 et c'est à B de jouer. Ou alors, A*

FIG. 2.15 – Modèle du jeu de Marienbad mis à plat

*retire exactement $p - 1$ allumettes dans L4, puis B 1 dans L4 et c'est au tour de A de jouer dans la configuration.*

*On peut également se poser la question de trouver une stratégie gagnante pour l'un des joueurs et cette question est étudiée dans [Gri02].*

## 2.4 Présentation générale de l'atelier logiciel AltaRica

Cette partie est destinée à présenter les langages AltaRica et les analyses de systèmes qui peuvent être alors réalisées à partir de spécifications AltaRica. Après présenté informellement la syntaxe des modèles AltaRica, nous exprimons la définition formelle des automates à contraintes [BR94]. Ces systèmes de transitions sont le fondement des différents langages AltaRica. Ces derniers seront ensuite présentés ainsi que la boîte à outils AltaRica.

Des opérateurs permettant d'exprimer des hiérarchies d'automates à contraintes ont été ajoutés au modèle [Poi00], ces opérateurs seront développés ultérieurement dans la partie 2.3. Cela permet de construire un langage hiérarchique de modélisation très expressif appelé AltaRica. Il existe trois variantes de ce langage représentées dans la Figure 2.16 et détaillées dans la section 2.4.1. Ces langages présentent néanmoins des caractéristiques communes induites par le modèle des automates à contraintes (ces attributs seront présentés en détails dans la section 2.2 page 28) :

**variables :** on manipule des variables locales, les variables d'état, et des variables partagées, les variables de flux ;

**contraintes :** il est possible d'appliquer des contraintes globales sur les variables d'un modèle. Ces contraintes sont regroupées dans l'assertion et doivent toujours être vérifiées. Ainsi on peut représenter des relations entre différents modèles au travers de variables de flux. Il est possible de spécifier une formule initiale ;

**macro-transitions :** en tant que langage à variables, les transitions sont gardées et modifient les valeurs des variables d'état ;

**synchronisations :** le produit synchronisé *à la Arnold-Nivat* [Arn92, Arn94] est étendu pour les modèles dérivant des automates à contraintes. Un opérateur de synchronisation particulier, appelé *broadcast* (*diffusion*), existe : il permet de définir un ensemble de vecteurs de synchronisation ordonné partiellement par une relation d'ordre ;

**hiérarchie :** le modèle permet de spécifier des sous modèles (élémentaires ou hiérarchiques) dans un modèle. Les deux opérations hiérarchiques sont la synchronisation des événements du modèle et de ses sous-modèles, et l'assertion sur toutes les variables visibles dans le modèle ;

**priorité :** on peut exprimer des priorités entre événements.

La Figure 2.16 schématise les inclusions syntaxiques des différents langages AltaRica basés sur le modèle des automates à contraintes et les outils associés à chacun. Nous présentons ces langages dans la prochaine section.

### 2.4.1 Les langages AltaRica

Les attributs communs à tous les modèles dérivés des automates à contraintes fournissent des opérateurs de modélisation qui facilitent la spécification des systèmes. Les langages AltaRica sont

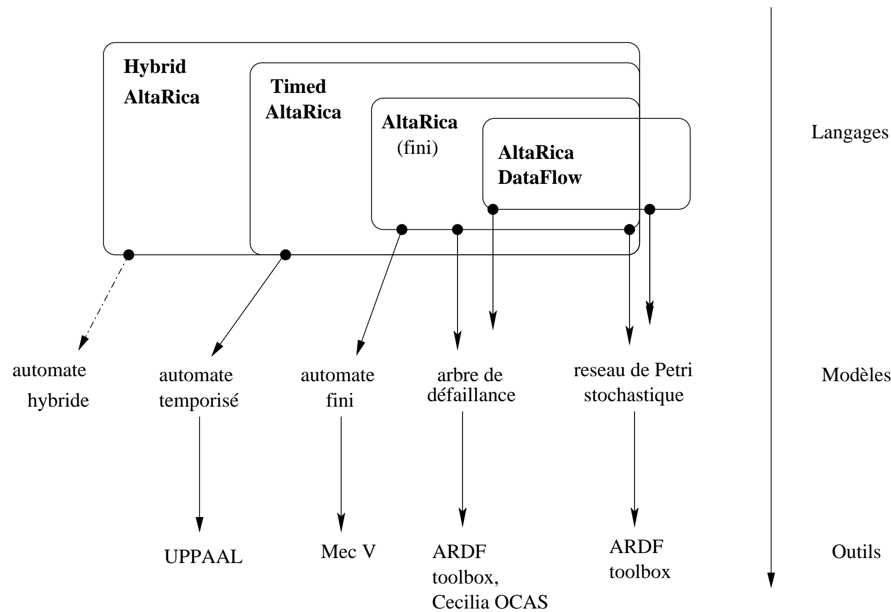



FIG. 2.16 – Plan de l'atelier AltaRica

des langages de spécification de haut niveau et un des principes de l'atelier AltaRica est de transformer ces modèles vers des formalismes de plus bas niveau et de traduire les modèles obtenus dans des formats reconnus par des outils de vérification.

Nous dirons qu'un modèle AltaRica est *compilé* en un autre modèle si le modèle AltaRica peut être automatiquement transformé en le deuxième modèle. Par exemple, si un modèle AltaRica peut être transformé en une machine à états finie, on donne la sémantique du modèle et on la représente sous forme d'une machine à états finie. Ainsi, le modèle *Paquet* est compilé en l'automate donné à la Figure 2.4.

**AltaRica - classique** Le langage AltaRica [GLP+98, AGPR00, Poi00] est exactement basé sur le modèle des automates à contraintes. Un modèle AltaRica fini, c'est-à-dire que les ensembles des variables sont finis, les domaines des variables sont finis et l'ensemble des événements est fini, peut (toujours) être compilé en une machine à états finie, en un arbre de défaillance[1] et un réseau de Petri stochastique. L'expressivité d'AltaRica fini est celle des automates finis.

**AltaRica DataFlow** Ce langage propose certaines réstrictions sur le modèle des automates à contraintes : les variables sont en nombre fini ; les domaines des variables d'états sont des intervalles fini d'entiers ; les variables de flux sont dans $\mathbb{R}$ et unidirectionnelles ; il n'y a ni priorité ni broadcast ; les assertions sont simplififées et contraignent les flux. Cette variante est équivalente aux *automates de mode* [MR98, Rau02]. L'expressivité de ce modèle est la même que celle d'AltaRica. Un automate de mode fini se compile en un ensemble de formules booléennes (et par conséquent en arbre de défaillance) et peut être interprété en un réseau de Petri stochastique.

[1] La méthode de l'arbre des défaillances [ARA94, Eri99] (fault tree) est encore actuellement l'une des plus fréquemment utilisées dans les études de SdF. Sa paternité reviendrait à H. Watson qui l'aurait créée en 1961 et développée au sein de la société Bell pour évaluer et améliorer la fiabilité du système de lancement du missile Minuteman. La construction de l'arbre se fait récursivement sur l'occurrence des événements redoutés : un premier événement redouté est choisi pour devenir la racine de l'arbre. On cherche la ou les conséquences immédiates de l'occurrence de cet événement (en terme d'événements redoutés). Puis, pour chacun des fils, on réitère le processus jusqu'à ce qu'on aboutisse à des événements n'entraînant aucun événement indésirable.

**Timed AltaRica et AltaRica hybride** Les modèles sont les automates à contraintes temporisés[2] [CPR02] respectivement les automates à contraintes hybrides[3]. Dans ces deux variantes, on considère des variables particulières à valeurs dans $\mathbb{R}$ qui suivent des *lois d'évolution*, c'est-à-dire dire qu'elles varient au cours du temps. Les lois considérées dans AltaRica hybride sont plus générales que celles de Timed AltaRica. Un automate à contraintes temporisé (resp. hybride) fini, c'est-à-dire que les ensembles des variables, des événements, des domaines des variables discrètes sont finis, se compile en un automate temporisé (resp. hybride).

### 2.4.2 Des outils pour des modèles AltaRica

Le modèle des automates à contraintes est très général et ne fait aucune hypothèse particulière de finitude. Pour pouvoir appliquer des algorithmes décidables, différentes restrictions ont été réalisées pour implanter les langages précédents. On suppose que les ensembles de variables, d'événements et les domaines sont finis. Des outils d'analyse sont intégrés dans l'atelier pour étudier les systèmes prenant en entrée des modèles AltaRica ou AltaRica DataFlow.

Mec V [ABC94, GV03, Vin03b] est un model-checker dont le langage de spécification a l'expressivité des formules du $\mu$-calcul étendu avec les quantificateurs du premier ordre et l'égalité. On peut ainsi vérifier des propriétés de logique temporelle sur un automate fini. Cet outil sera présenté plus avant dans la partie 6.2. Mec V =AltaRica + Toupie

*AltaTools* [Poi03] regroupe les outils propres au langage et ont pour objectif d'aider au debuggage des composants écrits en AltaRica. Les éléments principaux sont : un vérifieur syntaxique, un outil qui permet certaines réécritures syntaxiques des modèles AltaRica (en éliminant les broadcast par exemple), un solveur de contraintes, un mini model-checker qui calcule des points fixes, des traducteurs entre AltaRica et Lustre.

*ARBoost Technologies*[4] (AltaRica DataFlow ToolBox) regroupe plusieurs outils pour AltaRica DataFlow. Ainsi, *Aralia* [ARA96] est un outil pour la création, la présentation et le traitement d'arbres de défaillances (décomposition d'un arbre, recherches efficaces permettant d'atteindre très rapidement n'importe quel endroit de l'arbre, etc) et d'événements. On peut également citer le simulateur stochastique *alta-sto* qui évalue les valeurs des lois de distribution en fonction du temps.

*Moca-RP* [Sig95] est un logiciel destiné aux traitements probabilistes des systèmes dynamiques complexes pour lesquels il réalise des simulations de Monte Carlo à partir d'un modèle de comportement décrit à l'aide de Réseaux de Petri stochastiques interprétés. Une loi de probabilité est affectée à chaque transition. Dès qu'une transition devient valide, l'instant de son tir effectif est évalué en utilisant des nombres au hasard (simulation de Monte Carlo) répartis selon la loi de probabilité retenue.

### 2.4.3 Des études de cas avec AltaRica

La boîte à outils AltaRica (ses langages et ses outils) a été appliquée pour des études de cas. Dans [CS01], les auteurs recherchent une méthodologie d'utilisation d'AltaRica permettant de modéliser des architectures logicielles des systèmes de type *avionique modulaire intégré* et d'évaluer leurs capacités à tolérer et confiner les fautes. Leur choix s'est porté sur AltaRica car le langage spécifie formellement des systèmes en présence de fautes et de manière modulaire. L'analyse de défaillances se propageant dans un système complexe est relativement difficile mais les

[2]Définition 18 page 61
[3]Définition 30 page 98

caractéristiques de modélisation du langage (la propagation des flots) répond exactement aux besoins. Enfin, les outils qui accompagnent la modélisation permettent une étude assez approfondie. L'étude de cas porte sur un sous système embarqué qui contrôle le déplacement de l'avion en fonction des ordres du pilote et des paramètres de vol de l'avion. Convaincus de l'adéquation d'AltaRica avec de tels systèmes, les mêmes auteurs ont modélisé le système hydraulique d'un airbus A320 avec AltaRica [BCK+02].

Dans [Gri03], l'auteur modélise et valide (sous quelques réserves) un protocole de téléphonie mobile. Une entreprise voulait breveter un système de commande à distance *désactivant et activant des signaux émis par un téléphone portable* ainsi que le procédé de mise en œuvre. L'auteur a donc modélisé le protocole proposé et utilisé Mec V pour assurer les propriétés souhaitées par l'entreprise. L'auteur a trouvé des anomalies dans le protocole et montré qu'il fallait relâcher certaines contraintes.

Dans ce premier chapitre, nous avons introduit le formalisme AltaRica et le modèle des automates à contraintes. Nous allons définir des modèles à contraintes temporisés et hybrides à partir des automates à contraintes. A partir de ces nouveaux modèles, nous suivons la démarche de construction d'AltaRica et nous obtenons les langages Timed AltaRica et Hybrid AltaRica. Dans le prochain chapitre, nous poursuivons les rappels nécessaires à la formalisation de ces langages AltaRica temps réel. Pour cela, nous introduisons les automates temporisés [AD94] et hybrides [Hen96].

# Chapitre 3

# Rappels sur les automates temporisés et sur les automates hybrides

Les systèmes physiques opèrent dans un temps continu puisqu'ils peuvent changer d'états à n'importe quel instant. Les systèmes temps réel prennent en compte la dimension supplémentaire du temps et sont soumis à des contraintes temporelles strictes dont la violation peut entraîner des conséquences graves. Par exemple, un tel système peut garantir que l'exécution de tâches respecte certaines échéances. Sous cette hypothèse, il est nécessaire de manipuler *quantitativement* le temps, c'est à dire exprimer des conditions comme *l'événement e surviendra moins de 3 secondes après l'événement f*.

Pour modéliser précisément la dynamique des systèmes, on introduit un nouveau type de variables : les *variables continues*. Elles permettent de quantifier des grandeurs comme des mesures intervenant dans des mouvements mécaniques (nouvelle position d'un solide après application d'une force par exemple) ou dans des réactions chimiques (quantité d'un produit résultant d'une réaction par exemple). Les valeurs de ces variables évoluent en fonction de l'écoulement du temps qui passe et des actions physiques qui surviennent. Un système de transitions avec des variables continues exprime des comportements alternant des pas discrets et des pas continus, ces derniers étant habituellement modélisés par une équation différentielle : on appelle de tels systèmes des automates hybrides [Hen96].

**Exemple 7 (Un thermostat - I)** *Un exemple assez classique est celui d'un thermostat régulant la température d'une pièce. Nous reprenons en partie les spécifications de [Dan00]. Un thermostat est la combinaison d'un thermomètre et d'un radiateur de sorte que le radiateur marche en fonction de la température mesurée. On choisit deux températures $T_{min}$ et $T_{max}$, vérifiant $T_{min} < T_{max}$, telles que si la température est jugée trop basse, c'est à dire trop proche de $T_{min}$, le radiateur est mis en route et si elle jugée correcte, c'est-à-dire proche de $T_{max}$, le radiateur est éteint. On note $T_{moy} = (T_{min} + T_{max})/2$ et on choisit $\epsilon > 0$ de sorte que $T_{moy} + \epsilon, T_{moy} - \epsilon \in [T_{min}, T_{max}]$.*

*L'automate hybride donné à la Figure 3.1 modélise un thermostat : la variable $x$ représente la température. Initialement le système est en $s_0$ et $x = T_{moy}$. En l'état $s_0$ atteint avec $x = T$, la température est solution d'une équation différentielle du type*

$$\begin{cases} x(0) & = & T \\ \dot{x} & = & p \end{cases}$$

*avec $p \in [0, 4]$. Alors $x(t) = pt + T$ croît jusqu'à une certaine valeur $T_{moy} + \epsilon \leq T_1 \leq T_{max}$ et puis on atteint l'état de contrôle $s_1$ avec $x = T_1$. Alors la température est solution de l'équation :*

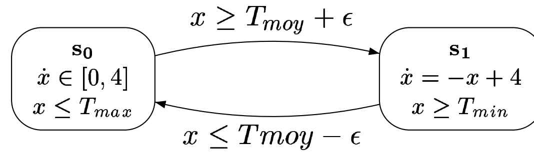



FIG. 3.1 – Exemple d'un automate hybride

$$\begin{cases} x(0) & = & T_1 \\ \dot{x} & = & -x + 4 \end{cases}$$

*Donc,* $x(t) = T_1 e^{-t} + 4(1 - e^{-t})$ *décroît jusqu'à une valeur* $T' \in [T_{min}, T_{moy} - \epsilon]$. *On revient en* $s_0$ *et ce processus se réitère indéfiniment.*

Dans ce chapitre, nous présentons dans la section 3.1 le formalisme des automates temporisés [AD90, AD94] : il s'agit d'une restriction des automates hybrides [Hen96] telle que les variables continues vérifient toujours la même équation différentielle. Il n'y a donc qu'un seul type de variable continue, appelé *horloge*, et toute horloge $x$ satisfait l'équation $\dot{x} = 1$. Nous présentons ensuite le modèle général des automates hybrides [AHH93, Hen96] dans la section 3.2. Dans les deux cas, nous suivrons le plan suivant : en premier lieu nous donnons les définitions formelles des modèles, ensuite nous rappelons les analyses décidables pour ces systèmes et enfin nous présentons des outils de model-checking existant.

## 3.1 Les automates temporisés

Les automates temporisés ont été introduits par Rajeev ALUR et David DILL au début des années 90 [AD90]. Ces automates prennent en compte la dimension temporelle. Pour représenter cette dernière, on introduit le temps dans la sémantique des événements et pour pouvoir mesurer les quantités temporelles, on introduit des horloges. Nous rappelons au préalable certaines notions sur le temps et les horloges.

### 3.1.1 Le temps et les horloges

*Temporiser* un système consiste à introduire des manipulations quantitatives du temps. Pratiquement, on ajoute des variables à valeur réelle qui sont modifiées au cours du temps en fonction de l'évolution de la grandeur qu'elles représentent. La notion de *temps* est vaste en informatique. L'auteur de [Koy92] imagine plusieurs approches du temps. Le temps est un ensemble dont les éléments peuvent être des :

**instants :** cela correspond à la réalité physique, on mesure précisément les instants,

**intervalles :** on se place sur des échelles de temps de durée non nulle, une action survient dans un certain intervalle de temps. Deux actions ont lieu en même temps si elles arrivent dans le même intervalle de temps,

**occurrences d'événements :** dans ce cas, on peut exprimer le moment où une action a eu lieu en fonction des autres actions, à savoir avant ou après.

Le cas standard des systèmes temporisés est le temps des instants et nous ne développerons que ce cas dans la suite. Les relations temporelles engendrées par le temps sont l'égalité ou la

précédence : *avoir lieu en même temps* ou *survenir avant*. Les opérations temporelles possibles sont les post et pré conditions temporelles, *que se passe-t-il si on laisse écouler du temps ?* ou *qu'est-il arrivé auparavant ?* Enfin, il faut déterminer la structure intrinsèque du temps : discret ou dense ; avec origine du temps ou sans origine ; borné ou non ; linéaire ou arborescent à branchement fini ou infini. Pratiquement, le temps est modélisé par un *domaine de temps*.

Un *domaine de temps* [NSY93] $\mathbb{T}$ est un monoïde commutatif contenant 0 muni de la loi additive $+$ vérifiant $\forall t, t' \in \mathbb{T}, t + t' = t \Leftrightarrow t' = 0$ et la relation d'ordre total $\leq$ définie par $t \leq t' \Leftrightarrow \exists t'',\ t + t'' = t'$. Ainsi, 0 est le plus petit élément de $\mathbb{T}$ et $\forall t, t'$, tels que $t \leq t'$, il existe un élément unique $t''$ tel que $t + t'' = t'$ noté $t' - t$. Un domaine est dit dense si pour tout $t, t' \in \mathbb{T}$ tels que $t < t'$, il existe $t''$ est tel que $t < t'' < t'$. Il est appelé discret sinon et dans ce cas tout élément non maximal admet un successeur : $\forall t \in \mathbb{T},\ succ(t) = t'$ avec $\forall t'' > t,\ t'' \geq t'$. Un domaine est linéaire si $\forall t, t' \in \mathbb{T},\ \big(t < t' \vee t = t' \vee t' < t\big)$, sinon il est *arborescent*. Par exemple l'ensemble $\mathbb{T} = \mathbb{N}$ correspond au cas linéaire discret et $\mathbb{T} = \{0,1\}^\star$ muni de l'ordre préfixe à celui arborescent à branchement fini discret [Com01].

**Généralités sur les horloges.** Une horloge suit une loi d'évolution, à savoir elle augmente de façon synchrone avec le temps physique. Dans un système, toutes les horloges avancent à la même vitesse. Le type le plus simple d'horloge est le chronomètre : il mesure le temps écoulé. Les horloges, contrairement aux variables discrètes, ne peuvent être représentées par des automates.

On considère un ensemble fini $X$ de variables à valeurs réelles positives appelées horloges. De même que pour le cas sans temps, on définit le langage du premier ordre $\mathcal{L}_X$ dont le domaine est $\mathbb{R}_{\geq 0}$, les fonctions et les symboles sont les mêmes que pour le langage $\mathcal{L}_Z$ définis à la partie 2.1.3 page 20. Une *valuation d'horloge* sur $X$ est une application $v : X \to \mathbb{R}_{\geq 0}$ qui affecte une valeur réelle positive à chaque horloge de $X$. On note $\mathbb{R}^X_{\geq 0}$ l'ensemble des valuations d'horloges. De même que dans le cas des variables discrètes, une valuation d'horloge représente sémantiquement l'état des horloges à un instant. Pour $t \in \mathbb{R}_{\geq 0}$, la valuation d'horloge $v + t$ est définie par $\forall x \in X, (v+t)(x) = v(x) + t$ et correspond à l'écoulement du temps.

Une *mise à jour d'horloge* (ou *affectation*) est une application $a : X \to \mathcal{F}(\mathcal{L}_X)$ qui à toute horloge $x$ associe une formule $f$ de la forme :

$$f := x \sim c | x \sim y + c$$

avec $\sim \in \{=, <, \leq, \geq, >\}, x, y \in X, c \in \mathbb{Q}$. Si $a$ est l'application particulière $\forall x,\ a(x) \equiv (x = 0) \vee a(x) \equiv tt$, on parle de *reset*. Si $a$ n'utilise que l'égalité dans les formules, on parle de mise à jour déterministe. Les mises à jour non déterministes correspondent souvent à l'intervention de l'environnement. Soit $\mathcal{U}(X)$ l'ensemble des mises à jour d'horloge, pour toute valuation $v$ et pour toute $a \in \mathcal{U}(X)$, $a(v)$ signifie $a(v)(x) = a(x)(v)$. On note également $\mathcal{U}_d(X)$ l'ensemble des mises à jour déterministes.

L'ensemble des *contraintes d'horloges* $\mathcal{C}(X) \subseteq \mathcal{F}(\mathcal{L}_X)$ sur l'ensemble $X$ est défini inductivement par :

$$g := x \backsim r |\ x - y \backsim r\ | g \wedge g\ | g \vee g$$

avec $x, y \in X, \backsim \in \{<, \leq, >, \geq, =\},\ r \in \mathbb{Q}$. Puisqu'une contrainte d'horloge $g$ est une formule particulière, on l'évalue à vrai ou faux et on l'interprète comme telle : $[\![g]\!] \subseteq \mathbb{R}^X_{\geq 0}$ et $g(\nu) = tt \Longleftrightarrow \nu \in [\![g]\!]$.

On note $\mathcal{C}_C(X)$ le sous ensemble propre de $\mathcal{C}(X)$ des contraintes convexes d'horloges défini par :

$$g := x \backsim r |\ x - y \backsim r\ | g \wedge g$$

avec $x, y \in X, \backsim \in \{<, \leq, >, \geq, =\},\ r \in \mathbb{Q}$.

On note $\mathcal{C}_{df}(X)$ [BDFP00] le sous ensemble propre de $\mathcal{C}_C(X)$ des contraintes *rectangulaires* (*diagonal free*) d'horloges défini par :

$$g := x \backsim r \ | g \wedge g$$

avec $x, y \in X, \backsim \in \{<, \leq, >, \geq, =\},\ r \in \mathbb{Q}$.

Si on considère un sous ensemble d'horloges $Y \subseteq X$, pour toute valuation $v \in \mathbb{R}^X_{\geq 0}$ on peut considérer la restriction de $v$ sur $Y$ et alors $v' = v_{|Y} \in \mathbb{R}^Y_{\geq 0}$. De même, pour toute mise à jour d'horloge $a \in \mathcal{U}(X)$, on peut considérer sa restriction sur $Y$ $a_{|Y} \in \mathcal{U}(Y)$. Enfin, pour toute formule $f \in \mathcal{F}(\mathcal{L}_X)$, on peut définir la projection de $f$ sur $Y$ par $proj_Y(f) = \exists z \in X \setminus Y.\ f(z)$, alors $proj_Y(f) \in \mathcal{F}(\mathcal{L}_Y)$.

### 3.1.2 Syntaxe et sémantique des automates temporisés

Pratiquement, un automate temporisé [AD94] est un automate auquel on ajoute des horloges et qui prend en compte l'écoulement de temps. Les comportements d'un automate temporisé sont des alternances d'actions discrètes de $E$ et de pas continus, à savoir d'éléments de $\mathbb{T}$. Une transition discrète se fait en un temps nul.

Nous donnons ici une définition avec des *invariants temporels* [LPY95] et sans *propriété* puisque nous n'en avons pas besoin.

**Définition 8 (Automate temporisé)** *Un automate temporisé est un sextuplet* $\mathcal{A} = (Q, E, X, q_0, I, \rightarrow)$ *tel que : $Q$ est un ensemble fini d'états de contrôle, $E$ est un ensemble fini d'actions, $X$ est un ensemble fini d'horloges, $q_o \in Q$ est l'état initial, $I : Q \rightarrow \mathcal{C}_C(X)$ est une application qui associe à chaque état de contrôle son invariant temporel et $\rightarrow \subseteq Q \times (\mathcal{C}(X) \times E \times \mathcal{U}(X)) \times Q$ est la relation de transition.*

**Définition 9 (Sémantique d'un automate temporisé)** *La sémantique d'un automate temporisé est donnée par le système de transitions $(S, E \cup \mathbb{T}, s_0, \rightarrow_S)$ où :*

1. $S = Q \times \mathbb{R}^X$,
2. $s_0 = (q_0, \nu_0)$ *avec* $\forall x \in X, \nu_0(x) = 0$,
3. $\rightarrow_S \subseteq S \times E \cup \mathbb{T} \times S$ *avec* $((q, \nu), e, (q', \nu')) \in \rightarrow_S \Leftrightarrow$
$$\begin{cases} \textit{soit } e \in E \wedge \exists (q, g, e, a, q') \in \rightarrow \ \textit{ telle que } g(\nu) = tt,\ \nu' \in [\![a(\nu)]\!] \textit{ et } \nu' \in [\![I(q')]\!] \\ \textit{soit } e = \delta \in \mathbb{T} \wedge q' = q \wedge \nu' = \nu + \delta \wedge \forall \delta' \leq \delta, v + \delta' \in [\![I(q)]\!] \end{cases}$$

*Pour tout* $d \in \mathbb{T} \cup E,\ s, s' \in S, (s, d, s') \in \rightarrow$, *on note* $s \rightarrow^d s'$.

Un *état* est un couple $(q, \nu)$ où $q$ est un état de contrôle et $\nu$ est une valuation d'horloges. On note $T(A)$ l'ensemble de l'espace d'états.

Un automate temporisé est *déterministe par rapport au temps* si $\forall d \in \mathbb{T}, \forall s, s_1, s_2 \in Q$, si $s \rightarrow^d s_1$ et $s \rightarrow^d s_2$ alors nécessairement $s_1 = s_2$. C'est toujours le cas d'après notre définition et on attend un tel comportement d'un automate temporisé. Dans l'extension temporisée d'AltaRica, étant donné le comportement des variables de flux, on verra que les modèles temporisés ne sont pas tous déterministes par rapport au temps.

**Exemple 8** *On considère l'automate temporisé donné à la Figure 3.2. Cet automate a deux états de contrôle $s_0$ et $s_1$, une horloge $x$. Initialement, l'automate est en $s_0$ et $x$ vaut* 0. *Un comportement possible est le suivant : le temps s'écoule jusqu'à $x = \delta < 1$, la transition $a$ est tirée, le temps s'écoule à nouveau jusqu'à $x = \delta' \leq 2$ puis la transition $b$ est tirée et l'automate reste indéfiniment en $s_1$.*

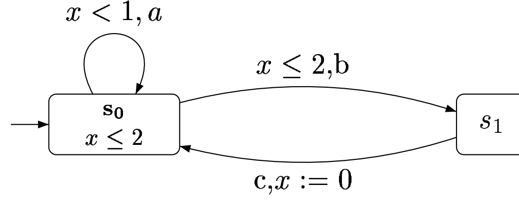



FIG. 3.2 – Exemple d'un automate temporisé

**Synchronisations des automates temporisés.** La synchronisation des automates temporisés [AD94] est une extension du produit synchronisé *à la Arnold-Nivat.* On considère $n$ automates temporisés $\mathcal{A}_i = (Q_i, E_i, X_i, q_{0,i}, I_i, \rightarrow_i)$ et une contrainte de synchronisation $V \subseteq E_1 \cup \{\epsilon\} \times \cdots \times E_n \cup \{\epsilon\}$, on suppose que les ensembles $X_i$ sont deux à deux disjoints. Alors le produit synchronisé des $\mathcal{A}_i$ par rapport à la contrainte $V$ est l'automate temporisé $\mathcal{A} = \mathcal{A}_1 \parallel_V \cdots \parallel_V \mathcal{A}_n = (Q, V, X, q_0, I, \rightarrow)$ défini par :

1. $Q = Q_1 \times \cdots \times Q_n$ ;
2. $X = X_1 \cup \cdots \cup X_n$ ;
3. $q_o = (q_{0,1}, \cdots, q_{0,n})$ ;
4. $I(q) = \bigwedge I_i(q_i)$ ;
5. $\rightarrow \subseteq Q \times (\mathcal{C}(X) \times V \times \mathcal{U}(X)) \times Q$ avec : $\big((q_1, \cdots, q_n), (\gamma, e, R), (q'_1, \cdots, q'_n)\big) \in \rightarrow \Longleftrightarrow e \in V \wedge \forall i \in [1, n]$, on a $\begin{cases} soit\ e_i = \epsilon \wedge q'_i = q_i \wedge proj_{X_i}(\gamma) = tt \wedge R_{|X_i} = id \\ soit\ e_i \in E_i \wedge \big(q_i, (proj_{X_i}(\gamma), e_i, R_{|X_i}), q'_i\big) \in \rightarrow_i \end{cases}$

**Langages temporisés** Le langage temporisé reconnu par un automate temporisé de domaine de temps $\mathbb{T}$ est un sous ensemble de $(E \times \mathbb{T})^\infty = (E \times \mathbb{T})^\star \cup (E \times \mathbb{T})^\omega$. Un mot temporisé est une suite $u = (e_1, t_1)(e_2, t_2) \cdots$ avec $\forall i, t_{i+1} \geq t_i$. La *suite de temps* $(t_i)_{i \geq 0}$ marque les instants où les événements $e_i$ surviennent. On dit que la suite *diverge* si sa limite vaut l'infini. On dit qu'elle est *stricte* si tous ses éléments sont distincts et si elle diverge.

Un *chemin* est une suite finie ou non de transitions : $p = q_0 \longrightarrow^{(g_1, e_1, a_1)} q_1 \longrightarrow^{(g_2, e_2, a_2)} q_2 \cdots$ avec $\forall i \in \mathbb{N}, (q_i, g_i, e_i, a_i, q_{i+1}) \in \rightarrow$. Une *exécution* $r$ suivant le chemin $p$ pour le mot temporisé $u$ est : $r = (q_0, v_0) \longrightarrow_{t_1}^{(g_1, e_1, a_1)} (q_1, v_1) \longrightarrow_{t_2}^{(g_2, e_2, a_2)} (q_2, v_2) \cdots$ avec $\forall i \in \mathbb{N}, q_i \in Q, v_i$ valuation d'horloge tels que $(vi + t_i - t_{i+1}) \models g_i$, $\forall t \in [0, t_{i+1} - t_i), v_i + t \models I(q_i)$ et $v_i = a_i(v_{i-1} + t_i - t_{i-1})$.

De même que dans le cas général, on peut ajouter une $\epsilon$-transition. Un mot temporisé reste néanmoins dans $(E \times \mathbb{T})^\infty$, on abstrait les occurrences de la transition inobservable $\epsilon$.

**Exemple 9** *Le langage reconnu par l'automate temporisé donné à la Figure 3.2 est (on adopte les notations des expressions temporisées régulières [ACM02] :*

$$L_1 = (<< a^\omega >_{[0,1[}\ b >_{[0,2]}\ c)^\omega << a^\omega >_{[0,1[}\ b >_{[0,2]}$$

*En effet, on résout le système d'équations en utilisant la méthode de [ACM02]) :*
$\begin{cases} X_0 = < a >_{[0,1[} \circ X_0 + < b >_{[0,2]} X_1 \\ X_1 = cX_0 + \epsilon \end{cases}$ *. Alors* $X_0 = < a >_{[0,1[} \circ X_0 + < b >_{[0,2]} cX_0 + < b >_{[0,2]}$.
*D'où* $X_0 = (< a >_{[0,1[})^{\circledast} \circ (< b >_{[0,2]} cX_0 + < b >_{[0,2]}) = << a^\omega >_{[0,1[}\ b >_{[0,2]} cX_0 + << a^\omega >_{[0,1[}\ b >_{[0,2]} = L_1$.

### 3.1.3 Le modèle des automates temporisés : avantages et inconvénients

De même que pour les automates à variables, on peut considérer des automates temporisés à variables. Pour les mêmes raisons que dans la section 2.1.4, page 21, un automate temporisé à variables résulte de la synchronisation de plusieurs automates temporisés. On obtient la même famille d'automates.

Toutes les analyses possibles sur le système dépendent du problème d'*accessibilité*[1] : trouver l'espace des états accessibles. Pour les automates temporisés, l'accessibilité dépend à la fois des gardes et des affectations [BDFP00]. La Figure 3.1[2] est tirée de [BFP00] et résume les résultats de [AD90, AD94, BDFP00, Bou02a].

<table>
<tr><th colspan="2">Affectation déterministe</th><th>Garde dans $\mathcal{B}_{df}(X)$</th><th>Garde dans $\mathcal{B}(X)$</th></tr>
<tr><td>$x := c$</td><td>(1)</td><td rowspan="4">Décidable</td><td rowspan="2">Décidable</td></tr>
<tr><td>$x := y$</td><td>(2)</td></tr>
<tr><td>$x := x + 1$</td><td>(3)</td><td rowspan="3">Indécidable</td></tr>
<tr><td>$x := y + c$</td><td>(4)</td></tr>
<tr><td>$x := x - 1$</td><td>(5)</td><td>Indécidable</td></tr>
<tr><th colspan="2">Affectation non déterministe</th><th>Garde dans $\mathcal{B}_{df}(X)$</th><th>Guards in $\mathcal{B}(X)$</th></tr>
<tr><td>$x :< c$</td><td>(6)</td><td rowspan="4">Décidable</td><td>Décidable</td></tr>
<tr><td>$x :> c$</td><td>(7)</td><td rowspan="4">Indécidable</td></tr>
<tr><td>$x :\backsim y + c$</td><td>(8)</td></tr>
<tr><td>$y + c <: x :< y + d$</td><td>(9)</td></tr>
<tr><td>$y + c <: x :< z + d$</td><td>(10)</td><td>Indécidable</td></tr>
</table>

TAB. 3.1 – Résultats d'accessibilité des automates temporisés

Les calculs d'accessibilité reposent sur la construction du *graphe des régions* [AD94] associé à un automate temporisé. Ce graphe est une représentation finie de l'espace d'états qui, lui, est infini. Le calcul des régions est basé sur le calcul des classes d'une relation d'équivalence sur les états.

**Le respect des échéances temporelles.** On exige souvent d'un système temps réel qu'il soit conforme à des délais temporels. D'un point de vue événementiel, on veut que l'action $e$ une fois possible se réalise avant un certain délai ou alors le système change d'état avant l'expiration du délai. Cette approche est proposée par les *automates à deadline* [SY96] : dans un automate à deadline, on associe à chaque transition une garde, une affectation et une échéance. La garde et l'affectation correspondent à la définition d'une transition dans un automate temporisé. L'échéance est un attribut supplémentaire qui contraint le comportement du système vis à vis de la garde. Trois types de comportements sont décrits :

**Eager** (urgente) [SY96, BST98, BS00] si une transition est *eager*, alors dès que la garde est vraie, le temps est bloqué et le système doit réaliser la transition eager.

**Delayable** (retardable) [SY96, BGS00] si une transition est *delayable*, le système peut ou non tirer la transition pendant que la garde est vraie. Seulement au dernier instant où elle est vraie

[1] Etant donné un ensemble $U$ de toutes les données possibles (un univers) et un ensemble $B$ de toutes les données correctes $B \subseteq U$ pour le problème $P$, le problème P est décidable si, et seulement si il existe un algorithme pour P (c'est-à-dire une procédure qui s'arrête et répond *oui* si l'entrée $x \in B$ et répond *non* si l'entrée $x \notin B$).

[2] Les contraintes non diagonales ont été définies dans la section 3.1.1 page 41.

(ou au premier instant où elle est fausse si la garde est ouverte à droite), le temps est bloqué et le système doit réagir.

**Lazy** (paresseuse) [SY96, BGS00] si une transition est *lazy*, elle peut être ou non tirée pendant sa validité.

On peut exprimer une transition delayable à partir d'un comportement lazy et eager. En effet tant que la garde est vraie et tant qu'il est possible de laisser écouler du temps, la transition est lazy et dès qu'on ne peut plus laisser écouler de temps sans que la garde deviennent fausse, la transition devient eager. Si la garde n'est pas fermée, par exemple $x > 30$, la transition devient eager à l'instant où la garde devient fausse et peut encore à ce moment être tirée. Ce résultat est démontré dans [BST98].

Cette approche est équivalente à celle que nous avons présentée dans la Définition 8 page 42 qui consiste à spécifier des invariants aux états. En effet, si la transition est lazy, l'invariant est $tt$, si elle est delayable, l'invariant correspond à la garde. Enfin si la transition est urgente, on rajoute une horloge $y$ au système qui est remise à zéro à chaque transition et on ajoute l'invariant $x = 0$ dans les états ayant des transitions sortantes eager. Il faut cependant faire attention à fermer les conditions car la sémantique des automates temporisés indique qu'une garde doit être vraie pour qu'une transition soit tirable. Une démonstration complète et détaillée est faite dans [Lab98]. Ainsi l'expressivité des deux modèles est absolument identique.

### 3.1.4 Des outils pour les automates temporisés

Dans cette partie, nous présentons trois outils de vérification d'automates temporisés. Notre choix s'est porté sur ces trois model-checker temporisés particuliers car leurs approches sont différentes. Le premier outil UPPAAL [BLL+95] est essentiellement développé par Wang Yi, Kim G. Larsen et Paul Pettersson dans les Universités d'Uppsala (UPP) en Suède et d'Aalborg (AAL) au Danemark. Ce model-checker est assez facile d'utilisation puisqu'il propose une interface graphique et a montré son efficacité au travers de plusieurs études de cas [HLS99, LPY01, BGK+02]. Le deuxième outil KRONOS [DOTY95] est également assez abouti, il est développé au laboratoire Verimag de l'Université de Grenoble par Conrado Daws, Alfredo Olivero, Stavros Tripakis et Sergio Yovine. Plusieurs études de systèmes et protocoles ont été réalisées à l'aide de KRONOS [MY96, TY98, NY00]. Le dernier model-checker CMC [LL98a] essentiellement développé par François Laroussinie au LSV à l'ENS de Cachan adopte une démarche un peu différente des deux premiers model-checker.

Aucun de ces outils n'est hiérarchique, un système est modélisé par un réseau d'automates temporisés se synchronisant.

On a vu que la décidabilité de l'accessibilité est prouvée en utilisant la construction de l'automate des régions. Cette construction fournit une abstraction correcte du comportement des automates temporisés, mais ne permet hélas pas une implémentation naturelle, notamment à cause de l'explosion de la taille de l'espace d'états. Une alternative pour représenter de manière finie l'espace infini d'états est l'utilisation de *zones d'horloges* qui correspondent également à un ensemble de valuations d'horloges. La structure de données correspondant à cette approche est la *difference bounded matrix* (DBM) [Dil89]. Les model-checker présentés ici utilisent cette structure de données ou des variantes.

#### UPPAAL

UPPAAL [BLL+95, PL00, SBB+99] permet de modéliser des automates temporisés communiquant par synchronisation et de les étudier.

**Synchronisations :** l'outil permet simplement les synchronisations binaires. Le modèle est celui des émissions/réceptions d'événements.

**Variables partagées :** on peut partager des horloges, mais on peut aussi spécifier des variables entières qui peuvent également être partagées.

**Model-checker :** la logique temporelle utilisée est un fragment de TCTL [HNSY92, ACD93] et celui-ci ne permet que vérifier des propriétés d'accessibilité. En effet, dans la logique d'UPPAAL, les opérateurs de branchement ne peuvent être encastrés les uns dans les autres.

Le point fort d'UPPAAL sont les temps de calcul rapides car les algorithmes sont optimisés. Cela résulte de l'utilisation de structures de données très compactes [BBD+02], de parcours de l'espace d'états à la volée et de l'utilisation d'accélération. En effet, le calcul d'accessibilité se fait dans UPPAAL par des algorithmes d'analyse avant. Or le calcul exact des successeurs ne termine pas toujours donc des méthodes d'abstraction, comme les opérateurs d'élargissement, sont utilisés.

Le problème est que pour l'opérateur d'élargissement utilisé et implanté dans UPPAAL, il est impossible de trouver un algorithme d'accessibilité exact utilisant cet opérateur [Bou02b].

Le model-checker fournit des diagnostics d'erreur : si une propriété n'est pas vérifiée, l'outil construit une trace violant la propriété.

**Options :** on peut spécifier des invariants temporels, des actions urgentes.

**Outils dans l'outil :** UPPAAL fournit une interface graphique particulièrement conviviale et un simulateur (pas à pas ou aléatoire). On peut également modéliser des automates temporisés avec des coûts [LBB+01] sur les arcs ou les états de contrôle et chercher des chemins à moindre coût.

KRONOS

KRONOS [DOTY95, Yov97, SBB+99] modélise des réseaux d'automates synchronisés et utilise la logique *TCTL*. Un automate temporisé est exprimé sous une forme textuelle.

**Synchronisations :** les transitions sont étiquetées par des ensembles d'événements, la transition résultant d'une synchronisation est la réunion de tous les événements des transitions impliquées. Cette opération n'est pas associative.

**Variables :** il n'y a que des horloges et pas de variables entières.

**Model-checker :** le point fort de KRONOS est l'expressivité de sa logique : KRONOS implante un algorithme de model-checking pour la logique temporelle TCTL. On peut ainsi vérifier des propriétés de sûreté comme de vivacité. Cet outil est le plus expressif en terme de logique de tous ceux présentés dans cette section.

On peut choisir les algorithmes d'analyse avant ou d'analyse arrière. On peut utiliser des parcours à la volée. Pour l'analyse avant, KRONOS a le même problème [Bou02b] que UPPAAL.

**Réductions :** KRONOS propose des techniques de réductions de l'espace d'états afin d'optimiser les vérifications. Il intègre des techniques d'ordre partiel [Pag96] et surtout propose d'optimiser le nombre d'horloges dans un système [DY96]. L'idée est d'éliminer les horloges inutiles en fonction des gardes et remises à jour des transitions. Dans KRONOS, seules certaines réductions sont implantées puisque *trouver le nombre minimal d'horloges nécessaire pour un automate temporisé* est un problème indécidable [Wil94] et que l'heuristique proposée est NP-COMPLET.

**Options :** on peut spécifier des invariants temporels. L'outil génère des contre exemples pour des propriétés non vérifiées.

CMC

CMC [LL98a, Lar99] (Compositional Model-Checking) modélise également des réseaux d'automates temporisés. Ce model-checker repose sur une approche compositionnelle des vérifications. Il n'est pas le seul, Mocha [AHM+98] est également un model-checker compositionnel.

La *vérification compositionnelle* [And95, LL95] traite le problème sous un autre angle : elle tire partie de la composition des automates. Etant donné un système concurrent $\mathcal{S} = \mathcal{S}_1 \parallel \cdots \parallel \mathcal{S}_n$, on veut répondre à la question *est-ce que* $\mathcal{S}$ *satisfait la formule* $\phi$ *?* sans mettre à plat le système. On quotiente la formule $\phi$ par les automates récursivement en partant de la propriété : $\mathcal{S}_1 \parallel \cdots \parallel \mathcal{S}_n \models \phi \Leftrightarrow \mathcal{S}_1 \parallel \cdots \parallel \mathcal{S}_{n-1} \models \phi \setminus \mathcal{S}_n$.

**Général :** les automates temporisés sont décrits textuellement par un ensemble d'horloges, de nœuds, d'invariants et de transitions gardées par des contraintes d'horloges. Une table décrit les vecteurs de synchronisation pour définir le réseau d'automates temporisés. Il n'y a pas de variables entières dans cette version ce qui n'est pas le cas dans HCMC [3] [CL00].

**Model-checker :** les propriétés sont exprimées avec $L_\mu$ un fragment du *T-$\mu$-calcul* [HNSY92, LLW95]. $L_\mu$ ne contient pas l'opérateur de plus petit point fixe, il n'y a donc pas d'alternances.

**Optimisations :** plusieurs stratégies de simplification de formules de $L_\mu$ sont implantées dans CMC. Les principales réductions consistent à simplifier les formules booléennes, éliminer les redondances. Un réduction particulière propre à l'approche compositionnelle, appelée *hitzone*, simplifie des contraintes par des approximations en fonction de l'état initial.

Tous les résultats et généralités rappelés sur les automates temporisés nous seront utiles pour construire le langage AltaRica temporisé. Nous introduirons les horloges dans le modèle et nous introduirons le temps dans les systèmes de transitions interfacés donnés à la Définition 2 page 25 pour exprimer la sémantique des automates à contraintes temporisés. Une fois ce travail théorique réalisé, nous traduirons nos modèles en automates temporisés d'UPPAAL et nous vérifierons des propriétés d'atteignabilité. Dans la prochaine section, nous présentons le modèle plus général des automates hybrides [Hen96].

## 3.2 Les automates hybrides

Un automate hybride [Hen96] est un automate à variables continues qui prend en compte le temps. En chaque état de contrôle, une loi d'évolution du type équation différentielle $f\left(\dot{x}(t), x(t)\right) = 0$ est spécifiée pour chaque variable. Les automates à variables discrètes considérés dans la partie 2.1.4 page 21 sont un cas particulier des automates hybrides où les variables suivent la loi $\dot{x} = 0$ pour toute variable $x$ et en tout état de contrôle. Les automates temporisés introduits précédemment sont également un cas particulier et les horloges vérifient $\dot{x} = 1$ en tout état de contrôle. On généralise encore la notion d'automate ce qui va permettre de modéliser encore plus finement des systèmes mais en contre partie va engendrer des systèmes dont l'étude sera parfois impossible car l'accessibilité sera indécidable.

**Notations.** On considère un ensemble fini de variables continues $X$. On définit le langage du premier ordre $\mathcal{L}_X$ dont le domaine est $\mathbb{R}$, les fonctions et les symboles sont les mêmes que pour le langage $\mathcal{L}_Z$ défini à la partie 2.1.3 page 20. Une *valuation de variables continues* sur $X$ est une application $v : X \to \mathbb{R}$ qui assigne une valeur réelle à chaque variable de $X$. On note $\mathbb{R}^X$ l'ensemble des valuations de variables. Si $|X| = n$, une valuation $v$ peut s'interpréter comme un vecteur $\overline{v} \in \mathbb{R}^n$.

[3] L'extension hybride de CMC

Une *expression linéaire* sur $X$ est de la forme $c + \Sigma_i c_i x_i$ avec $c, c_i \in \mathbb{Z}$ et $x_i \in X$. L'ensemble des *contraintes linéaires* $\mathcal{C}_l(X) \subseteq \mathcal{F}(\mathcal{L}_X)$ sur l'ensemble $X$ est défini inductivement par :

$$g := e_1 \backsim e_2 \; | g \wedge g \; | g \vee g$$

avec $e_1, e_2$ sont des expressions linéaires, $\backsim \in \{<, \leq, >, \geq, =\}$.

Une *inégalité rectangulaire* sur $X$ est une formule de la forme $x \sim c$ avec $c \in \mathbb{Q}$ et $\sim \in \{<, \leq, =, \geq, >\}$. Un *prédicat rectangulaire* est une conjonction d'inégalités triangulaires. On note $Rect(X)$ l'ensemble des prédicats rectangulaires.

Une *affectation linéaire* sur $X$ est de la forme $\overline{v} := A.\overline{v} + b$ avec $A \in M_n(\mathbb{Z})$ et $b \in \mathbb{Z}^n$. On note l'affectation $\alpha = (A, b)$ et on écrit $\overline{v} := \alpha(\overline{v})$. On note $\mathcal{U}_l(X)$ l'ensemble des affectations linéaires.

On considère en outre pour tout ensemble de variables continues $X = \{x_1, \cdots, x_n\}$, l'ensemble des variables pointées $\dot{X} = \{\dot{x}_1, \cdots, \dot{x}_n\}$ tel que $X \cap \dot{X} = \emptyset$. Ces variables correspondent aux dérivées des variables continues.

### 3.2.1 Syntaxe et sémantique des automates hybrides

Dans cette partie, nous définissons formellement les automates hybrides [AHH93, Hen96]. Nous avons choisi de les exprimer sous la forme d'automates étendus [CL00] ce qui est parfaitement équivalent à la définition standard d'un graphe direct fini [Hen96].

**Définition 10 (Automate hybride)** *Un automate hybride est un octuplet* $\mathcal{H} = (Q, E, X, q_0, init, I, F, \rightarrow)$ *où :*

1. *$Q$ est un ensemble fini d'états de contrôle ;*
2. *$E$ un ensemble fini d'événements ;*
3. *$X = \{x_1, \cdots, x_n\}$ est un ensemble fini de variables continues, $n$ est appelé la dimension de $\mathcal{H}$. On note $\dot{X} = \{\dot{x}_1, \cdots, \dot{x}_n\}$ l'ensemble des variables pointées ;*
4. *$q_0$ est l'état de contrôle initial et les valeurs des variables continues initiales sont données par le prédicat $init \in \mathcal{C}_l(X)$ ;*
5. *l'*invariant *$I$ est une application de $Q \rightarrow \mathcal{C}_l(X)$ associant à chaque état de contrôle un prédicat de variables qui sera satisfait tant que le système restera en cet état de contrôle. De même que pour les automates temporisés, cet invariant sera convexe ;*
6. *la* condition de flot *$F : Q \rightarrow \mathcal{C}_l(X \cup \dot{X})$ associe à chaque état de contrôle les lois d'évolution des variables sous forme d'une formule sur $X \cup \dot{X}$. Alors, $\forall q \in Q$, $F(q)$ est l'ensemble des équations différentielles $\dot{x}(t) = f\big(x(t), t\big)$ avec $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ continue et lipschitzienne en la deuxième variable[4] ;*
7. *$\rightarrow \subseteq Q \times (\mathcal{C}_l(X) \times E \times \mathcal{U}_l(X)) \times Q$ est la relation de transition. Alors, $t = (q, \gamma, e, \alpha, q') \in \rightarrow$ représente un arc entre les états de contrôle $q$ et $q'$ avec la garde $\gamma$, l'étiquette $e$ et l'affectation linéaire $\alpha$.*

**Exemple 10 (Un thermostat - II)** *Le Figure 3.1 de l'exemple 7 page 39 représente l'automate hybride* $\mathcal{H} = (Q, E, X, s_0, init, I, F, \rightarrow)$ *avec* $Q = \{s_0, s_1\}$, $E = \emptyset$, $X = \{x\}$, $init \equiv x = 0$, $I(s_0) \equiv x \leq T_{max}$, $I(s_1) \equiv x \geq T_{min}$, $F(s_0) \equiv \dot{x} = -x + 4$, $F(s_1) \equiv \dot{x} = -x$ *et* $\rightarrow = \{(s_0, x \geq T_{moy} + \epsilon, (1,0), s_1), (s_1, x \leq T_{moy} - \epsilon, (1,0), s_0)\}$.

[4] Ainsi on est assuré de l'existence de solutions et de l'unicité de la solution pour toute condition initiale $(t_0, x_0)$ d'après le théorème de Cauchy Lipschitz

**Définition 11 (Sémantique d'un automate hybride)** *Soit $\mathcal{H} = (Q, E, X, s_0, init, I, F, \rightarrow)$ un automate hybride, la sémantique de $\mathcal{H}$ est donnée par le système de transitions $(S, E \cup \mathbb{T}, S_0, \rightarrow_S)$ où $S = Q \times \mathbb{R}^n$, $S_0 = \{(q_0, \overline{v}) | \overline{v} \in [\![init]\!]\}$ et $\rightarrow_S \subseteq S \times E \cup \mathbb{T} \times S$ est définie par :*

$$((q,\overline{v}), e, (q', \overline{v}')) \in \rightarrow_S \Leftrightarrow \begin{cases} \text{soit } e \in E \wedge \exists (q, g, e, \alpha, q') \in \rightarrow \text{ telle que } \overline{v} \in [\![g]\!],\ \overline{v}' \in [\![\alpha(\overline{v})]\!], \\ \quad \text{et } \overline{v}' \in [\![I(q')]\!] \\ \text{soit } e = \delta \in \mathbb{T} \wedge q' = q \wedge \exists f \in F(q) \wedge \text{ soit } x \text{ la solution de} \\ \quad \begin{cases} \dot{x}(t) &= f\big(x(t), t\big) \\ x(0) &= \overline{v} \end{cases} \\ \quad \overline{v}' = x(\delta) \wedge \forall \delta' \leq \delta,\ x(\delta') \in [\![I(q)]\!] \end{cases}$$

*Pour tout $d \in \mathbb{T} \cup E$, $s, s' \in S$, $(s, d, s') \in \rightarrow$, on note $s \rightarrow^d s'$. Un état de $H$ est une paire $(q, \overline{v}) \in Q \times \mathbb{R}^n$.*

**Définition 12 (Trajectoire temporelle hybride [LJS+01])** *Une* trajectoire temporelle hybride *$\tau$ est une suite, finie ou non, d'intervalles de $\mathbb{R}$, $\tau = (I_n)_n$ vérifiant :*

1. *pour tout $n$, $I_n$ est fermé sauf si $\tau$ est fini, auquel cas le dernier intervalle est simplement fermé à gauche,*
2. *si $I_i = [l_i, L_i]$ alors $\forall i, l_i \leq L_i$ et $\forall i > 0$, $l_i = L_{i-1}$.*

*On note $\mathcal{T}$ l'ensemble des trajectoires temporelles hybrides.*

**Définition 13 (Trajectoire - exécution - trace [LJS+01])** *Une* trajectoire *$r$ d'un automate hybride $H$ est un triplet $r = (\tau, q, x)$ avec $\tau = (I_i)_i = ([l_i, L_i])_i \in \mathcal{T}$ une trajectoire temporelle, $q : \cup I_i \rightarrow Q$ et $x : \cup I_i \rightarrow \mathbb{R}^n$ tels que :*

- Pas continu *: $\forall i$ tel que $l_i < L_i$, $x$ est continu sur $I_i$, $q$ est constante sur $I_i$ et pour tout $t \in [l_i, Li)$, $x(t) \in I(q(t))$ et $\dot{x}(t) \in F(q(t))$ ;*
- Pas discret *: $\forall i$, $\exists \big(q(L_i), \gamma, e, R, q(l_{i+1})\big) \in \rightarrow$ avec $x(L_i) \in \gamma$ et $x(l_{i+1}) = R(x(L_i))$.*

*On dit alors que l'automate hybride $H$ accepte la trajectoire $r$.*

*On appelle* exécution *une trajectoire commençant en l'état initial, c'est-à-dire qu'elle vérifie la condition initiale $\big(q(\tau_0), x(\tau_0)\big) \in [\![Init]\!]$.*

*On appelle* trace *une suite $\pi = (\pi_i)$ d'éléments de $Q$ telle que $\pi$ est la projection sur $Q$ d'une trajectoire acceptée par l'automate hybride.*

Un état $(q, \nu)$ de $H$ est *atteignable* s'il est le dernier état d'une exécution finie.

**Exemple 11 (Un thermostat - III)** *On considère à nouveau l'automate hybride du thermostat Figure 3.1 page 40. On choisit la trajectoire temporelle hybride $\tau = [0, 0.5], [0.5, 0.7], [0.7, 1.2[$. Alors une exécution possible est $r = (\tau, q, y)$ représentée dans la Figure 3.3. L'exécution $r$ est définie :*

- *sur $\tau_0 = [0, 0.5]$ par $q_{|\tau_0} = s_0$ et $\dot{x} = 4$, $x(0) = 20$ ;*
- *sur $\tau_1 = [0.5, 0.7]$ par $q_{|\tau_1} = s_1$ et $\dot{x}_{|\tau_1} = -x_{|\tau_1} + 4$, $x_{|\tau_1}(0.5) = 22$ ;*
- *sur $\tau_2 = [0.7, 1.2]$ par $q_{|\tau_2} = s_0$ et $\dot{x}_{|\tau_2} = 2$ avec $x_{|\tau_2}(0.7) = 18.73$.*

### 3.2.2 Composition d'automates hybrides

La synchronisation des automates hybrides, de même que la synchronisation des automates temporisés définie à la section 3.1.2 page 43, est une extension du produit synchronisé *à la Arnold-Nivat.* On considère $n$ automates hybrides $H_i = (Q_i, E_i, X_i, q_{0,i}, init_i, I_i, F_i, \rightarrow_i)$ et une contrainte

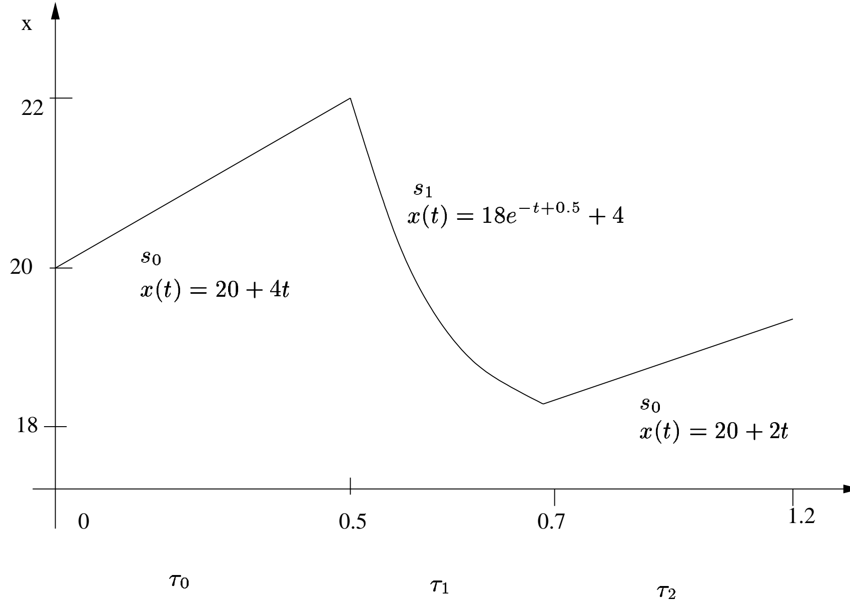



FIG. 3.3 – Une exécution de l'automate hybride thermostat

de synchronisation $V \subseteq E_1 \times \cdots \times E_n$, on suppose que les ensembles $X_i$ sont deux à deux disjoints sinon on renomme les variables. Alors le produit synchronisé des $H_i$ par rapport à la contrainte $V$ est l'automate hybride $H = H_1 \parallel_V \cdots \parallel_V H_n = (Q, V, X, q_0, init, I, F, \rightarrow)$ défini par :

1. $Q = Q_1 \times \cdots \times Q_n$ ;
2. $q_o = (q_{0,1}, \cdots, q_{0,n})$ ;
3. $X = X_1 \cup \cdots \cup X_n$ ;
4. $I(q_1, \cdots, q_n) = \bigwedge I_i(q_i)$ ;
5. $F(q_1, \cdots, q_n) = \bigwedge F_i(q_i)$ ;
6. $\rightarrow \subseteq Q \times (\mathcal{C}_l(X) \times V \times \mathcal{U}_l(X)) \times Q$ avec $\big((q_1, \cdots, q_n), (\gamma, e, R), (q'_1, \cdots, q'_n)\big) \in \rightarrow \Longleftrightarrow e \in V \wedge \forall i \in [1, n]$, on a $\begin{cases} e_i = \epsilon \wedge q'_i = q_i \wedge proj_{X_i}(\gamma) = tt \wedge R_{|X_i} = id \\ e_i \in E_i \wedge \big(q_i, (proj_{X_i}(\gamma), e_i, proj_{X_i}(R)), q'_i\big) \in \rightarrow_i \end{cases}$

### 3.2.3 Décidabilité dans les automates hybrides

**Définition 14 (Sous classes d'automates hybrides [AHLP00, HM00, Ade03])** *On dit qu' un automate hybride est :*

- *réinitialisé : si $\forall e \in E$, toutes les variables continues sont mises à jour par $e$ ;*
- *c-réinitialisé : si pour tout cycle, il existe une transition qui remet à jour toutes les variables continues ;*
- *initialisé : si pour toute transition $(q, \gamma, e, R), q') \in \rightarrow$, pour tout $x \in X$ tel que $F(q)(x) \neq F(q')(x)$ alors $x$ est remise à jour par $R$ ;*
- *rectangulaire : si tous les prédicats sont rectangulaires, i.e. la condition initiale $init \in Rect(X)$, l'invariant $I : Q \rightarrow Rect(X)$, la condition de flot $F : Q \rightarrow Rect(\dot{X})$ et $\rightarrow \subseteq Q \times (Rect(X) \times E \times \big(Rect(X') \cup \{x' = x\}\big) \times Q$.*
- *rectangulaire $qD$ : si $H$ est un automate hybride rectangulaire ayant exactement $q$ variables continues ;*

- *rectangulaire fixe : pour toute variables $x$, il existe un intervalle $I$ tel que en tout état de contrôle $\dot{x} \in I$ ;*
- *singulier : si $H$ est rectangulaire fixe et que tous les intervalles $I$ sont des singletons ;*
- *polynômial : si en tout état de contrôle $q$, s'il existe une matrice nilpotente $A \in M_n(\mathbb{R})$ telle que $F(q)$ est définie par $\vec{\dot{x}} = A\vec{x}$. Dans ce cas, on résout un système linéaire d'équations et les solutions seront des polynômes fonction du temps.*

Les automates hybrides sont difficilement étudiables car pour la plupart, l'accessibilité est indécidable [HKPV95]. Au delà de deux variables continues, il faut des hypothèse assez fortes pour que l'accessibilité soit décidable : les propriétés de (ré)initialisation ont cette particularité mais ne sont pas toujours facile à prouver.

Le Tableau 3.2 résume les résultats de décidabilité et nous nous inspirons de [HM00, Ade03, HKPV95, AHLP00]. On remarque que les logiques vont en décroissant et que l'accessibilité est décidable pour toutes ces familles.

| Logique décidable | Automates hybrides |
|---|---|
| $\mu$-calcul | automates hybrides singuliers<br>automates hybrides polynômiaux réinitialisés<br>automates hybrides réinitialisés<br>automates hybrides c-réinitialisés |
| fragment du $\mu$-calcul sans la négation | automates hybrides 2D initialisés |
| LTL | automates hybrides rectangulaires initialisés |
| fragment du $\mu$-calcul sans négation, conjonction, plus grand point fixe | |
| logique d'accessibilité bornée | réseau d'automates temporisés |

TAB. 3.2 – Résultats de décidabilité

### 3.2.4 Les outils pour les automates hybrides

Dans cette section, nous présentons un outil dédié à l'étude de réseaux d'automates hybrides.

HYTECH

HYTECH [HHWT97] est développé à l'Université de Californie à Berkeley par Tom Henzinger, Pei-Hsin Ho, et Howard Wong-Toi. Cet outil permet de représenter des automates hybrides polyédriques, i.e. toutes les formules sont linéaires, interagissant par des communications. Il n'y a pas d'interface graphique. Dans le cas hybride, il y a toujours des variables discrètes et des horloges.

**Synchronisations :** les spécifications ne sont pas hiérarchiques. Les synchronisations multiples : les transitions synchronisées portent la même étiquette.

**Model-checking :** les opérateurs d'une logique temporelle ne sont pas implantés et l'utilisateur doit écrire son propre calcul de model-checking symbolique à partir des opérateurs (opérateurs ensemblistes $\cap, \cup$, $Pre$, $Post$, reachforward, reachbackward). Par exemple pour l'accessibilité

```
 init:= P
atteign:= reach forward from init endreach
```

Les algorithmes sont basés sur [AHH93] et résultent de calcul polyédrique.

L'outil propose un calcul arrière et avant. En cas de non respect d'une propriété, l'outil exhibe des diagnostiques d'erreur.

**Analyse paramétrée :** on peut spécifier des paramètres et vérifier un certain nombre de propriétés en fonction de ce paramètre. HYTECH réalise un calcul symbolique de propriétés à partir de contraintes polyèdriques dépendant du paramètre. Ces paramètres ne sont que sur les délais ou les affectations. Si on ajoute un paramètre sur une condition de flot, les contraintes ne restent plus linéaires.

Dans cette première partie, nous avons décrit le contexte des travaux qui seront exposés dans les parties suivantes. L'objectif est de développer un langage de haut niveau de modélisation des systèmes temps réel, en conservant d'une part les spécificités d'AltaRica, et en ajoutant d'autre part des opérateurs aux automates à variables continues pour les rendre hiérarchiques. AltaRica est un langage à variables et intégrer des variables continues se fait assez naturellement. Nous construirons en premier lieu Timed AltaRica dans le chapitre 4 : nous exprimerons la syntaxe et la sémantique et nous étendrons la notion de priorité afin de prendre en compte le temps. Ensuite, dans le chapitre 5, nous étendrons davantage le langage et nous proposerons un langage Hybrid AltaRica.

## Deuxième partie

# Le langage AltaRica temps réel

# Chapitre 4

# Le langage Timed AltaRica

Dans ce chapitre nous proposons une extension temporisée du langage AltaRica conservant les opérateurs d'AltaRica comme les coordinations de flux, les broadcast ou encore les priorités que nous étendrons de façon à prendre en compte des aspects temporels. Cette construction va se faire pas à pas et s'inspire d'un part de la construction du langage AltaRica et d'autre part de la définition des automates temporisés présentés précédemment.

Nous étendons la syntaxe dans la section 4.1 : nous introduisons naturellement les horloges et nous reprenons les notations et définitions de valuations, affectations et contraintes d'horloges données dans la partie 3.1.1 page 41. Ensuite, nous étendons les STI pour obtenir des *systèmes de transitions temporisés interfacés* et les notions de bisimulation pour ces nouveaux systèmes. Dans la section 4.2, nous ajoutons l'urgence [SY96, BST98, BS00] et les priorités temporelles [BST98, BS00, BGS00] dans Timed AltaRica. Dans la section 4.3, nous construisons la hiérarchie et nous démontrons, de même que pour AltaRica, un théorème de réécriture d'un nœud Timed AltaRica en un composant Timed AltaRica. Timed AltaRica est ainsi un langage hiérarchique de description de systèmes temporisés de haut niveau. Enfin, dans la section 4.4, nous donnons la correspondance entre un modèle Timed AltaRica et un automate temporisé.

## 4.1 Une extension temporisée du langage AltaRica

Dans un premier temps, nous allons étendre la syntaxe du langage AltaRica donnée à la partie 2.2.1 page 28 en introduisant des horloges. Comme AltaRica manipule des variables, on a naturellement introduit des variables d'un nouveau type *clock*. Les variables horloges peuvent aussi bien appartenir aux variables d'états qu'aux variables de flux, ce qui permet de partager des horloges et de contraindre un système avec un délai global.

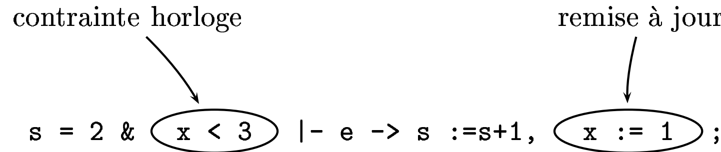
### 4.1.1 Extension syntaxique temporisée

L'ajout du temps dans la syntaxe touche l'ensemble des champs.

**Déclarations.** D'abord, on ajoute le mot clé *clock*, ce type peut être utilisé aussi bien pour les variables de flux que les variables d'état. On parle alors d'horloges de flux (respectivement d'état).

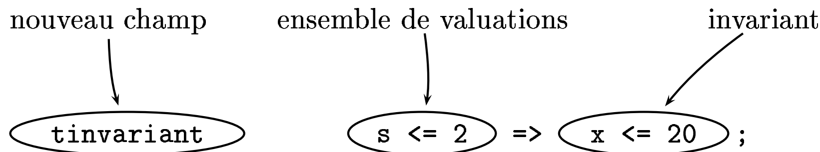
```
state s : [0,5];  x : clock;
```

**Transitions.** Les transitions ont été enrichies de façon à pouvoir tester les valeurs des horloges lors d'une transition et pouvoir remettre à jour certaines horloges. Ainsi, une transition est de la forme suivante :




**Contraintes.** On peut bien évidemment donner une valeur initiale à n'importe quelle horloge d'états.

Pour respecter les règles de cohérence du typage, on ne peut pas poser d'assertion comparant une variable discrète et une variable d'horloge. On a séparé les contraintes : aucune horloge n'apparaît dans l'assertion et on a introduit le champ *tinvariant* pour spécifier les contraintes touchant les horloges. Les seules contraintes tolérées seront des invariants temporels et seront de la forme suivante :




### Les composants Timed AltaRica

Le constituant élémentaire est le composant temporisé : il s'agit d'une extension temporisée du composant AltaRica. Nous présentons la syntaxe générale d'un composant temporisé au travers de la Figure 4.1. Les ajouts temporels au langage AltaRica sont indiqués à l'aide de caractères **gras** et les mots clés avec des caractères `types machine à écrire`. Les contraintes convexes d'horloges ont été définies dans les notations 3.1.1 page 41.

```
node Composant_temporise
 flow   f : type ...; y : clock ...
 event  e,h ...
 state  s : type ...; x : clock ...
 trans  formule(s,f) & contrainte(x,y) |- e -> s:= expression(s,f),
                              x:=expression(x,y);
       ...
 init   formule (s,f,x);
 assert formule(s,f);
 tinvariant formule(s,f) => contrainte_convexe(x,y);
edon
```

FIG. 4.1 – Syntaxe d'un composant Timed AltaRica

**Exemple 12 ([Un système de réservoir critique - I)** *On modélise un réservoir d'essence spécifié en Timed AltaRica dans la Figure 4.2 et dont la sémantique se trouve à la Figure 4.6 page 63. On suppose que la totalité de la contenance du réservoir est vidée en* 100 *u.t. L'horloge $x$ est utilisée pour représenter la quantité du carburant dans le réservoir. Ce réservoir peut se trouver dans* 3 *états :*

- *le réservoir est plein et au repos* $s = 0$,
- *le réservoir est sollicité et fournit du carburant à un autre système (par exemple un avion)* $s = 1$,
- *le réservoir est vide* $s = 2$.

```
node Reservoir
 flow R: bool; Y:clock;
 event  marche , arret, fuite;
 state  s :[0,2]; x: clock;
 trans s=0 & R=vrai |- marche -> s:=1, x:=0;
    s=1 & x=100|- arret -> s:=2;
    s!=2 |- fuite -> s:=2, x:=101;
 tinvariant (s=1) => (x <= 100); x=Y;
edon
```

FIG. 4.2 – Un réservoir en Timed AltaRica

*Un fonctionnement sans panne est le suivant : quand* $s = 0$, *le réservoir est inutilisé et* $x$ *croît sans limite puisqu'aucun invariant n'est spécifié. Ensuite il peut se mettre en* marche, *alors* $x$ *est remise à* 0 *et croît jusqu'à* 100 *auquel cas le réservoir est vide. Par respect pour l'invariant temporel, la transition* arret *est tirée et le composant se met dans la configuration* $s = 2$.

*Seulement, il se peut qu'une fuite survienne et donc de n'importe quel état* $s\ != 2\ \big(\equiv (s = 0) \vee (s = 1)\big)$ *le réservoir peut à tout moment se vider. En cas de fuite, le réservoir est considéré comme vide car on ne sait pas à quelle vitesse le carburant disparaît. Si la transition* fuite *est tirée, on remet l'horloge à* 101 *ainsi le réservoir est considéré vide.*

*La variable* $x$ *est exportée grâce à la variable de flux* $Y$ *contrainte par l'invariant temporel* $Y = x$, *ainsi les autres composants connaissent la quantité de carburant restante. On remarque que le champ* assert *n'apparaît pas car il n'est pas nécessaire ici.*

#### La grammaire de Timed AltaRica

Le grammaire concrète est formellement définie en Annexe C page 143. Pour cela, nous avons repris la grammaire concrète d'AltaRica définie dans [Vin03a] avec les attributs de visibilité et nous avons ajouté les extensions syntaxiques présentées ici.

### 4.1.2 Systèmes de transitions temporisés interfacés

On temporise dans un premier temps les STI donnés à la Définition 2 page 25, ce qui nous permettra d'exprimer la sémantique des composants temporisés (que l'on peut également voir comme une extension temporisée des automates à contraintes).

**Définition 15 (Système de transitions temporisé interfacé [CPR02])** *Un* système de transitions temporisé interfacé *(STTI) de dimension* $(n, m)$ *sur le domaine temporel* $\mathbb{T}$ *est un quintuplet* $\mathcal{A} = \langle E_t, F_t, S_t, \pi, T \rangle$ *avec :*

1. $E_t = E_+ \cup \{\epsilon\} \cup \mathbb{T}$ *où* $E_+$ *est un ensemble fini d'événements tel que* $\epsilon \notin E_+$ *;*
2. $F_t = F \times \mathbb{R}^m$ *est un ensemble de valeurs de flux, avec* $F$ *ensemble de valeurs discrètes de flux et* $\mathbb{R}^m$ *ensemble de valeurs continues;* $S_t \subseteq S \times \mathbb{R}^n$ *ensemble d'états, avec* $S$ *ensemble d'états discrets et* $\mathbb{R}^n$ *ensemble d'états continus;*
3. $\pi : S_t \to 2^{F_t}$ *est une application qui associe à chaque état* $q \in S_t$ *toutes les valeurs de flux admissibles en* $q$. *On suppose que* $\forall q \in S_t,\ \pi(q) \neq \emptyset$.

*4. $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ est la relation de transition, elle vérifie :*

*(a) $(q, g, e, q', g') \in T \Rightarrow g \in \pi(q) \wedge g' \in \pi(q')$*

*(b) $\forall q \in S_t, \big(g, g' \in \pi(q) \iff (q, g, \epsilon, q, g') \in T\big)$*

*(c) $\forall q \in S_t, \big(g \in \pi(q) \iff (q, g, 0, q, g) \in T\big)$*

*Une configuration d'un STTI est un couple $(q, g) \in S_t \times F_t$ avec $g \in \pi(q)$.*

**Remarque 2** *On remarque qu'il y a deux différences majeures entre un STI et un STTI :*

1. *l'ensemble des événements est fini dans les STI et infini de la forme $E \cup \mathbb{T}$ dans les STTI;*
2. *la relation de transitions est explicite. Cela est nécessaire pour assurer que les valeurs de flux ne sont pas modifiées pendant une transition étiquetée 0 (item 4.c). Cela nous permettra également d'exprimer que les valeurs de variables discrètes de flux restent constantes pendant des transitions temporelles dans les composants temporisés (cf. Définition 19 page 61).*

*La transition $\epsilon$ se fait en un temps nul et elle peut avoir lieu à tout instant. Alors en un temps nul, il peut y avoir un enchaînement de transitions $\epsilon$. En revanche, pendant une transition étiquetée 0, aucune transition discrète n'a lieu.*

### Les valeurs continues de flux

Les valeurs de flux dans un STTI, de même que les valeurs de flux d'un STI, peuvent être modifiées à tout instant par une transition $\varepsilon$ : ce comportement est exprimé par la condition $\forall s \in S_t,\ \forall f, f' \in \pi(s),\ (s, f, \varepsilon, s, f') \in T$.

De plus, lorsqu'une transition étiquetée $e$ est tirable depuis une configuration $(s, f)$, i.e. $(s, f, e, s', f') \in T$, alors toutes les transitions $(s, f, e, s', f'') \in T$ si $f'' \in \pi(s')$. Ces propriétés s'appliquent pour les valeurs réelles de flux.

On peut justifier ce choix par l'exemple suivant : on considère le produit libre de deux composants temporisés partageant une même variable de flux $Y$. On anticipe la syntaxe des nœuds temporisés donnés à la Définition 27 page 81 :

```
node T
state y : clock;
flow Y : clock;
event e
true |-e-> y:=0;
tinvariant y=Y;
edon
```

```
node B
state s :[0,1];
flow Y : clock;
event f
 s=0 & Y <= 10 |-f-> s:=1;
edon
```

```
node main
sub t:T; b:B;
tinvariant t.Y=b.Y;
edon
```

$Y$ appartient au composant $T$ et est lue par le composant $B$. Ainsi dans le nœud *main*, $Y$ est possiblement remise à jour à n'importe quel moment quand la transition `<e,epsilon>` est tirée, c'est-à-dire dans B la transition $\epsilon$. Donc à tout moment $Y$ peut être remise à jour par $\epsilon$ dans B. On constate que si on ajoute une synchronisation `<e,f>` alors $Y$ doit être remise à jour sur $f$ dans $B$.

**Exemple 13** *On considère le STTI $\mathcal{S} = \langle E_t, F_t, S_t, \pi, T\rangle$ avec $E_+ = \{f\}$, $S = \{0, 1\}$, $F = \mathbb{R}$, $\pi = \mathbb{R}$ et $T = \{(0, y, f, 1, y')\}$ avec $y \leq 10$[1].*

*On représente l'indéterminisme du STI dans les Figures 4.3 et 4.4 : on énumère les transitions possibles depuis une configuration $(0, y)$ avec $y \in \mathbb{R}$. On a également dessiné une transition 0 en pointillés.*

[1]Il s'agit de la sémantique du composant $B$ défini ci-dessus (cf Définition 19 page 61).

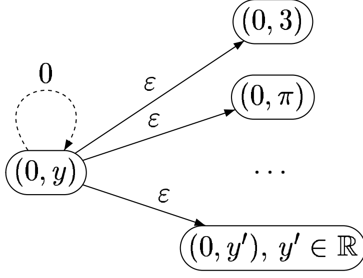



FIG. 4.3 – $(0, y, \varepsilon, 0, y') \in T$

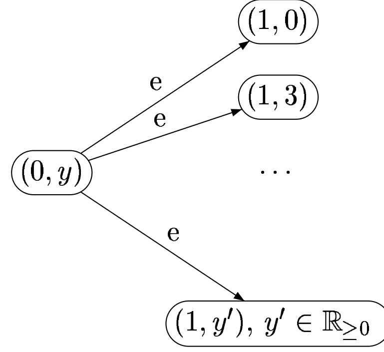



FIG. 4.4 – $(0, y, e, 1, y') \in T$ avec $y \leq 10$

### 4.1.3 La bisimulation temporisée

On étend la relation de bisimulation interfacée pour les STTI. Cela nous servira entre autre à prouver la modularité du langage Timed AltaRica.

**Définition 16 (Bisimulation temporisée interfacée [CPR02])** *Soient $\mathcal{A}_1 = \langle E_t, F_t, S_{t,1}, \pi_1, T_1\rangle$ et $\mathcal{A}_2 = \langle E_t, F_t, S_{t,2}, \pi_2, T_2\rangle$ deux STTI. Une relation de bisimulation temporisée interfacée pour $\mathcal{A}_1$ et $\mathcal{A}_2$ est une relation $R \subseteq S_{t,1} \times S_{t,2}$ donnée par :*

1. $\forall q_1 \in S_{t,1}, \exists q_2 \in S_{t,2}, (q_1, q_2) \in R$ *et* $\forall q_2 \in S_{t,2}, \exists q_1 \in S_{t,1}, (q_1, q_2) \in R$
2. $\forall (q_1, q_2) \in R,\ \pi_1(q_1) = \pi_2(q_2)$
3. $\forall (q_1, g, e, q_1', g_1') \in T_1,\ \forall q_2 \in S_{t,2}$ *tel que* $(q_1, q_2) \in R$ *alors* $\exists (q_2, g, e, q_2', g_2') \in T_2,\ (q_1', q_2') \in R$
4. $\forall (q_2, g, e, q_2', g_2') \in T_2,\ \forall q_1 \in S_{t,1}$ *tel que* $(q_1, q_2) \in R$ *alors* $\exists (q_1, g, e, q_1', g_1') \in T_1,\ (q_1', q_2') \in R.$

*Deux STTI sont bisimilaires temporellement si et seulement si il existe une relation de bisimulation temporisée interfacée entre eux.*

On étend la notion d'homomorphisme de bisimulation [Poi00] au cas temporisé. C'est une approche équivalente à la relation de bisimulation pour comparer les systèmes.

**Définition 17 (Homomorphisme de bisimulation temporisée interfacée [CPR02])** *Soient $\mathcal{A}_1 = \langle E_t, F_t, S_{t,1}, \pi_1, T_1\rangle$ et $\mathcal{A}_2 = \langle E_t, F_t, S_{t,2}, \pi_2, T_2\rangle$ deux STTI. Un* homomorphisme de bisimulation temporisée interfacée $h : \mathcal{A}_1 \to \mathcal{A}_2$ *est une application $h : S_{t,1} \to S_{t,2}$ telle que :*

1. *$h$ est surjective,*
2. $\forall q_1 \in S_{t,1},\ \pi_1(q_1) = \pi_2(h(q_1)),$
3. $\forall (q_1, g, e, q_1', g_1') \in T_1,$ *alors* $\exists g_2',\ (h(q_1), g, e, h(q_1'), g_2') \in T_2,$
4. $\forall q_1 \in S_{t,1}, \forall q_2' \in S_{t,2},\ (h(q_1), g, e, q_2', g_2') \in T_2 \Rightarrow \exists q_1' \in S_{t,1},\ \exists g_1',\ h(q_1') = q_2' \wedge (q_1, g, e, q_1', g_1') \in T_1,$

L'item 3 est équivalent à $\forall (q_1, g, e, q_1', g_1') \in T_1,\ (h(q_1), g, e, h(q_1'), g_1') \in T_2$ et est aussi équivalent à $\forall (q_1, g, e, q_1', g_1') \in T_1$, alors $\forall g_2',\ (h(q_1), g, e, h(q_1'), g_2') \in T_2$. Cela vient du fait que les $\pi_i$ sont égaux et de la définition des STTI.

**Théorème 3** *Deux STTI $\mathcal{A}_1$ et $\mathcal{A}_2$ sont bisimilaires temporellement si, et seulement si il existe un STTI $\mathcal{B}$ et deux homomorphismes de bisimulation temporisée interfacée $h_1 : \mathcal{A}_1 \to \mathcal{B}$ et $h_2 : \mathcal{A}_2 \to \mathcal{B}$.*

**Preuve du théorème 3**. *On suppose qu'il existe deux homomorphismes $h_1$ et $h_2$ et on note $\mathcal{B} = \langle E_t, F_t, S_t, \pi, T\rangle$. On considère la relation $R \subseteq S_{t,1} \times S_{t,2}$ définie par*

$$R(s,s') \iff h_1(s) = h_2(s')$$

*(i.e. $R = \bigcup_{s\in S_{t,1}} \cup_{s'\in h_2{}^{-1}(h_1(s))}(s,s')$ avec $h^{-1}(q) = \{s \in S_{t,2} | h_2(s) = q\}$).*

*On montre que $R$ est une relation de bisimulation interfacée temporisée en assurant les 4 points de la définition.*

1. *Soient $q_1 \in S_{t,1}$ et $s = h_1(q_1) \in S_t$, comme $h_2$ est surjective, on est assuré qu'il existe $q_2 \in h_2^{-1}(s)$. Si $q_2 \in S_{t,2}$, comme $h_1$ est surjective, on sait qu'il existe $q_1 \in S_{t,1}$ tel que $h_1(q_1) = h(q_2)$, alors $(q_1,q_2) \in R$. On a le premier point de la définition d'une bisimulation interfacée.*
2. *Ensuite, $\forall q_1 \in S_{t,1}, \pi_1(q_1) = \pi(h_1(q_1)) = \pi(h_2(q_2)) = \pi_2(q_2)$ avec $(q_1,q_2) \in R$.*
3. *Soit $(q_1,g,e,q_1',g_1') \in T_1$ alors $(h_1(q_1),g,e,h_1(q_1'),g_1') \in T$ et $\exists q_2 \in S_{t,2}$ tel que $h_1(q_1) = h_2(q_2)$. Alors, d'après les propriétés des homomorphismes $\exists q_2' \in S_{t,2}$ tel que $h_2(q_2') = h_1(q_1')$ et $(q_2,g,e,q_2',g_1') \in T_2$.*
4. *Enfin, soit $(q_2,g,e,q_2',g_2') \in T_2$, on a $(h_2(q_2),g,e,h_2(q_2'),g_2') \in T$. Donc par surjectivité, $\exists q_1 \in S_{t,1}$ tel que $h_1(q_1) = h_2(q_2)$, i.e. $(q_1,q_2) \in R$. On applique encore le point 4 des homomorphismes, alors $\exists q_1', (q_1,g,e,q_1',g_2') \in T_1$ avec $h_1(q_1') = h_2(q_2')$.*

*La réciproque est plus compliquée puisqu'il faut construire $\mathcal{B}$. L'idée est schématisée dans le diagramme 4.5 : on va quotienter chaque STTI par une relation d'équivalence obtenue à partir de $R$ et les homorphismes seront les homomorphismes surjectifs d'un système de transitions dans son quotient. Le dernier point est d'assurer que les deux STTI obtenus sont isomorphes ($\phi$ sera l'isomorphisme).*

*On considère la relation $(R \circ R^{-1})^\star$ sur $S_{t,1}$. On rappelle que $R^{-1} = \{(q,q') | (q',q) \in R\}$, $R \circ S = \{(q,q') | \exists q_1,\ (q,q_1) \in R \wedge (q_1,q') \in S\}$ et que $R^\star = \bigcup_{n\in\mathbb{N}} R^n$ où $R^n$ est la relation $R$ itérée $n$ fois. Ainsi, $(q,q') \in (R\circ R^{-1})^\star \Leftrightarrow \exists q_1,\cdots,q_n \in S_{t,1}, \exists q_1',\cdots,q_n' \in S_{t,2}, (q,q_1') \in R \wedge (q_1',q_1) \in R^{-1} \wedge (q_1,q_2') \in R \cdots (q_n,q_n') \in R \wedge (q_n',q') \in R^{-1}$. C'est une relation d'équivalence puisque $R \circ R^{-1}$ est symétrique et $R^\star$ est la clotûre réflexive et transitive de $R$.*

*On peut alors quotienter le STTI $\mathcal{A}_1$ par cette relation d'équivalence, on obtient le STTI $\mathcal{A}_1/(R\circ R^{-1})^\star$ dont les états sont les classes d'équivalence. Soit $h$ l'homomorphisme surjectif $\mathcal{A}_1 \to \mathcal{A}_1/(R\circ R^{-1})^\star$, $h$ associe à un état sa classe d'équivalence. De même, on considère la relation d'équivalence $(R^{-1}\circ R)^\star$ sur $\mathcal{A}_2$ et l'homomorphisme surjectif $h_2 : \mathcal{A}_2 \to \mathcal{A}_2/(R^{-1}\circ R)^\star$.*

*On montre enfin que $\mathcal{A}_1/(R\circ R^{-1})^\star$ et $\mathcal{A}_2/(R^{-1}\circ R)^\star$ sont isomorphes. On considère alors l'application*

$$\begin{array}{llll} \phi: & \mathcal{A}_1/(R\circ R^{-1})^\star & \to & \mathcal{A}_2/(R^{-1}\circ R)^\star \\ & [q] & \mapsto & [q'] \text{ tel que } (q,q') \in R \end{array}$$

*$\phi$ est bien une application puisque si $q_1 \in [q]$ et $(q_1,q_1') \in R$ alors $[q'] = [q_1']$. En effet, $q_1 \in [q]$ entraîne que $\exists \tilde{q}_2,\cdots,\tilde{q}_n \in S_{t,1}, \exists q_2',\cdots,q_{n-1}' \in S_{t,2}, (q_1,q_1') \in R \wedge$*

$$\underbrace{(\tilde{q}_2,q_1') \in R \wedge (q_1,q_2') \in R \cdots (\tilde{q}_n,q) \in R.}_{\wedge (q,q')\in R \implies q_1' \in [q']}$$

*Cela entraîne également l'injectivité et la surjectivité.*

*On pose $\mathcal{B} = \mathcal{A}_2/(R^{-1}\circ R)^\star$. Alors, $h_1 = \phi \circ h : \mathcal{A}_1 \to \mathcal{B}$ est un homomorphisme surjective comme composée d'un isomorphisme et un homomorphisme surjectif.*

□

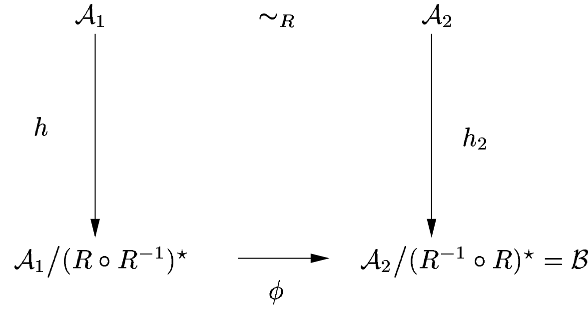



FIG. 4.5 – Diagramme bisimulation et homomorphisme de bisimulation

### 4.1.4 Sémantique des composants temporisés

On peut maintenant exprimer la sémantique d'un composant temporisé Timed AltaRica. De même que dans AltaRica, on définit d'abord proprement la syntaxe : un composant correspond à un automate à contraintes temporisé, puis on en donne la sémantique sous forme de STTI.

**Définition 18 (Composant temporisé - syntaxe abstraite [CPR02])** *Un composant temporisé est un septuplet $\mathcal{A} = \langle V_S, C_S, V_F, C_F, E, A, M\rangle$ où :*

1. *$V_S, V_F$ sont des ensembles disjoints appelés respectivement ensembles des variables d'états et de flux. $C_S, C_F$ sont des ensembles disjoints appelés respectivement ensembles d'*horloges d'état *et de* flux réels. *On note $Z = V_S \cup V_F$ et $C = C_S \cup C_F$. On suppose que $Z \cap C = \emptyset$ ;*
2. *$E = E_+ \cup \{\epsilon\}$ est un ensemble d'événements et $\epsilon \notin E_+$ est l'action inobservable ;*
3. *l'*assertion *du composant $A$ est une formule telle que : $A = A_Z \wedge A_C \in \mathcal{F}(\mathcal{L}_{Z\cup C})$ avec $A_Z \in \mathcal{F}(\mathcal{L}_Z)$ et $A_C = \bigwedge_{k\in K} P_k \implies I_k$ où $K$ est un ensemble d'indices fini, $P \in \mathcal{F}(\mathcal{L}_Z)$ et $I \in \mathcal{C}(C)$ définit une région convexe de $\mathbb{R}^p$ si $|C| = p$.*
4. *$M \subseteq (\mathcal{F}(\mathcal{L}_Z) \times \mathcal{C}(C)) \times E \times (\mathcal{T}(\mathcal{L}_Z)^{V_S} \times \mathcal{U}(C_S))$ est un ensemble de* macro-transitions *$((g,\gamma), e, (a, R))$ telles que :*
   - *(a) $(g,\gamma)$ est une* garde *où $g \in \mathcal{F}(\mathcal{L}_Z)$ est une formule sur les variables discrètes et $\gamma \in \mathcal{C}(C)$ est une contrainte d'horloges ;*
   - *(b) $e \in E$ est l'événement déclenchant la transition ;*
   - *(c) $a : V_S \to \mathcal{T}(Z)$ est une affectation pour les variables de $V_S$ et $R \in \mathcal{U}(C_S)$ est l'ensemble des horloges remises à jour par la transition.*

L'item 3 de la Définition précédente nous permet de spécifier des contraintes entre les variables d'horloges et les variables de flux réelles. On peut ainsi exporter une variable d'horloge d'état $x$ dans une variable de flux $Y$ en écrivant $Y = x$ puisque cette formule est équivalent à $tt \implies Y = x$.

La sémantique d'un composant temporisé est exprimée par un STTI et associe des valeurs aux variables, dans $\mathcal{D}$ pour les discrètes et $\mathbb{R}$ pour les horloges. L'assertion $A$ est transformée en la fonction $\pi$ des valeurs admissibles. Enfin la relation de transitions est gardée ce qui entraîne que les valeurs des variables permettant une transition sont celles vérifiant la formule $g \wedge \gamma$ et dont l'affectation est admissible.

**Définition 19 (Sémantique d'un composant temporisé [CPR02])** *Soit $\mathcal{A} = \langle V_S, C_S, V_F, C_F, E, A, M\rangle$ un composant temporisé avec $|C_S| = n$ et $|C_F| = m$. La sémantique de $\mathcal{A}$ sur le domaine temporel $\mathbb{T}$ est le système de transitions temporisé interfacé $[\![\mathcal{A}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$ de dimension $(n, m)$ construit de la manière suivante :*

1. $E_t = E \cup \mathbb{T}$ ;
2. $F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$ ;
3. $S_t = \{q \in \mathcal{D}^{V_S} \times \mathbb{R}^n | \exists g \in F_t, (q,g) \in [\![A]\!]\}$ ;
4. $\pi : S_t \to 2^{F_t}$ *tel que* $\pi(q) = \{g \in F_t | \ (q,g) \in [\![A]\!]\}$ ;
5. $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ *est telle que* $T = [\![M]\!]$ *avec :*
   (a) *soit la transition* $t = ((g,\gamma), e, (a,R))$, *on définit* $[\![t]\!]$ *par :*

$$((s,\nu),(f,\mu),e,(s',\nu'),(f',\mu')) \in [\![t]\!] \text{ si } \begin{cases} ((s,\nu),(f,\mu)) \in [\![A \wedge g \wedge \gamma]\!] \wedge s' = a(s,f) \\ \wedge \quad \nu' = R(\nu,\mu) \\ \wedge \quad (f',\mu') \in \pi(s',\nu') \end{cases}$$

   (b) *soit* $\delta \in \mathbb{T}$, *on définit* $[\![\delta]\!]$ *par :*

$$((s,\nu),(f,\mu),\delta,(s,\nu'),(f,\mu')) \in [\![\delta]\!] \text{ si } \begin{cases} ((s,\nu),(f,\mu)) \in [\![A]\!] \wedge \nu' = \nu + \delta \\ \wedge \quad ((s,\nu'),(f,\mu')) \in [\![A]\!] \\ \wedge \quad \forall \delta' \leq \delta, \exists \mu_{\delta'}, (\nu + \delta', \mu_{\delta'}) \in [\![I(s,f)]\!] \end{cases}$$

   *avec* $I(s,f) = \bigwedge_{k \in K, (s,f) \in P_k} I_k$.

   (c) $[\![M]\!] = \cup_{t \in M} [\![t]\!] \bigcup \cup_{\delta \in \mathbb{T}} [\![\delta]\!]$.

**Remarque 3** *Les automates à contraintes temporisés ou composants temporisés sont une extension syntaxique des automates à contraintes. On distingue dans un automate à contraintes temporisé des variables particulières à valeurs réelles, dont le comportement suit des règles établies dans la syntaxe abstraite : elles sont remises à jour, elles évoluent de manière continue quand elles sont des horloges d'état et n'apparaissent pas dans n'importe quel type de formule dans l'assertion.*

*Un deuxième remarque concerne les pas continus dans un composant temporisé : on peut laisser passer un temps* $\delta$ *dans une configuration* $((s,\nu),(f,\mu))$, *c'est-à-dire* $\big((s,\nu),(f,\mu),\delta,(s,\nu+\delta),(f,\mu')\big) \in T$, *si, et seulement si on peut atteindre* $(s,\nu+\delta)$ *en laissant passer* $\delta$ *u.t. depuis* $(s,\nu)$ *sans exécuter de transition discrète, même* $\varepsilon$, *et si on assure qu'on peut toujours trouver une valeur de flux réel en chaque point de* $[0,\delta]$.

*On n'a fait aucune hypothèse sur les flux réels et à partir de maintenant, nous supposons que les* **seuls flux réels admissibles sont des horloges**. *On se place ainsi dans la famille des automates temporisés avec remise à jour. Alors l'item 5.c est entièrement défini par :*

$$((s,\nu),(f,\mu),\delta,(s,\nu'),(f,\mu')) \in [\![\delta]\!] \text{ si } \begin{cases} ((s,\nu),(f,\mu)) \in [\![A]\!] \wedge \nu' = \nu + \delta \\ \wedge \quad \mu' = \mu + \delta \wedge ((s,\nu'),(f,\mu')) \in [\![A]\!] \\ \wedge \quad \forall \delta' \leq \delta, (\nu + \delta', \mu + \delta') \in [\![I(s,f)]\!] \end{cases}$$

*avec* $I(s,f) = \bigwedge_{k \in K, (s,f) \in P_k} I_k$.

**Exemple 14 (Un système de réservoir critique - II)** *On exprime la sémantique du composant temporisé Reservoir donné dans l'Exemple 12 page 56. On obtient l'automate temporisé donné à la Figure 4.6.*

*Un état de contrôle est naturellement une valuation des variables discrètes : s variable d'état* $\in [0,2]$ *et R variable de flux booléenne* $\in [0,1]$. *On a représenté ici les* $\epsilon$*-transitions et on constate que les valeurs de flux externes peuvent se modifier à n'importe quel moment par une* $\epsilon$*-transition. A chaque transition discrète, les variables de flux sont réévaluées ce qui explique que toutes les transitions discrètes sont dupliquées selon R est vrai ou faux.*

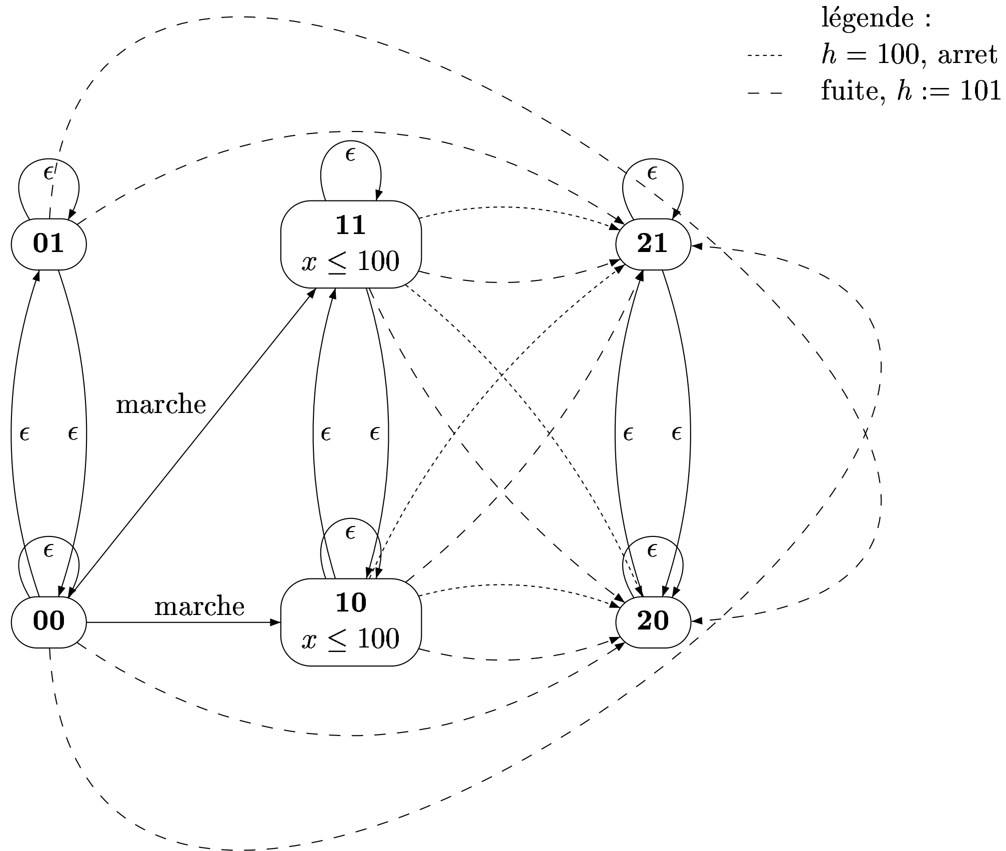



FIG. 4.6 – Représentation du STTI $[\![Reservoir]\!]$

Dans cette première section, nous avons défini les composants temporisés : ces composants temporisés sont les éléments de base de Timed AltaRica. Nous étendons dans la prochaine section la notion de priorité à des priorités quantifiant le temps.

## 4.2 Les priorités temporelles

Une priorité dans AltaRica, décrite dans la partie 2.1.7 page 26, empêche certains événements de se produire depuis une configuration. Nous introduisons deux priorités particulières dans Timed AltaRica : la première, l'*urgence* [SY96, BST98, BS00] empêche le temps de s'écouler dans certaines configurations et la deuxième, les *priorités temporelles* [BST98, BS00, BGS00] interdisent certains événements dans des configurations, seulement ces interdictions dépendent uniquement des valuations des horloges, c'est-à-dire qu'en attendant un certain laps de temps, il se peut que les interdictions soient levées.

L'intérêt de ces deux nouveaux opérateurs est leur simplicité et leur concision pour exprimer des comportements complexes. L'urgence représente des situations courantes dans des systèmes critiques, si le système entre dans un état critique, il doit agir immédiatement. Ce type de comportement revient à agir sur les invariants temporels.

Les priorités temporelles peuvent être utilisées dans la synthèse de contrôleur [AGP+99] pour l'ordonnancement temps réel [Alt01, AGS02]. Ce type de comportements revient à agir sur les

gardes des transitions.

Avant de définir ces priorités dans Timed AltaRica, nous introduisons un opérateur particulier, l'opérateur de *faible précondition* qui modifie syntaxiquement les gardes des transitions de façon à ce que les états atteignables par les transitions sans prendre en compte l'assertion soient admissibles.

### 4.2.1 La faible précondition

On considère un composant temporisé $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$. Pour les manipulations syntaxiques qui vont suivre, nous introduisons un opérateur symbolique qui va assurer que pour toutes les transitions $((g, \gamma), e, (a, R))$ spécifiées, chaque origine de la transition, c'est-à-dire chaque valuation dans $[\![g]\!]$, atteint au moins une configuration admissible. Pour cela, nous modifions syntaxiquement les gardes des transitions :

**Définition 20 (Opérateur de faible précondition [CPR02])** *Soit* $Q \subseteq S_t \times F_t$ *et une transition* $t = ((g, \gamma), e, (a, R)) \in M$*, alors la faible précondition de* $Q$ *par rapport à* $t$ *est le sous ensemble* $Pre_t(Q) = \{(s, f) \in S_t \times F_t | \exists f' \in F_t, ((a, R)(s, f), f') \in Q\}$.

**Exemple 15** *On considère le composant temporisé suivant :*

```
node Ex_precondition
 flow R: [0,1];
 event  e
 state  s :[0,1]; x: clock;
 trans s=0 & x<=20 |- e -> s:=1;
 assert s=1 => R=1;
 tinvariant R=1 => (x <= 10);
edon
```

*Il n'y a donc qu'une macro-transition* $t$*. L'interprétation de l'assertion avec l'invariant temporel est* $[\![A]\!] = \{0\}\times\mathbb{R}\times\{0\}\cup\{0\}\times[0,10]\times\{1\}\cup\{1\}\times[0,10]\times\{1\}$*. Nous avons deux cas de figures : soit une configuration est dans la faible précondition de la transition, Figure 4.7 avec* $((0,5),0) \in Pre_t([\![A]\!])$*, soit elle ne l'est pas, Figure 4.8 avec* $((0,15),0) \notin Pre_t([\![A]\!])$.

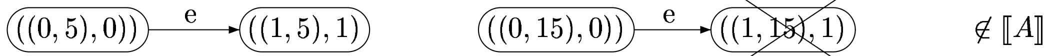



FIG. 4.7 – $((0,5),0) \in Pre_t([\![A]\!])$

FIG. 4.8 – $((1,15),0) \notin Pre_t([\![A]\!])$

On réécrit syntaxiquement les gardes des transitions de sorte que pour toute configuration $(s, \nu, f, \mu) \in [\![g \wedge \gamma]\!]$, il existe une configuration admissible atteignable, c'est-à-dire $\exists f', \mu'$, $((a, R)(s, \nu, f, \mu), f', \mu') \in [\![A]\!]$. Si au contraire aucune configuration admissible n'est atteignable par $t$ depuis $(s, \nu, f, \mu)$, on élimine $(s, \nu, f, \mu)$ de la garde en la modifiant en $g' = g \wedge \neg(s, f)$ et $\gamma' = \gamma \wedge \neg(\nu, \mu)$.

La procédure consiste à trouver pour toute transition $t = ((g, \gamma), e, (a, R)) \in M$ la nouvelle garde $g' \wedge \gamma'$ de sorte que $[\![g]\!] \cap [\![\gamma]\!] \cap Pre_t([\![A]\!]) = [\![g']\!] \cap [\![\gamma']\!]$. Pour cela, on suit les étapes suivantes :

1. Pour toute transition $t$, on écrit $Pre_t([\![A]\!])$ en une formule $\phi_t \wedge \theta_t$ avec $\phi_t$ est tel que $vlib(\phi_t) \subseteq V_S \cup V_F$ et $\theta_t$ est tel que $vlib(\theta_t) \subseteq C_S \cup C_F$.
   On a $g' \wedge \gamma' = g \wedge \gamma \wedge \big(\exists (s, f),\ A_Z \wedge A_C \wedge a(s, f)\big)$. Si les domaines discrets sont finis, pour éliminer les quantificateurs $\exists$, il suffit d'énumérer l'ensemble des valuations qui satisfont la condition de faible précondition et l'exprimer par une formule.

2. Alors, $g' = g \wedge \phi_t$ et $\gamma' = \gamma \wedge \theta_t$. On transforme syntaxiquement la transition : $t = ((g,\gamma), e, (a,R)) \in M$ en $t = ((g',\gamma'), e, (a,R))$.

**Exemple 16** *On reprend le composant temporisé décrit dans l'exemple 15 et on applique la procédure de précondition. On a* $Pre_t([\![A]\!]) = \{(s,x,R) \in [\![A]\!] \mid \exists R', ((1,x),R') \in [\![A]\!]\} = \{(s,x,R) \mid ((1,x),1) \wedge x \leq 10\} = \{0\} \times [0,10] \times \{0\} \cup \{0\} \times [0,10] \times \{1\} \cup \{1\} \times [0,10] \times \{1\}$ *et* $[\![g]\!] \cap [\![\gamma]\!] = \{0\} \times [0,20] \times \{0\} \cup \{0\} \times [0,20] \times \{1\}$. *D'où,* $g' \equiv g$ *et* $\gamma' \equiv x \leq 10$.

**Lemme 1** *Soit* $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ *un composant temporisé et* $\mathcal{A}' = (V_S, C_S, V_F, C_F, E, A, M')$ *le composant temporisé obtenu avec la procédure de faible précondition, alors* $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!]$.

**Preuve du lemme 1**. *On note* $[\![\mathcal{A}]\!] = (S_t, F_t, E_t, \pi, T)$ *et* $[\![\mathcal{A}']\!] = (S_t, F_t, E_t, \pi, T')$. *Puisqu'on a réduit les gardes, toute transition de* $[\![\mathcal{A}']\!]$ *est dans* $[\![\mathcal{A}]\!]$. *Réciproquement, si* $(q, g, e, q', g') \in T$ *on en déduit qu'il existe une transition* $t = ((g,\gamma), e, (a,R))$ *telle que* $(q, g, e, q', g') \in [\![t]\!]$. *Donc on a* $(q,g) \in Pre_t([\![A]\!])$, *i.e.* $(q, g, e, q') \in T'$. □

Dans toute la suite de cette partie, nous supposerons que les gardes des transitions ont été modifiées par la procédure de faible précondition. Nous présentons dans la prochaine section l'*urgence* [SY96, BST98, BS00].

### 4.2.2 L'urgence

Un comportement potentiellement attendu d'un système est celui d'une réaction urgente dans un certain contexte. Si un système critique est à un point de rupture, il faut agir instantanément afin d'améliorer la situation. Un événement $e \in E_+$ est *urgent* [SY96, BST98, BS00] si toutes les transitions étiquetées par $e$ sont *eager*[2], i.e. dans tout état permettant de tirer une transition eager, le système doit immédiatement faire une transition discrète car le temps est bloqué. Soit $E$ un ensemble d'événements, on note $\mathcal{U}_g(E)$ le sous-ensemble de $E_+ = E \setminus \{\epsilon\}$ des événements urgents.

**Exemple 17** *On reprend l'exemple 3.2 page 43 et on calcule les langages temporisés en ajoutant des événements urgents.*

| *événements urgents* | *langages* |
|---|---|
| $\mathcal{U}_g = \{a\}$ | $(< a^\omega b >_0 c)^\omega$ |
| $\mathcal{U}_g = \{b\}$ | $(< a^\omega b >_0 c)^\omega$ |
| $\mathcal{U}_g = \{a,b\}$ | $(< a^\omega b >_0 c)^\omega$ |
| $\mathcal{U}_g = \{c\}$ | $(<< a^\omega >_{[0,1[} b >_{[0,2]} < c >_0)^\omega$ |
| $\mathcal{U}_g = \{a,b,c\}$ | $(< a^\omega b >_0 < c >_0)^\omega$ |

#### L'urgence dans Timed AltaRica

Dans Timed AltaRica, nous avons choisi d'implanter une urgence indéterministe en ce sens que dans une configuration où une transition étiquetée par une action urgente est possible, le système doit quitter d'une manière ou d'une autre la configuration, que ce soit en exécutant la transition ou une autre tirable. Si $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ est un composant temporisé dont la sémantique contient le comportement de la Figure 4.9, alors si on ajoute la relation d'urgence, on représente intuitivement le blocage du temps dans la Figure 4.10.

[2] cf. partie 3.1.3 page 44

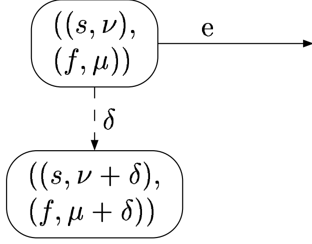



FIG. 4.9 – $[\![\mathcal{A}]\!]$ sans urgence

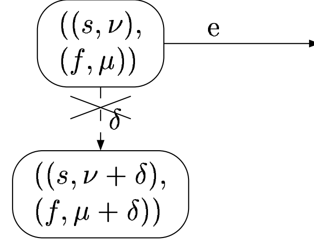



FIG. 4.10 – $[\![\mathcal{A}]\!]$ avec urgence $e > time$

La syntaxe de l'opérateur d'urgence est extrêmement simple, un événement $e$ urgent est noté $e > time$. De même que dans la syntaxe AltaRica des priorités, on peut noter l'ensemble $\mathcal{U}_g(E) = \{e_1, \cdots, e_n\}$ d'événements urgents :

```
event {e1,..,en} > time
```

**Exemple 18** *On considère le composant temporisé suivant :*

```
node Ex_urgence_sans_flux
state s : [0,1]; x :clock;
event e > time, f,g
trans s=1 & 10 <= x <=20 |- e -> s:=0;
   |- f -> s:=0;
   |- g -> s:=1;
edon
```

### Encodage syntaxique de l'urgence

L'urgence ne modifie pas l'expressivité du modèle et peut être encodée à l'aide d'invariants temporels. Etant donnés un composant temporisé $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ et $\mathcal{U}_g(E)$ un ensemble d'événements urgents, on construit un composant temporisé $\mathcal{A} \restriction_{\mathcal{U}_g(E)} = (V'_S, C'_S, V'_F, C'_F, E, A', M')$ bisimilaire temporellement sans événements urgents.

Pour pouvoir exprimer l'encodage syntaxique d'une relation d'urgence dans Timed AltaRica, nous avons besoin d'un opérateur de fermeture sur les formules. Cet opérateur permet notamment d'exprimer des limites temporelles à ne pas dépasser.

**Définition 21 (Fermeture de prédicat [BGS00])** *Etant donnée une contrainte $\gamma$ sur un ensemble d'horloges $X$, on définit le* front montant*, notée $\gamma\uparrow$, de $\gamma$ pour une valuation $v$ par :*

$$\gamma\uparrow(v) = \big(\gamma(v) \wedge \exists\epsilon > 0, \forall 0 < \epsilon' \leq \epsilon,\ \neg\gamma(v - \epsilon')\big) \vee \big(\neg\gamma(v) \wedge \exists\epsilon > 0, \forall 0 < \epsilon' \leq \epsilon,\ \gamma(v + \epsilon')\big)$$

*Le front descendant $\gamma\downarrow$ est définie de même en inversant $v - \epsilon'$ et $v + \epsilon'$.*

Le calcul des fronts est implantable en Hytech [HHWT97]. Pour cela, il faut remarquer que $\gamma\uparrow \equiv \big(\gamma \wedge \exists\epsilon > 0,\ \neg(post_{]0,\epsilon]}(\gamma))\big) \vee \big(\neg\gamma \wedge \exists\epsilon > 0,\ \neg(pre_{]0,\epsilon]}(\neg\gamma))\big)$ (on intervertit *pre* et *post* pour le front descendant). Nous donnons quelques exemples dans le tableau suivant :

| Formule $\gamma$ | $\gamma\uparrow$ | $\gamma\downarrow$ |
|---|---|---|
| $5 \leq x \leq 10$ | $x = 5$ | $x = 10$ |
| $x \leq 5 \wedge y \geq 10$ | $y = 10 \wedge x \leq 5$ | $x = 5 \wedge y >= 10$ |
| $x = 5$ | $x = 5$ | $x = 5$ |
| $x \leq 5 \wedge y \leq 10$ | *ff* | $x = 5 \wedge y <= 10 \vee y = 10 \wedge x <= 5$ |
| $x \leq y + 2 \wedge y \leq 10$ | *ff* | $y = 10 \wedge x <= 12$ |

On introduit un deuxième opérateur également nécessaire à la construction des priorités dépendantes du temps. L'*opérateur modal* associe à une contrainte d'horloge $\phi$ la contrainte élargie $\phi'$ telle que $\phi'$ est vraie si, et seulement si $\phi$ est vraie ou sera vraie dans un certain délai.

**Définition 22 (Opérateur modal [BST98, BS00])** *Soient un ensemble d'horloges $X$, une contrainte d'horloges $\phi \in \mathcal{B}(X)$, un entier $k \in \mathbb{N}$ et une valuation $v$, on définit le prédicat obtenu à partir de l'opérateur modal $\Diamond_k\phi$ par :*

$$(\Diamond_k\phi)(v) \iff \exists t \in \mathbb{T}, t \leq k, \phi(v+t)$$

On note $\Diamond\phi$ le passé de $\phi$, i.e. $\Diamond\phi = \Diamond_\infty\phi$. De même que pour les fronts, on peut calculer les prédicats $\Diamond_k$ avec Hytech en remarquant que $\Diamond_k\phi = Pre_{[0,k]}(\phi)$. Nous donnons quelques exemples dans le Tableau suivant :

| Formule $\gamma$ | $\Diamond_3\gamma$ | $\Diamond\gamma$ |
|---|---|---|
| $5 \leq x \leq 10$ | $2 \leq x \leq 10$ | $x \leq 10$ |
| $x \leq 3$ | $x \leq 3$ | $x \leq 3$ |
| $x \leq 5 \wedge y \geq 10$ | $(0 \leq x \leq 5 \wedge y \geq 10) \vee$ | $(0 \leq x \leq 5 \wedge y \geq 10) \vee$ |
| | $0 \leq x \leq y - 5 \wedge 7 \leq y \leq 10$ | $0 \leq x \leq y - 5 \wedge y \leq 10$ |
| $x \leq y + 2 \wedge y \geq 10$ | $x \leq y + 2 \wedge 7 \leq y$ | $x \leq y + 2$ |

**Préliminaires.** Dans les automates temporisés, l'urgence s'encode par une modification des invariants temporels. Or dans Timed AltaRica, modifier les invariants temporels revient à ajouter des assertions et peut transformer les valeurs de flux admissibles. On suppose que les gardes des transitions urgentes sont fermées à gauche, i.e. $\gamma \uparrow \wedge \gamma = \gamma \uparrow$ car cela n'a pas de sens de dire que la transition deviendra urgente dans un moment infinitésimal. Par exemple, on ne peut donner un sens à la garde est $x > 10$.

Un deuxième constat est le suivant : soient $e \in \mathcal{U}_g(E)$ et $t_u = ((g_u, \gamma_u), e, (a_u, R_u))$ une transition urgente de $M$, pour chaque transition $t = ((g, \gamma), e, (a, R)) \in M$, on compare les successeurs de $t$, à savoir $(a, R)(g \wedge \gamma)$, aux configurations où $t_u$ est tirable, i.e. $g_u \wedge \gamma_u$. Il y a trois cas :

1. les configurations issues de $t$ ne sont pas dans la garde de la transition urgente et lorsque qu'on laisse passer un temps quelconque on n'atteint jamais une configuration pour laquelle on peut tirer $t_u$, c'est-à-dire $(a, R)(g \wedge \gamma) \wedge g_u \wedge \Diamond\gamma_u = ff$ ;
2. les configurations issues de $t$ ne sont pas dans la garde de la transition urgente mais en laissant passer un certain temps, on atteint une configuration où $t_u$ est tirable, c'est-à-dire $(a, R)(g \wedge \gamma) \wedge g_u \wedge \gamma_u = ff$ et $(a, R)(g \wedge \gamma) \wedge g_u \wedge \Diamond\gamma_u \neq ff$ ;
3. certaines configurations issues de $t$ sont dans la garde de la transition urgente, c'est-à-dire $(a, R)(g \wedge \gamma) \wedge g_u \wedge \gamma_u \neq ff$.

On en déduit qu'en chaque configuration discrète de $g_u$, il y a trois possibilités pour les valuations des horloges $(\gamma_u)$ ou $((\Diamond\gamma_u) \wedge \neg\gamma_u)$ ou$(\neg(\Diamond\gamma_u))$. On ne peut atteindre une configuration d'un autre ensemble que par une transition discrète, éventuellement $\epsilon$, mais pas en laissant passer du temps.

**Première procédure de construction de $\mathcal{A}\upharpoonright_{\mathcal{U}_g(E)}$.** Nous proposons ici une première façon d'encoder l'urgence dans un composant temporisé. Cette méthode est un peu artificielle car on insère variables de flot qu'on ne contrôle pas. Nous proposons une deuxième méthode dans la section 4.3.2 page 85 qui utilise la hiérarchie et contrôle les variables introduites. On introduit une variable discrète $U$ qui distinguera ces trois ensembles.

La nouvelle variable $U$ ne peut être une variables d'état sinon elle modifierait la sémantique des transitions $\epsilon$. En effet,

1. si on peut passer par une transition $\epsilon$ d'un ensemble à l'autre, alors $U$ est nécessairement un flux ;
2. si aucune transition $\epsilon$ ne relie les trois ensembles, alors les invariants temporels contraindront les comportements et aucune nouvelle transition $\epsilon$ n'apparaîtra.

On introduit également une horloge $Y$ qui contraindra le composant à ne pas rester dans une configuration urgente. Pour ne pas modifier les transitions $\epsilon$, $Y$ est également une horloge de flux. De plus, $U$ et $Y$ sont des variables de flux auxquelles on ajoute l'attribut de visibilité `private`[3] à ces deux variables pour ne pas modifier l'interface du composant. Une manière équivalente sans utiliser cet artifice consisterait à encapsuler le composant dans un nœud temporisé qui cacherait $U$ et $Y$.

La construction se fait par induction sur les événements urgents et les transitions étiquetées par ces événements.

**Données :** $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ et $\mathcal{U}_g(E)$ un ensemble d'événements urgents
**Résultat :** $\mathcal{A}\upharpoonright_{\mathcal{U}_g(E)} = (V'_S, C'_S, V'_F, C'_F, E, A', M)$
on pose $C'_S = C_S$, $V'_S = V_S$, $C'_F = C_F$, $V'_F = V_F$ et $A' = A$
**pour tout** $e_u \in \mathcal{U}_g(E)$ **faire**
  **pour tout** $t_{u_i} = ((g_{u_i}, \gamma_{u_i}), e_u, (a_{u_i}, R_{u_i})) \in M$ **faire**
    $C'_F = C'_F \cup \{Y\}$, $V'_F = V'_F \cup \{U_i\}$ avec $U_i \in [0,2]$, $U_i \notin V_S \cup V_F$ et $Y \notin C_S \cup C_F$ {on introduit une horloge de flux $Y$ et une variable de flux $U_i$}
    $U_i$ et $Y$ sont `private`

$$A' = A' \bigwedge U_i = 2 \wedge g_u \implies \gamma_u \wedge Y = 0$$
$$\bigwedge U_i = 1 \wedge g_u \implies \Diamond(\gamma_u \uparrow)$$
$$\bigwedge U_i = 0 \wedge g_u \implies \neg\Diamond\gamma_u$$

    { on ajoute des invariants temporels de sorte que de toute configuration permettant de faire une transition urgente le temps ne peut s'écouler}
  **fin pour**
**fin pour**

**Algorithme 4.1:** Procédure de résolution syntaxique de l'urgence

**Exemple 19** *On reprend le composant temporisé sans variable de flux de l'exemple 18 page 66 et on applique l'Algorithme 4.1, on obtient alors le composant temporisé suivant :*

```
node Ex_urgence_sans_flux_reecrit
flow U : [0,2] : private; Y: clock : private;
state s : [0,1]; x :clock;
event e > time, f,g
trans s=1 & 10 <= x <=20 |- e -> s:=0;
   vrai |- f -> s:=0;
```

[3] Les attributs de visibilité ont été définis dans la section 2.11 page 30

```
   vrai |- g -> s:=1;
tinvariant U=2 & s=1 => 10<=x<=20 & Y=0;
    U=1 & s=1 => x<=10;
    U=0 & s=1 => x>20;
edon
```

*On constate qu'on n'a pas ajouté de transition* $\epsilon$ *au nœud d'origine puisque* $\pi(0,x) = [0,2] \times \mathbb{R}$ *et* $\pi(1,x) = \begin{cases} 0 \times 0 \text{ si } x \in [10,20] \\ 1 \times \mathbb{R} \text{ si } x \leq 10 \\ 2 \times \mathbb{R} \text{ si } x > 20 \end{cases}$

**Exemple 20** *On considère le composant temporisé suivant :*

```
node Ex_urgence
flow R : [0,1]; X : clock;
state s : [0,1]
event e > time
trans R=1 & 10 <= X <=20 |- e -> s:=0;
edon
```

*On applique l'Algorithme 4.1 de réécriture au composant temporisé* Ex_urgence *et nous obtenons le composant temporisé* Ex_urgence_reecrit. *On a une unique transition urgente* R=1 $\wedge$ $10 \leq X \leq 20|-$ e $->$ s :=0. *Donc, on a* $g_u \equiv R = 1$ *et* $\gamma_u \equiv x \in [10,20]$. *On en déduit le composant temporisé suivant.*

```
node Ex_urgence_reecrit
flow R : [0,1]; X : clock;
     U : [0,2] : private; Y: clock : private;
state s : [0,1]
event e
trans R=1 & 10 <= X <=20 |- e -> s:=0;
tinvariant U=2 & R=1 => 10<=X<=20 & Y=0;
    U=1 & R=1 => X<=10;
    U=0 & R=1 => X>20;
edon
```

*On peut représenter la sémantique de ce composant temporisé : une configuration discrète est de la forme* $(s,R,U)$. *On a décomposé pour plus de lisibilité le STTI* $[\![A \upharpoonright_{\mathcal{U}_g(E)}]\!]$ *en la Figure 4.11 correspondant aux* $\epsilon$*-transitions et la Figure 4.12 pour les transitions discrètes e.*

### L'urgence dans les STTI

Etant donnés un STTI $\mathcal{A}$ et un ensemble d'événements urgents $\mathcal{U}_g(E)$, on exprime le STTI $\mathcal{A} \upharpoonright_{\mathcal{U}_g(E)}$ bisimilaire qui respecte l'urgence à savoir on ne peut laisser passer un temps $t$ dans une configuration $(q,g)$ que si on ne traverse pas de configuration permettant de tirer une transition eager. Pour cela, on introduit un opérateur qui contraint les transitions continues.

**Définition 23 (Opérateur de restriction d'urgence [Pag03b])** *Soient* $\mathcal{A} = \langle E_t, F_t, S_t, \pi, T\rangle$ *un STTI de dimension* $(m,n)$ *et* $\mathcal{U}_g(E)$ *un ensemble d'événements urgents. On définit l'opérateur de restriction d'urgence* $\upharpoonright$ *pour la relation de transition* $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ *et l'ensemble* $\mathcal{U}_g(E)$ *par :* $(q,g,e,q',g') \in T \upharpoonright_{\mathcal{U}_g(E)} \Longleftrightarrow \big[(q,g,e,q',g') \in T \wedge \big(e = t \in \mathbb{T}, \forall t' \in \mathbb{T}, t' \leq t, (q,g,t',q+t',g+t') \in T \Rightarrow \forall e' \in E_+, (q+t',g+t',e',q'') \in T \Longrightarrow e' \not> time\big)\big]$. *On pose alors* $\mathcal{A} \upharpoonright_{\mathcal{U}_g(E)} = \langle E_t, F_t, S_t, \pi, T \upharpoonright_{\mathcal{U}_g(E)}\rangle$.

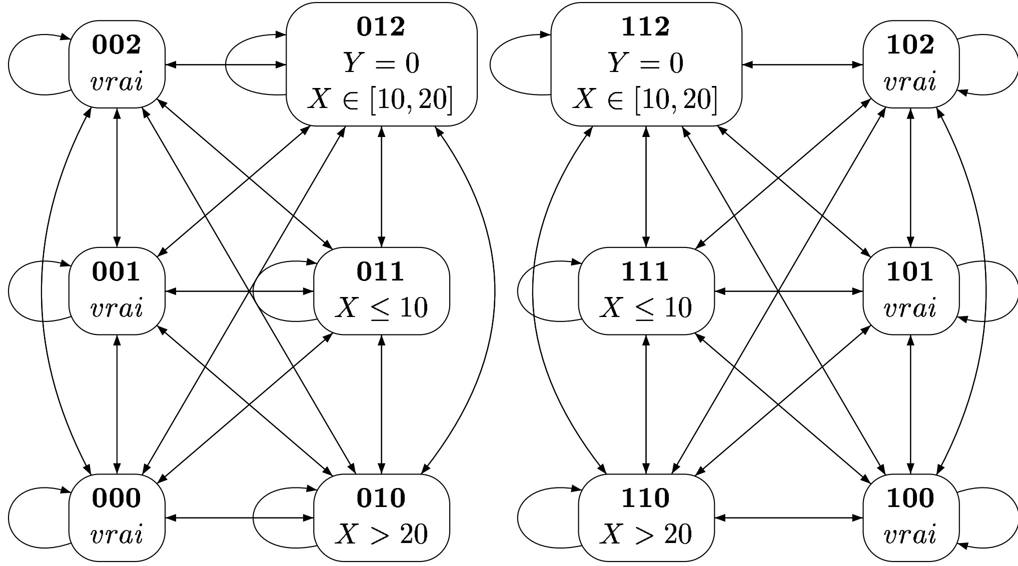



FIG. 4.11 – $[\![A \restriction_{\mathcal{U}_g(E)}]\!]$ - transitions $\big(vrai, \epsilon, X \in \mathbb{R} \wedge Y \in \mathbb{R}\big)$

**Proposition 2 (Equivalence de la résolution de l'urgence syntaxique et sémantique)** *Soient $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ un composant temporisé et $\mathcal{U}_g(E)$ un ensemble d'événements urgents, alors $[\![\mathcal{A}]\!] \restriction_{\mathcal{U}_g(E)} = (S_t, F_t, E_t, \pi, T \restriction_{\mathcal{U}_g(E)})$ et $[\![\mathcal{A} \restriction_{\mathcal{U}_g(E)}]\!] = (S_t, F_t, E_t, \pi', T')$ sont bisimilaires temporellement.*

**Preuve de la proposition 2**. *On considère l'application identité $h : S_t \to S_t$, donc $h$ est surjective. Il reste à prouver que les applications $\pi$ et $\pi'$ coïncident puis que les relations de transitions $T \restriction_{\mathcal{U}_g(E)}$ et $T'$ sont identiques. On pose $p = \big|\{((g_u, \gamma_u), e_u, (a_u, R_u)) \in M |\ e_u \in \mathcal{U}_g(E)\}\big|$ et $n = 3p$. On remarque que l'application $\pi'$ est définie par :*

$$\begin{aligned}
\pi'(q) &= \{g \mid \exists U = (U_1, \cdots, U_n), Y.(q, g, U, Y) \in [\![A']\!]\} \\
&= \{g \mid g \in [\![A]\!] \wedge \forall i \in [1,p], \begin{cases} (q,g) \in [\![\neg g_{u_i}]\!] \implies U_i \in [0,2], Y \in \mathbb{R} \\ (q,g) \in [\![g_{u_i} \wedge \gamma_{u_i}]\!] \implies U_i = 0 \wedge Y = 0 \\ (q,g) \in [\![g_{u_i} \wedge \Diamond(\gamma_{u_i} \uparrow)]\!] \implies U_i = 1 \\ (q,g) \in [\![g_{u_i} \wedge \neg\Diamond(\gamma_{u_i})]\!] \implies U_i = 2 \end{cases} \\
&= \pi(q)
\end{aligned}$$

*En effet, si $g \in [\![A]\!]$, on trouve toujours au moins une valeur de $U$ et $Y$ telles que $(q, g, U, Y) \in [\![A']\!]$.*

*L'opérateur de restriction d'urgence ne modifie que les transitions temporelles. Soit une transition $(q, g, \delta, q + \delta, g + \delta) \in T$, il y a deux possibilités :*

1. *on a $(q, g, \delta, q + \delta, g + \delta) \in T \restriction_{\mathcal{U}_g(E)}$. Alors on a :*

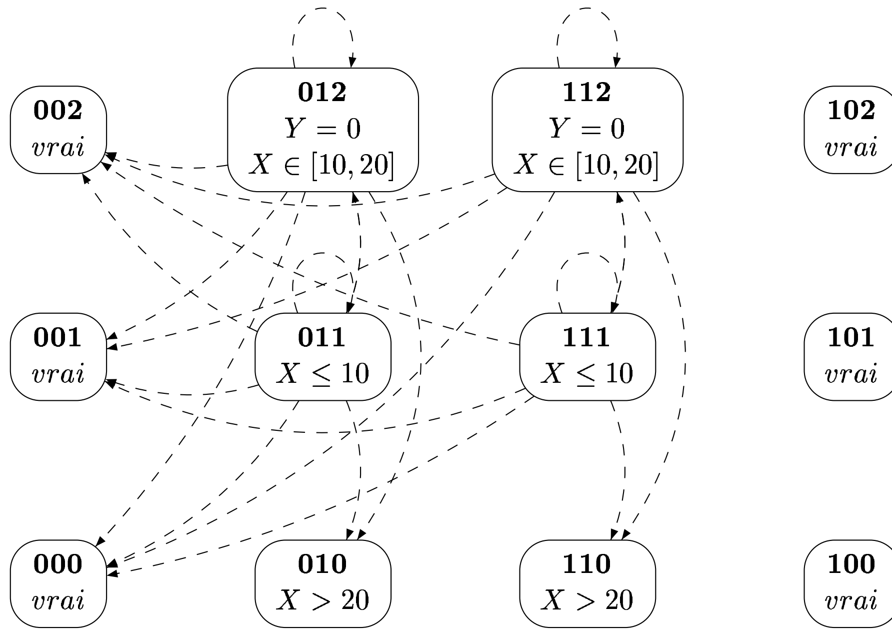



FIG. 4.12 – $[\![A \restriction_{\mathcal{U}_g(E)}]\!]$ - transitions $\big(X \in [10,20], e, X \in \mathbb{R} \wedge Y \in \mathbb{R}\big)$

$$
\begin{array}{ll}
(q,g,\delta,q+\delta,g+\delta) \in T \restriction_{\mathcal{U}_g(E)} & \\
\quad \Rightarrow & \forall i \in [1,p],\ \forall \delta' \leq \delta, (q+\delta', g+\delta') \notin [\![g_{u_i} \wedge \gamma_{u_i}]\!] \\
\quad \Rightarrow & \forall i \in [1,p], (q,g) \notin [\![g_{u_i} \wedge \Diamond_\delta(\gamma_{u_i})]\!] \\
\quad \Rightarrow & \forall i \in [1,p],\ \big((q,g) \notin [\![g_{u_i}]\!]\big) \vee \big((q,g) \in [\![g_{u_i}]\!] \wedge (q,g) \notin [\![\Diamond_\delta(\gamma_{u_i})]\!]\big) \\
\quad \Rightarrow & \forall i \in [1,p],\ \big((q,g) \notin [\![g_{u_i}]\!]\big) \vee \big((q,g) \in [\![g_{u_i}]\!] \wedge (q,g) \in [\![\neg\Diamond(\gamma_{u_i})]\!]\big) \\
 & \qquad \vee\big((q,g) \in [\![g_{u_i}]\!] \wedge (q,g) \in [\![\Diamond(\Diamond_\delta(\gamma_{u_i}) \uparrow)]\!]\big) \\
\quad \Rightarrow & \left\{ \begin{array}{l}
\textit{Si } (q,g) \notin [\![g_{u_i}]\!],\ U_i \textit{ peut avoir n'importe quelle valeur et on} \\
\qquad \textit{peut laisser écouler n'importe que laps de temps admis} \\
\qquad \textit{par l'assertion} \\
\textit{Si } (q,g) \in [\![g_{u_i}]\!] \wedge (q,g) \in [\![\neg\Diamond(\gamma_{u_i})]\!],\ \textit{nécessairement } U_i = 2 \\
\qquad \textit{et on peut laisser passer } \delta \textit{ u.t.} \\
\textit{Si } (q,g) \in [\![g_{u_i}]\!] \wedge (q,g) \in [\![\Diamond(\Diamond_\delta(\gamma_{u_i}) \uparrow)]\!],\ \textit{on a nécessaire-} \\
\qquad \textit{ment } U_i = 1 \textit{ et on peut laisser passer un temps} \leq \delta
\end{array} \right. \\
\quad \Rightarrow & (q,g,\delta,q+\delta,g+\delta) \in T'
\end{array}
$$

*2. on a* $(q,g,\delta,q+\delta,g+\delta) \notin T \restriction_{\mathcal{U}_g(E)}$. *Or,*

$$
\begin{array}{l}
(q,g,\delta,q+\delta,g+\delta) \notin T \restriction_{\mathcal{U}_g(E)} \\
\quad \Rightarrow \quad \exists \delta' < \delta,\ \exists e_u > time,\ (q,g,\delta',q+\delta',g+\delta')\ et\ (q+\delta',g+\delta',e_u,q',g') \in T \\
\quad \Rightarrow \quad \exists e_u > time,\ \exists \delta' < \delta, \begin{cases} \delta' = 0,\ alors\ (q,g) \in [\![g_u \wedge \gamma_u]\!] \\ \delta' > 0\ alors\ (q,g) \in [\![g_u \wedge \Diamond_{\delta'}(\gamma_u \uparrow)]\!] \end{cases} \\
\quad \Rightarrow \quad \exists e_u > time,\ \exists \delta' < \delta, \\
\qquad \begin{cases} \delta' = 0 \wedge (q,g) \in [\![g_u \wedge \gamma_u]\!]\ alors\ U_i = 2\ et\ Y = 0\ le\ temps\ ne \\ \quad peut\ s'\acute{e}couler\ (q,g,U_1,\cdots,2,\cdots,0,e_u,q',g') \in T' \wedge \forall t > 0, \\ \quad (q,g,U_1,\cdots,2,\cdots,0,t,q+t) \notin T' \\ \delta' > 0 \wedge (q,g) \in [\![g_u \wedge \Diamond_{\delta'}(\gamma_u \uparrow)]\!]\ alors\ U_i = 1\ et\ le\ temps\ ne \\ \quad peut\ s'\acute{e}couler\ que\ jusqu'\grave{a}\ \delta'\ et\ apr\grave{e}s\ il\ faut\ r\acute{e}aliser \\ \quad une\ transition\ discr\grave{e}te\ puisqu'on\ arrive\ \grave{a}\ la\ limite\ de \\ \quad l'invariant\ temporel,\ e_u\ est\ entre\ autre\ tirable \end{cases} \\
\quad \Rightarrow \quad (q,g,\delta,q+\delta,g+\delta) \notin T'
\end{array}
$$

*Finalement, on obtient* $T \restriction_{\mathcal{U}_g(E)} = T'$ *ce qui termine la démonstration. On obtient, de même que pour AltaRica et la résolution des priorités, le diagramme commutatif donné à la Figure 4.13.*

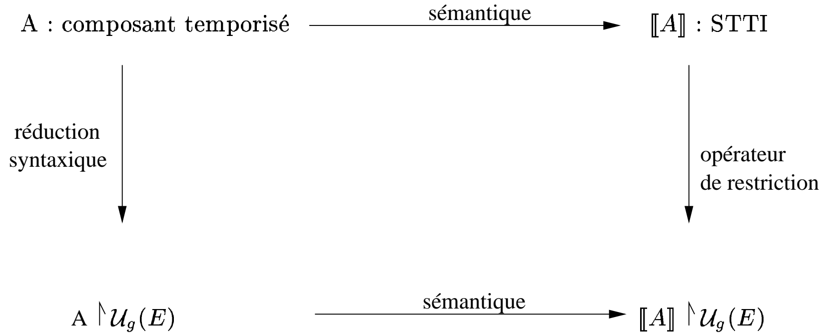



FIG. 4.13 – Diagramme commutatif sémantique/urgence

□

**Exemple 21** *On reprend l'exemple 20, seulement cette fois on donne la sémantique du composant temporisé* Ex_urgence *et on applique l'opérateur de restriction d'urgence. On obtient alors le STTI représenté à la Figure 4.14. On obtient le même STTI que celui représenté dans les Figures 4.11 et 4.12 lorsqu'on élimine la variable de flux $U$. Nous sommes contraints de conserver l'horloge $Y$ pour représenter l'urgence de certains états comme cela est fait pour les automates temporisés.*

On peut alors combiner l'urgence et les priorités, ainsi deux événements urgents peuvent être ordonnés de même qu'un événement urgent et un événement quelconque. Dans [CPR02], on regroupe une relation de priorité et un ensemble d'événements urgents sous le nom de relation de priorité temporelle simple.

**Définition 24 (Relation de priorité temporelle simple [CPR02])** *Soit $\mathcal{U}_g(E)$ un ensemble d' événements urgents, on appelle* relation de priorité temporelle simple $<$ *sur $E$ l'ordre partiel strict défini sur $E_+^{time} = E \cup \{time\}$ tel que :*

1. $<_{|E}$ *est une relation de priorité (cf. Définition 4 page 27),*
2. $\forall v \in E_+, v \nless time.$

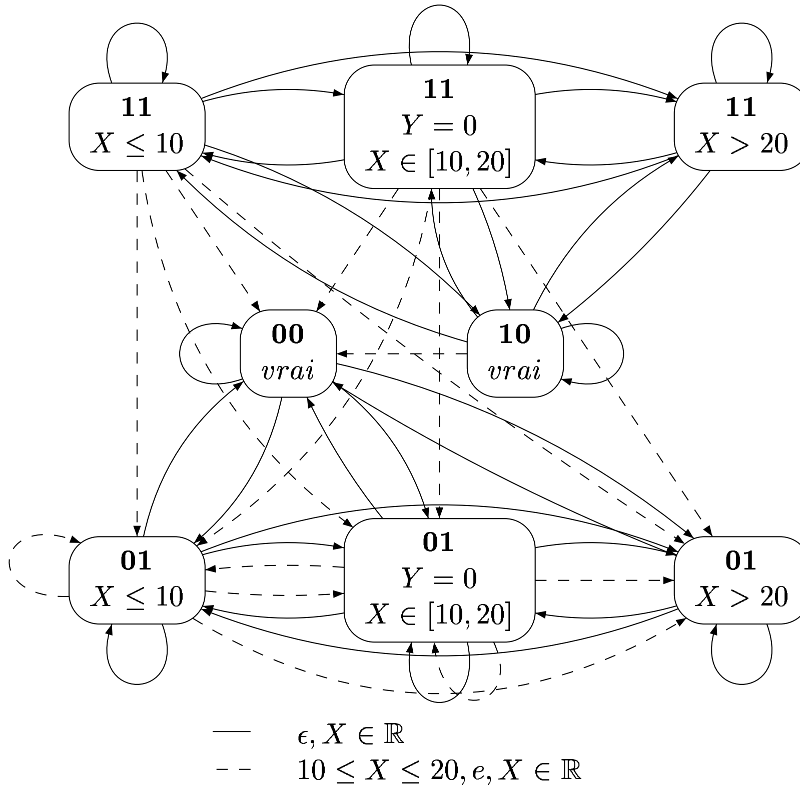



FIG. 4.14 – $[\![A]\!] \upharpoonright_{\mathcal{U}_g(E)}$

Syntaxiquement, on réalise les deux opérations de restriction $\upharpoonright_<$ puis $\upharpoonright_{\mathcal{U}_g(E)}$ présentées précédemment. Sémantiquement, l'effet d'une priorité temporelle simple est à la fois celui de la relation de priorité et celui de l'urgence. Ainsi, étant donnés un STTI $\mathcal{A} = \langle E_t, F_t, S_t, \pi, T \rangle$ de dimension $(m, n)$ et $<$ une relation simple, la relation de transitions du STTI restreint par la relation simple est $T \upharpoonright_<$ et vérifie : $(q, g, e, q') \in T \upharpoonright_< \Leftrightarrow (q, g, e, q', g') \in T \wedge$

$$\begin{cases} si\ e \in E\ alors\ \forall e' \in E, (q, g, e', q'', g'') \in T \Rightarrow \neg e < e' \\ si\ e = t \in \mathbb{T}, \forall t' \in \mathbb{T}, t' \leq t,\ (q, g, t', q + t', g + t') \in T \Rightarrow \\ \qquad\qquad \forall e' \in E_+, (q + t', g + t', e', q'', g'') \in T \Longrightarrow \neg e' > time. \end{cases}$$

Nous avons présenté une première extension des priorités d'AltaRica. En mêlant urgence et priorité, nous obtenons des *priorités temporelles simples*. Dans la prochaine section, nous présentons une nouvelle généralisation des priorités. Les *priorités temporelles* [BST98, BS00, BGS00] influent sur les transitions discrètes contrairement à l'urgence qui ne modifie que les transitions continues.

### 4.2.3 Cas général : les priorités temporelles

Les priorités temporelles [BST98, BS00, BGS00] sont une fonctionnalité intéressante dans les automates puisqu'on modélise des comportements comme *l'événement e n'est possible que si l'événement f ne peut être réalisé dans les k prochaines minutes*. Ce comportement est noté $e <_k f$.

On rappelle la définition formelle d'une relation de priorité temporelle.

**Définition 25 (Relation de priorité temporelle [BGS00, CPR02])** *Une* relation de priorité temporelle $<$ *sur un ensemble d'événements fini E est une relation d'arité 3 dans* $E_+^{time} \times (\{\mathbb{N} \cup$

$\{\infty\}) \times E_{+}^{time}$, *avec* $E_{+}^{time} = E_{+} \cup \{time\}$, *satisfaisant les conditions suivantes (on note* $a_1 <_k a_2$ *pour* $(a_1, k, a_2) \in <$*) :*

- *la relation binaire* $<_0$ *est une relation de priorité temporelle simple,*
- $e <_k e' \wedge e = time \Longrightarrow k = 0$,
- $\forall k \in \mathbb{N}, <_k$ *est un ordre partiel strict,*
- $e <_k e' \Longrightarrow \big(\forall k' < k, e <_{k'} e'\big)$,

**Remarque 4** *La relation de priorité classique d'AltaRica et l'urgence sont intégrées dans les priorités temporelles d'après la première condition.*

### Les priorités temporelles dans Timed AltaRica

La syntaxe de l'opérateur de priorité temporelle sera proche de la définition précédente et on note $e_1 <_k e_2$ par $e_1(<, k)e_2$.

```
event e1 (<,k) e2
```

**Exemple 22** *On considère le composant temporisé suivant :*

```
node Ex_priorite_temporelle
flow R : [0,1]; X : clock;
state s : [0,1]; x : clock;
event e (<,5) f
trans R=1 & 10 <= X <=20 |- e -> s:=0;
    s=0 & x>=15 |- f -> s:=1, x:=0;
edon
```

*On représente partiellement le comportement de ce composant temporisé en la configuration* $(s = 0, x = 7, R = 1, X = 10)$. *Dans la Figure 4.15, on représente une partie du STTI sémantique du composant* Ex_priorite_temporelle *sans prendre en compte la priorité temporelle* $e <_5 f$. *Dans la Figure 4.16, on ne conserve que les transitions respectant la priorité temporelle. On remarque que seules les transitions discrètes, autres que* $\epsilon$*, sont concernées.*

*Dans la Figure 4.16, on ne peut plus exécuter e depuis les configurations* $(0, 17, 1, 10)$ *et* $(0, 12, 1, 10)$ *puisque la transition* $f$ *est tirable en moins de* 5 *u.t.*

### Les priorités temporelles dans les STTI

Etant donnés un STTI $\mathcal{A}$ et une relation de priorité temporelle $<$, on exprime le STTI $\mathcal{A} \restriction_<$ bisimilaire qui respecte la relation de priorité temporelle. Pour cela, on introduit un opérateur qui contraint les transitions discrètes et continues.

**Définition 26 (Opérateur de restriction de priorité temporelle [CPR02])** *Soient* $\mathcal{A} = \langle E_t, F_t, S_t, \pi, T\rangle$ *un STTI de dimension continue* $(m, n)$ *et* $<$ *une relation de priorité temporelle sur* $E$. *On définit l'*opérateur de restriction de priorité temporelle $\restriction$ *pour la relation de transitions* $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ *et la relation de priorité temporelle* $<$ *par :*

$$(q,g,e,q',g') \in T \restriction_< \Leftrightarrow (q,g,e,q',g') \in T \wedge \begin{cases} si\ e = t \in \mathbb{T}, \forall t' \in \mathbb{T}, t' \le t,\ si\ (q,g,t',q+t',g+t') \in T \\ \qquad alors\ \forall e' \in E_{+}, (q+t', g+t', e', q'', g'') \in T \Longrightarrow \\ \qquad\qquad time \not<_0 e'. \\ sinon\ si\ (q,g,t,q+t,g+t) \in T, t \in \mathbb{T}, t \le k, \\ \qquad alors\ \forall e', (q+t, g+t, e', q'', g'') \in T \Longrightarrow e \not<_k e'. \end{cases}$$

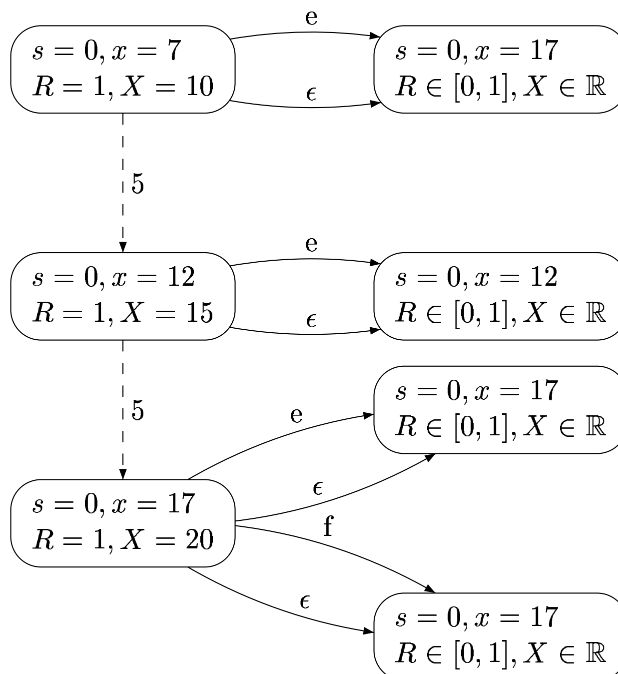



FIG. 4.15 – Quelques transitions issues de $(0, 7, 1, 10) \in [\![Ex_priorite_temporelle]\!]$ sans priorité

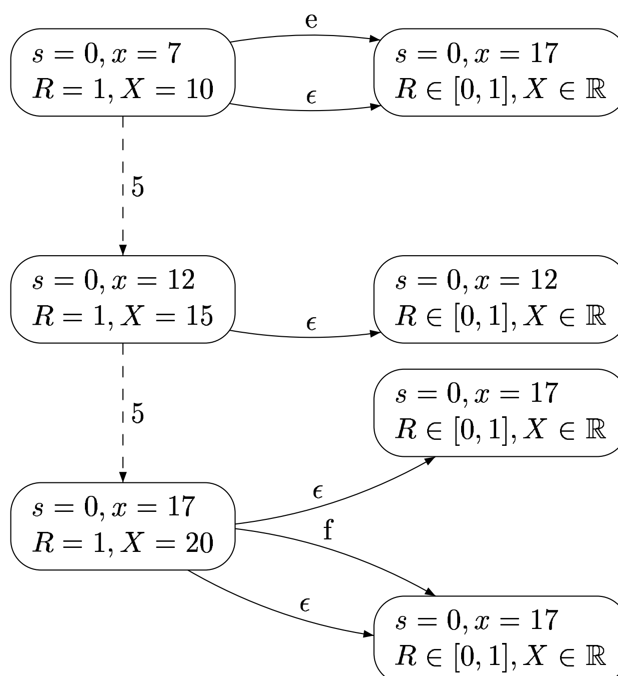



FIG. 4.16 – Les mêmes transitions issues de $(0, 7, 1, 10) \in [\![Ex_priorite_temporelle]\!]$ avec priorité

*On note* $\mathcal{A}\restriction_{<} = \langle E_t, F_t, S_t, \pi, T\restriction_{<}\rangle$.

La bisimulation temporisée interfacée est stable par relation de priorité temporelle, le théorème 1 page 27 s'étend au cas temporisé de la façon suivante :

**Théorème 4 (Stabilité de la bisimulation temporisée interfacée par priorité [CPR02])** *Soient deux STTI* $\mathcal{A}_i = (S_{t,i}, F_t, E_t, \pi_i, T_i), i = 1, 2$ *et* $<$ *une relation de priorité temporelle sur* $E$. *Si* $\mathcal{A}_1$ *et* $\mathcal{A}_2$ *sont bisimilaires temporellement, alors* $\mathcal{A}_1\restriction_{<}$ *et* $\mathcal{A}_2\restriction_{<}$ *le sont aussi.*

**Preuve du théorème 4**. *Soit* $h : \mathcal{A}_1 \longrightarrow \mathcal{A}_2$ *un homomorphisme de bisimulation interfacée temporisée. On montre que* $h$ *est également un homomorphisme de bisimulation temporisée interfacée de* $\mathcal{A}_1\restriction_{<}$ *sur* $\mathcal{A}_2\restriction_{<}$.

*L'opérateur de restriction de priorité temporelle* $\restriction_{<}$ *n'intervient que sur la relation de transitions et non sur l'ensemble des flux admissibles. Les points 1 et 2 de la définition d'un homomorphisme de bisimulation temporisée interfacée sont vérifiés.*

*Pour le point 3, soit* $(q_1, g, e, q_1', g') \in T_1\restriction_{<}$, *alors* $(h(q_1), g, e, h(q_1'), g') \in T_2$. *On suppose que* $(h(q_1), g, e, h(q_1'), g') \notin T_2\restriction_{<}$, *alors d'après la définition 26 il y a deux cas :*

1. *soit* $e = t \in \mathbb{T}$ *et* $\exists e' >_0 time, t' < t, (h(q_1), g, t', q_2'', g'') \in T_2 \wedge (q_2'', g'', e', q_2''', g''') \in T_2$. *Puisque* $h$ *est un homomorphisme et* $(h(q_1), g, t', q_2'', g'')$ *et* $(q_2'', g'', e', q_2''', g''')$ *sont dans* $T_2$, *alors* $\exists q_1'', q_1''' \in S_{1_t}, h(q_1'') = q_2'', h(q_1''') = q_2''' \wedge (q_1, g, t', q_1'', g'')$ *et* $(q_1'', g'', e', q_1''', g''')$ *sont dans* $T_1$. *D'où* $(q_1, g, e, q_1', g')$ *ne peut appartenir à* $T_1\restriction_{<}$ *ce qui contredit l'hypothèse de départ.*
2. *soit* $e \in E_+$ *et* $\exists e', e <_k e' \wedge (h(q_1), g, t, q_2'', g'') \in T_2, t \in \mathbb{T}, t \leq k \wedge (q_2'', g'', e', q_2''', g''') \in T_2$. *Puisque* $h$ *est un homomorphisme et* $(h(q_1), g, t, q_2'', g'')$ *et* $(q_2'', g'', e', q_2''', g''')$ *sont dans* $T_2$, *alors* $\exists q_1'', q_1''' \in S_{1_t}, h(q_1'') = q_2'', h(q_1''') = q_2'''$ *et* $(q_1, g, t, q_1'', g'')$ *et* $(q_1'', g'', e', q_1''', g''')$ *qui appartiennent à* $T_1$. *Ce qui contredit une nouvelle fois que* $(q_1, g, e, q_1', g') \in T_1\restriction_{<}$. *D'où le point 3.*

*Pour le point 4, on considère* $q_1 \in S_{1_t}, q_2' \in S_{2_t}$ *tel que* $(h(q_1), g, e, q_2', g') \in T_2\restriction_{<}$, *alors* $\exists q_1' \in S_{1_t}, h(q_1') = q_2' \wedge (q_1, g, e, q_1', g') \in T_1$. *On suppose à nouveau que pour tout* $q_1', h(q_1') = q_2'$, $(q_1, g, e, q_1', g') \notin T_1\restriction_{<}$ *:*

1. *si* $e \in \mathbb{T}$, *cela signifie que* $\exists t_1 \leq t, (q_1, g, t_1, q_1'', g'') \in T_1$ *et* $\exists e' >_0 time$ *tirable depuis* $(q_1'', g'')$. *Alors puisque* $h$ *est un homomorphisme,* $(h(q_1), g, t_1, h(q_1''), g'') \in T_2 \wedge (h(q_1''), g'', e', q_2''', g''') \in T_2$ *et cela contredit* $(h(q_1), g, e, q_2', g') \in T_2\restriction_{<}$.
2. *si* $e$ *est dans* $E_+$ *et qu'il existe* $e <_k e', t \leq k$ *tel que* $(q_1, g, t, q_1'', g''), (q_1'', g'', e', q_1''', g''') \in T_1$. *On en déduit que* $(h(q_1), g, t, h(q_1''), g''), (h(q_1''), g'', e', h(q_1'''), g''') \in T_2$. *Cela contredit à nouveau l'hypothèse* $(h(q_1), g, e, q_2', g') \in T_2\restriction_{<}$. *D'où le point 4 et la démonstration du théorème.*

□

Puisqu'une relation d'urgence est en particulier une relation de priorité temporelle, on en déduit immédiatement que la bisimulation temporisée interfacée est stable par relation d'urgence.

**Corollaire 1 (Stabilité de la bisimulation temporisée interfacée par urgence)** *Soient deux STTI* $\mathcal{A}_i = (S_{t,i}, F_t, E_t, \pi_i, T_i), i = 1, 2$ *et* $\mathcal{U}_g(E)$ *un ensemble d'événements urgents. Si* $\mathcal{A}_1$ *et* $\mathcal{A}_2$ *sont bisimilaires temporellement, alors* $\mathcal{A}_1\restriction_{\mathcal{U}_g(E)}$ *et* $\mathcal{A}_2\restriction_{\mathcal{U}_g(E)}$ *le sont aussi.*

### Résolution syntaxique des priorités temporelles

De même que pour l'urgence, un composant temporisé $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ muni d'une priorité temporelle $<$ peut se réécrire syntaxiquement en un composant temporisé sans priorité $\mathcal{A} \upharpoonright_< = (V'_S, C'_S, V'_F, C'_F, E, A', M')$. On suppose qu'il n'y a aucun événement urgent, le cas de l'urgence ayant été traité dans le paragraphe section 4.2.2 page 65. S'il n'y a pas d'événements urgents, la relation de priorité temporelle ne modifie que les transitions discrètes et $\mathcal{A} \upharpoonright_< = (V_S, C_S, V_F, C_F, E, A, M')$.

**Procédure de construction de $\mathcal{A} \upharpoonright_<$.** Dans les automates temporisés, les priorités temporelles s'encodent par une modification des gardes de certaines transitions [BGS00]. Or dans Timed AltaRica, il faudra évaluer les configurations à partir desquelles deux transitions sont comparables.

Ainsi, si $e <_k e'$, on considère deux transitions $t = ((g, \gamma), e, (a, R)) \in M$ et $t' = ((g', \gamma'), e', (a', R')) \in M$. Pour toute configuration $(s, \nu, f, \mu) \in [\![A]\!]$, il y a trois possibilités :

1. si $(s, f) \in [\![g \wedge \neg g']\!]$ alors la transition $t'$ ne sera jamais réalisée,
2. si $(s, f) \in [\![g \wedge g']\!]$, il y a deux cas :
   (a) soit $(\nu, \mu) \in [\![\gamma \wedge \Diamond_k \gamma']\!]$, alors la transition $t'$ est tirable depuis la configuration $(s, \nu, f, \mu)$ dans moins de $k$ u.t.,
   (b) soit $(\nu, \mu) \in [\![\gamma \wedge \neg(\Diamond_k \gamma')]\!]$, alors la transition $t'$ n'est pas tirable depuis la configuration $(s, \nu, f, \mu)$ dans moins de $k$ u.t.

On scinde alors la transition $t$ en $t_1 = (g \wedge \neg g', \gamma), e, (a, R))$ et $t_2 = (g \wedge g', \gamma \wedge \neg(\Diamond_k \gamma')), e, (a, R))$. On réitère ce processus pour toutes les transitions $t'$ étiquetées par $e'$ puis on applique une deuxième induction sur les événements $e''$ comparables à $e$. On écrit l'Algorithme 4.2 d'encodage syntaxique d'une relation de priorité temporelle pour un composant temporisé. Cette procédure termine toujours car il n'y a qu'un nombre fini de priorités, de transitions et de variables. Le nombre de boucles *répéter* dépend de l'ordre dans lequel sont choisies les relations $e <_k e'$, une optimisation consisterait à prendre l'événement le plus grand pour l'ordre jusqu'au plus petit.

**Données :** $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ et $<$ une relation de priorité temporelle
**Résultat :** $\mathcal{A} \upharpoonright_< = (V_S, C_S, V_F, C_F, E, A, M')$
on pose $M' = M$
**répéter**
  **pour tout** $e <_k e' \in <$ **faire**
    **pour tout** $t = ((g, \gamma), e, (a, R)) \in M$ **faire**
      on pose $r = \left|\{t_i | t_i = ((g_i, \gamma_i), e', (a_i, R_i)) \in M\}\right|$
      $\forall T, F \subseteq [1, r], T \cap F = \emptyset, T \cup F = [1, r]$ { on construit toutes les partitions à deux éléments de $[1, r]$}
      $t_T^F = ((g_T^F, \gamma_T^F), e, (a, R))$ {on construit les transitions $t_T^F$}
      avec $g_T^F = g \wedge \bigwedge_{j \in T} (\wedge_{j \in T} g_j) \bigwedge (\wedge_{j \in F} \neg g_j)$ et $\gamma_T^F = \gamma \wedge \bigwedge_{j \in T} \neg \Diamond_k \gamma_j$
      $M' = M' \setminus \{t\} \cup \{t_T^F | \forall T, F \subseteq [1, r], T \cap F = \emptyset, T \cup F = [1, r]\}$ {on modifie la relation de transitions}
    **fin pour**
  **fin pour**
**jusqu'à** $M'$ invariant

**Algorithme 4.2:** Procédure de résolution syntaxique des priorités temporelles

**Exemple 23** *On reprend l'exemple 22 page 74. On réécrit le composant temporisé* Ex_priorite_temporelle *avec l'Algorithme 4.2 et on obtient le composant temporisé suivant :*

```
node Ex_priorite_temporelle_reecrit
flow R : [0,1]; X : clock;
state s : [0,1]; x:clock;
event e, f
trans R=1 & 10 <= X <=20 & s=1 |- e -> s:=0;
    R=1 & 10 <= X <=20 & s=0 & x < 10 |- e -> s:=0;
    s=0 & x>=15 |- f -> s:=1, x:=0;
edon
```

Etant donnés un composant temporisé $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ et une relation de priorité temporelle $<$, il est équivalent de calculer la sémantique du composant temporisé sans priorité temporelle $\mathcal{A} \upharpoonright_<$ construit syntaxiquement selon la Procédure 4.2 ou d'appliquer l'opérateur de restriction sur la sémantique de $\mathcal{A}$.

**Proposition 3 (Equivalence des résolutions syntaxique et sémantique [CPR02])** *Soit* $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ *un composant temporisé et* $<$ *une relation de priorité temporelle, alors* $[\![\mathcal{A}]\!] \upharpoonright_< = [\![\mathcal{A} \upharpoonright_<]\!]$.

**Preuve de la proposition 3**. *On pose* $[\![\mathcal{A}]\!] \upharpoonright_< = (S_t, F_t, E_t, \pi, T \upharpoonright_<)$ *et* $[\![\mathcal{A} \upharpoonright_<]\!] = (S_t, F_t, E_t, \pi, T')$. *Il suffit de montrer que* $T \upharpoonright_< = T'$. *Soit* $(q, g, e, q', g') \in T$, *il y a deux cas :*

1. *soit* $e \in \mathbb{T}$, *alors* $(q, g, e, q+e, g+e) \in T \upharpoonright_<$ *et* $(q, g, e, q+e, g+e) \in T'$. *En effet on a supposé qu'il n'y avait pas d'urgence, sinon il faut également appliquer la résolution syntaxique de l'urgence proposée dans l'Algorithme 4.1 page 68.*
2. *soit* $e \in E$, *et alors* $\exists t = ((g, \gamma), e, a, R)) \in M$ *telle que* $(q, g, e, q', g') \in [\![t]\!]$,

$$
\begin{array}{rcl}
(q, g, e, q', g') \in T \upharpoonright_< & \Leftrightarrow & \text{pour tout } k \in \mathbb{N}, \text{ et } \delta \in \mathbb{T} \text{ tel que } \delta \leq k \text{ et } (q, g, \delta, q + \delta) \in T, \\
 & & \quad \forall e',\ (q + \delta, g + \delta, e', q'') \in T \Longrightarrow e \not<_k e' \\
 & \Leftrightarrow & \forall e',\ \forall ((g', \gamma'), e', (a', R')) \in M \text{ tel que } \exists k,\ e <_k e', \\
 & & \quad (q, g) \notin [\![g' \wedge \diamond_k(\gamma')]\!] \\
 & \Leftrightarrow & \forall e',\ r = \left|\{t_i | t_i = ((g_i, \gamma_i), e', (a_i, R_i)) \in M \wedge \exists k,\ e <_k e'\}\right|, \\
 & & \quad \forall T, g \subseteq [1, r], T \cap F = \emptyset, T \cup F = [1, r],\ T \neq \emptyset \\
 & & \quad\quad (q, g) \notin \bigcup [\![g_F^T \wedge \bigwedge_{i \in T} \diamond_k \gamma_i]\!] \\
 & \Leftrightarrow & \forall e', r = \left|\{t_i | t_i = ((g_i, \gamma_i), e', (a_i, R_i)) \in M \wedge \exists k,\ e <_k e'\}\right|, \\
 & & \quad \begin{cases} \text{soit } (q, g) \in [\![g^{\emptyset}_{[1,r]} \wedge g]\!] \\ \text{soit } \exists T, F \subseteq [1, r], T \cap F = \emptyset, T \cup F = [1, r],\ T \neq \emptyset \\ \quad\quad (q, g) \in [\![g_F^T \wedge \gamma_T^F]\!] \end{cases} \\
 & \Leftrightarrow & \forall e', r = \left|\{t_i | t_i = ((g_i, \gamma_i), e', (a_i, R_i)) \in M \wedge \exists k,\ e <_k e'\}\right|, \\
 & & \quad \exists T, F \subseteq [1, r], T \cap F = \emptyset, T \cup F = [1, r], \\
 & & \quad (q, g, e, q', g') \in [\![t_T^F]\!] \\
 & \Leftrightarrow & (q, g, e, q', g') \in T'
\end{array}
$$

□

**La vivacité.** Les auteurs de [BGS00] ont une approche différente puisqu'ils cherchent à construire des *systèmes vivants*. Un automate temporisé est *vivant* (*live*) si toute exécution finie peut être étendue en une exécution infinie, c'est-à-dire la suite de temps associée à l'exécution diverge. Cela revient à dire que rien ne bloque le système. Ainsi par exemple, dans un système vivant, deux sous-systèmes partageant une même ressource peuvent y accéder chacun infiniment souvent. Pour cela,

les auteurs de [BGS00] ajoutent une condition de transitivité $e_1 <_k e_2 \wedge e_2 <_l e_3 \Longrightarrow e_1 <_{k+l} e_3$ dans la Définition 25 des priorités temporelles.

Dans Timed AltaRica, les priorités temporelles contraignent le composant initial, elles n'ont donc pas pour objectif de construire des systèmes vivants. On a quand même un résultat positif, si un composant temporisé est vivant alors le fait de lui appliquer une relation de priorité temporelle quelconque, c'est-à-dire avec des événements urgents ou non, conserve cette propriété.

**Proposition 4 (Conservation de la vivacité)** *Soit $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ un composant temporisé vivant et $<$ une relation de priorité temporelle, alors $\mathcal{A} \upharpoonright_<$ est vivant.*

**Preuve de la proposition 4**. *On pose $[\![\mathcal{A}]\!] = (S_t, F_t, E_t, \pi, T)$, alors une configuration $(q, g)$ est non bloquante si*

$$\begin{cases} \textit{soit } \forall t \in \mathbb{T}, (q, g, t, q+t, g+t) \in T \\ \textit{soit } \exists t \in \mathbb{T}, \exists e \in E,\ (q, g, t, q+t, g+t) \in T,\ (q+t, g+t, e, q', g') \in T \end{cases}$$

*On montre que toute configuration $(q, g)$ de $[\![\mathcal{A}]\!] \upharpoonright_<$ est non bloquante. Par hypothèse, $(q, g)$ est non bloquante dans $[\![\mathcal{A}]\!]$ donc :*

1. *si $\forall t \in \mathbb{T}, (q, g, t, q+t, g+t) \in T$, soit $\forall t \in \mathbb{T}, (q, g, t, q+t, g+t) \in T \upharpoonright_<$ et $(q, g)$ est non bloquante; soit $\exists t \in \mathbb{T}, (q, g, t, q+t, g+t) \notin T \upharpoonright_<$, alors $\exists \delta < t, \exists e > time$, tels que $(q, g, \delta, q+\delta, g+\delta) \in T$ et $(q+\delta, g+\delta, e, q', g') \in T$. On choisit $\delta$ de sorte que $e$ soit le premier événement urgent tirable ainsi $(q, g, \delta, q+\delta, g+\delta) \in T \upharpoonright_<$. Ensuite, si $(q+\delta, g+\delta, e, q', g') \in T \upharpoonright_<$ on a $(q, g)$ est non bloquante. Sinon, si $(q+\delta, g+\delta, e, q', g') \notin T \upharpoonright_<$, cela signifie qu'il existe $e'$ tel que $e <_k e'$, donc d'après l'item 4, $e' > e$, d'après transitivité de la relation d'ordre $<$ et d'après l'item 2 de la définition 25 page 73, on a $e'$ urgent et tirable en $(q+\delta, g+\delta)$. On prend alors un plus grand élément urgent $e'$ comparable à $e$ et tirable en $(q+\delta, g+\delta)$, alors $(q+\delta, g+\delta, e', q'', g'') \in T \upharpoonright_<$, i.e. $(q, g)$ non bloquante.*

2. *si $\exists t \in \mathbb{T}, \exists e \in E,\ (q, g, t, q+t, g+t) \in T,\ (q+t, g+t, e, q', g') \in T$, soit $(q, g, t, q+t, g+t) \in T \upharpoonright_<$ et $(q+t, g+t, e, q', g') \in T \upharpoonright_<$ alors $(q, g)$ est non bloquante ; soit $(q, g, t, q+t, g+t) \notin T \upharpoonright_<$ et on se retrouve dans le cas précédent ; soit $(q, g, t, q+t, g+t) \in T \upharpoonright_<$ et $(q+t, g+t, e, q', g') \notin T \upharpoonright_<$, alors $\exists e', e <_k e',\ \exists \delta \leq k,\ (q+t, g+t, \delta, q+t+\delta, g+t+\delta) \in T,\ (q+t+\delta, g+t+\delta, e', q'', g'') \in T$.*

   *On remarque que la relation de priorité vérifie localement la condition de transitivité : si $e <_k e'$ et $e' <_l e''$, si en une configuration $(q, g)$ $e$ est tirable, $e'$ est tirable dans $t \leq k$ unités de temps et $e''$ est tirable dans $t + t' \leq k + l$ unités de temps, alors seules les transitions $(q, g, t+t', q+t+t', g+t+t')$ puis $(q+t+t', g+t+t', e'', q', g')$ sont dans $T \upharpoonright_<$. Ainsi, en toute configuration discrète $(s, f)$ telle que $e$, $e'$ et $e''$ sont tirables, on a $e <_{l+k} e''$.*

   *Fort de cette remarque, on choisit un élément maximal, local en $(q, g)$, $e'$ tel que $e <_k e'$ et $e'$ tirable en $(q+t+\delta, g+t+\delta)$ avec $\delta \leq k$. Alors, soit $(q+t, g+t, \delta, q+t+\delta, g+t+\delta) \notin T \upharpoonright_<$ et on se retrouve dans le cas 1, soit $(q+t, g+t, \delta, q+t+\delta, g+t+\delta) \notin T \upharpoonright_<$ et nécessairement $(q+t+\delta, g+t+\delta, e', q'', g'') \in T \upharpoonright_<$, ce qui entraîne que $(q, g)$ est non bloquante.*

□

Nous avons introduit les priorités temporelles dans Timed AltaRica, nous avons montré comment les encoder syntaxiquement dans un composant temporisé ou comment appliquer des opérateurs de restriction sur la sémantique du composant temporisé. Dans la prochaine partie, nous présentons les nœuds temporisés et la manière d'étendre les notions de priorité temporelle.

## 4.3 La hiérarchie dans Timed AltaRica

Dans cette partie, nous définissons formellement un modèle général Timed AltaRica. Nous étendons la notion de nœud AltaRica, Définition 6 page 32, en nœud temporisé. On introduit donc des variables de type horloges, des sous-nœuds temporisés et des relations de priorité temporelle dans un nœud AltaRica. Nous définissons d'abord la sémantique des nœuds temporisés. Nous étudions l'effet combiné des priorités temporelles et des broadcast. Enfin, nous prouvons le théorème clé de la hiérarchie : tout nœud Timed AltaRica peut se réécrire en un composant temporisé bisimilaire temporellement.

### 4.3.1 Les nœuds Timed AltaRica

#### Syntaxe des nœuds temporisés

Les nœuds sont de simples extensions temporisées des nœuds AltaRica. Leur syntaxe générale est représentée dans la Figure 4.17.

```
node Nœud
 flow   f : type ...; y : clock ...
 event  e,h ...
 state  s : type ...,; x : clock ...
 sub C: Composant; ...
 trans  formule(s,f,C.f) & contrainte(x,y,C.y) |- e -> s:= expression;
      ...
 sync <e,C.h ...>,
     <h,C.e? ...> ...
      ...
 assert formule(s,f,C.f);
 init   formule (s,f,x,C.x);
 tinvariant formule(s,f,C.f) => contrainte_convexe(x,y,C.y);
edon
```

FIG. 4.17 – Syntaxe d'un nœud Timed AltaRica

**Exemple 24 (Un système de réservoir critique - III)** *On reprend le composant temporisé* Reservoir *spécifié dans la Figure 4.2. On imagine à présent un nœud* Pompe, *Figure 4.18, qui administre deux réservoirs et a accès à la quantité de chaque réservoir grâce aux variables de flux $R_i.Y$. Initialement, le premier réservoir fournit l'essence et ce instantanément d'après la condition d'urgence* marche>time *et la condition initiale $res = 0$.*

*Un fonctionnement sans panne est le suivant : le premier réservoir* R1 *se vide peu à peu et le nœud* Pompe *vérifie sa contenance toutes les* 10 *unités de temps avec l'événement* ok. *Si la quantité devient trop faible, c'est-à-dire $R1.Y \geq 90$, le nœud* Pompe *fait basculer la source du carburant sur le deuxième réservoir. Pour cela, il réalise la transition* chgt *qui modifie les variables de flux $R_i.R$ et alors l'événement* marche *devient tirable. D'après la condition d'urgence, les événements marche se synchronisent entre* Pompe *et* R2 *immédiatement. Enfin, lorsque les deux réservoirs sont vides, le nœud* Pompe *réalise immédiatement la transition* vide *et atteint la configuration $res = 2$.*

*En cas de panne de* R1, *le nœud* Pompe *en est informé en moins de* 10 *u.t. par la formule $R_1.Y > 100$, alors* Pompe *bascule sur* R2 *et le deuxième réservoir fournit instantanément le carburant nécessaire pour le vol de l'avion. Il se peut que* R2 *soit déjà en panne, le nœud* Pompe *détecte une fuite par une différence entre l'horloge locale d'un réservoir représentant la quantité et l'horloge*

```
node Pompe
 state res : [0,2]; z,t:clock;
 event {marche,vide} > time, ok, chgt
 sub R1, R2 : Reservoir;
 trans res=0 & R1.Y < 90 & t=R1.Y & z=10 |- ok -> z:=0;
    res=0 & R1.Y >= 90 |- chgt -> res:=1;
    |- marche ->;
    res <2 & (R1.Y!=t | R1.Y>=100) & (R2.Y!= t | R2.Y>=100) |- vide -> res:=2
 sync <marche, R1.marche?,R2.marche?> >=1 ;
 assert (res=0) => (R1.R=vrai & R2.R=faux);
     (res>0) => (R1.R=faux & R2.R=vrai);
 tinvariant (res=0) => (R1.Y <=100 & z<=10)
 init R1.s:=0,R2.s:=0,t:=0,res:=0,R1.x:=0,R2.x:=0,z:=0
edon
```

FIG. 4.18 – Système de gestion du carburant

*globale* $R_2.Y! = t$. *En effet, en cas de fuite, l'horloge locale d'un réservoir est mise à jour à* 101 *et donc* $R_i.x > t$.

*La* Pompe *est dans une configuration du type* $res = 2$ *lorsque tous les réservoirs sont vides. Un scénario pour atteindre* $res = 2$ *est* R1 *se vide normalement,* R2 *est sollicité à* $t = 90$ *et* R2 *a déjà subi une panne.*

### Sémantique des nœuds Timed AltaRica

Un nœud Timed AltaRica permet d'associer des composants temporisés et/ou des nœuds temporisés. Nous avons exprimé la syntaxe générale dans la Figure 4.17. Formellement, un nœud temporisé est :

**Définition 27 (Nœud temporisé - syntaxe abstraite [CPR02])** *Un* nœud temporisé *est un uplet* $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ *avec :*

1. $V_F$ *est un ensemble de variables de flux et* $C_F$ *est l'ensemble des horloges de flux ;*
2. $E = E_+ \cup \{\epsilon\}$ *est un ensemble fini d'événements et* $\epsilon \notin E_+$ *est l'action inobservable ;*
3. $<$ *est une relation de priorité temporelle sur* $E$ *;*
4. *pour tout* $i \in [1, n]$, $\mathcal{N}_i$ *est un composant temporisé ou un nœud temporisé,* $V_{F_i}$ *(resp.* $C_{F_i}$*) est l'ensemble des variables (resp. d'horloges) de flux de* $\mathcal{N}_i$ *et* $E_i$ *est l'ensemble des événements. On suppose* $\forall i \neq j \in [1, n], V_{F_i} \cap V_{F_j} = C_{F_i} \cap C_{F_j} = \emptyset$ *;*
5. $\mathcal{N}_0$ *est un composant temporisé spécial appelé le* contrôleur, *il s'agit du comportement local du nœud. L'ensemble des événements de ce nœud est* $E_0 = E$ *muni de la relation de priorité vide. L'ensemble des variables de flux de* $\mathcal{N}_0$ *est* $F_0 = F \cup F_1 \cup \cdots \cup F_n$*, et l'ensemble des horloges de flux est* $C_{F_0} = C_F \cup F_1 \cup \cdots \cup C_{F_n}$ *;*
6. $\tilde{V} \subseteq E_0 \times E_1 \times \cdots \times E_n$ *est une expansion de l'ensemble des vecteurs de synchronisation avec la relation de priorité* $<_{\tilde{V}}$ *(cette relation exprime l'ordre induit par les broadcast, cf. paragraphe page 31 et Remarque 1 page 32).*

On peut enfin donner la sémantique d'un nœud temporisé sous forme d'un STTI. On remarquera que l'on n'est pas obligé d'exprimer les comportements continus intermédiaires, i.e. *on peut rester un délai* $\delta$ *dans une configuration si, et seulement si tous les délais intermédiaires peuvent s'écouler en cette configuration.* Cela découle directement de la sémantique des composants temporisés.

**Définition 28 (Nœud temporisé - sémantique [CPR02])** *Soit un nœud temporisé* $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$. *On suppose que pour tout* $i \in [0,n]$, $[\![\mathcal{N}_i]\!] = \langle E_{t,i}, F_{t,i}, S_{t,i}, \pi_i, T_i\rangle$ *de dimension* $(n_i, m_i)$. *Alors la sémantique de* $\mathcal{N}$ *est le STTI* $[\![\mathcal{N}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$ *de dimension* $(\sum_{k=0}^{n} n_i, |C_F|)$ *défini par :*

1. $F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$ *où* $m = |C_F|$ *;*
2. *pour tout* $q_i \in S_{t,i}$, $i \in [0,n]$, $\pi(q_0, q_1, \cdots, q_n) = \{g \in F_t | \forall i \in [1,n], \exists g_i \in F_{t,i},\ g_i \in \pi_i(q_i)$ *et* $(g, g_1, \cdots, g_n) \in \pi_0(q_0)\}$,
3. $S_t = \{q \in S_{t,0} \times S_{t,1} \times \cdots \times S_{t,n} | \pi(q) \neq \emptyset\}$,
4. $T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ *est défini par :*
   (a) *on considère* $<_0$ *la relation de priorité temporelle engendrée par* $<$ *et définie par :* $(e_0, e_1, \cdots, e_n) <_0 (e'_0, e'_1, \cdots, e'_n) \iff e_0 < e'_0$,
   (b) *soit* $T_{\mathcal{N}}$ *l'ensemble des transitions définies par :*
   – *pas discret : let* $e = (e_0, e_1, \cdots, e_n)$

$$\langle (s_0, s_1, \cdots, s_n), f, e, (s'_0, s'_1, \cdots, s'_n), f'\rangle \in T_{\mathcal{N}} \iff \begin{cases} \exists f_0 = (f, f_1, \cdots, f_n) \in \pi_0(s_0) \\ \exists f'_0 = (f', f'_1, \cdots, f'_n) \in \pi_0(s'_0) \\ \forall i \in [0,n], (s_i, f_i, e_i, s'_i, f'_i) \in T_i \end{cases}$$

   – *pas continu :* $\delta \in \mathbb{T}$

$$\langle (s_0, s_1, \cdots, s_n), f, \delta, (s'_0, s'_1, \cdots, s'_n), f'\rangle \in T_{\mathcal{N}} \iff \begin{cases} \exists f_0 = (f, f_1, \cdots, f_n) \in \pi_0(s_0) \\ \exists f'_0 = (f', f'_1, \cdots, f'_n) \in \pi_0(s'_0) \\ \forall i \in [0,n], (s_i, f_i, \delta, s'_i, f'_i) \in T_i \end{cases}$$

   (c) *alors* $T = \left(T_{\mathcal{N}} \lceil <_{\tilde{V}}\right) \lceil <_0$.

### Broadcast et priorité temporelle

On peut se demander si les priorités induites par les broadcast et les relations de priorité temporelles sont indépendantes. En fait ce n'est pas le cas, il n'y a pas commutativité des opérateurs de restriction $\lceil <_{\tilde{V}}$ et $\lceil <_0$ : l'ordre est important et il faut commencer par appliquer le broadcast. La raison est que $\lceil <_{\tilde{V}}$ regroupe un ensemble de priorités locales et que $<_0$ est un ordre global. Nous illustrons cette non commutativité dans l'exemple suivant.

**Exemple 25** *On considère le nœud temporisé* nœud0 *suivant :*

```
node sous_noeud0
  state s : bool;
  event e
  trans
       |- e ->;
       |- e' ->;
  init s := vrai;
edon
```

```
node noeud0
  state s : bool; x: clock;
  event e (<,5) e'
  trans
        |- e -> x:=0;
     x>= 10 |- e' ->;
  init s := vrai;
  sub C,C' : sous_noeud0;
  sync <e?, C.e, C'.e>;
edon
```

*Les vecteurs de broadcast* $V = \langle e?, e_1, e_2\rangle, \langle e', \epsilon, \epsilon\rangle$ *génèrent l'ensemble de vecteurs* $\tilde{V} = \langle e, e_1, e_2\rangle, \langle \epsilon, e_1, e_2\rangle, \langle e', \epsilon, \epsilon\rangle$. *On en déduit que* $(vrai, x = 10, \langle e, e_1, e_2\rangle, vrai, 0), (vrai, x = 10, \langle \epsilon, e_1, e_2\rangle, vrai, 10), (vrai, x = 10, \langle e', \epsilon, \epsilon\rangle, vrai, 10)$ *appartiennent à* $T_{\mathcal{N}}$.

*Alors,* $T_{\mathcal{N}} \restriction <_{\tilde{V}}$ *ne contient plus que* $(vrai, x = 10, \langle e, e_1, e_2\rangle, vrai, 0), (vrai, x = 10, \langle e', \epsilon, \epsilon\rangle, vrai, 10)$ *d'après la définition de* $<_{\tilde{V}}$ *donnée dans la Remarque 1 page 32. Finalement, seule la transition* $(vrai, x = 10, \langle e', \epsilon, \epsilon\rangle, vrai, 10)$ *appartient* $T = \left(T_{\mathcal{N}} \restriction <_{\tilde{V}}\right) \restriction <_0$.

*Si on applique les opérateurs de restriction de priorité dans l'autre sens,* $T_{\mathcal{N}} \restriction <_0$ *ne contient que* $(vrai, x = 10, \langle \epsilon, e_1, e_2\rangle, vrai, 10), (vrai, x = 10, \langle e', \epsilon, \epsilon\rangle, vrai, 10)$ *d'après la définition de* $<_0$. *Ensuite, ces deux vecteurs appartiennent à* $\left(T_{\mathcal{N}} \restriction <_0\right) \restriction <_{\tilde{V}}$*, en effet deux vecteurs sont comparables pour* $<_{\tilde{V}}$ *si, et seulement si* $\exists a \in I$*, tel que ces deux vecteurs appartiennent à* $Vec(a)$.

*Alors,* $\left(T_{\mathcal{N}} \restriction <_{\tilde{V}}\right) \restriction <_0 \subsetneq \left(T_{\mathcal{N}} \restriction <_0\right) \restriction <_{\tilde{V}}$.

### 4.3.2 Modularité de Timed AltaRica

#### Compositionnalité et réutilisabilité

Cette partie porte sur les aspects comportementaux des nœuds temporisés : si on considère $n$ composants temporisés bisimilaires respectivement à $n$ autres, alors le produit synchronisé des $n$ premiers composants temporisés reste bisimilaire au produit synchronisé des $n$ seconds, avec la même contrainte de synchronisation. Ainsi, si on souhaite remplacer un sous composant par un composant optimisé ayant le comportement attendu alors le nouveau nœud est toujours correct. Formellement, cela signifie que la bisimulation temporisée interfacée est stable par compositionnalité.

**Théorème 5 (Stabilité de la bisimulation temporisée interfacée par composition [CPR02])** *Soient* $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ *et* $\mathcal{N}' = \langle V_F, C_F, E, <, \mathcal{N}'_0, \cdots, \mathcal{N}'_n, (\tilde{V}, <_{\tilde{V}})\rangle$ *deux nœuds temporisés tels que* $\forall i \in [0..n]$ *il existe un homomorphisme de bisimulation temporisée interfacée* $h_i$ *de* $[\![\mathcal{N}_i]\!]$ *dans* $[\![\mathcal{N}'_i]\!]$*, alors il existe un homomorphisme de bisimulation temporisée interfacée* $h$ *de* $[\![\mathcal{N}]\!]$ *dans* $[\![\mathcal{N}']\!]$.

**Preuve du théorème 5**. *On considère* $\forall i \in [0, n], [\![\mathcal{N}_i]\!] = \langle E_t, F_{t,i}, S_{t,i}, \pi_i, T_i\rangle$ *et* $\forall i \in [0, n], [\![\mathcal{N}'_i]\!] = \langle E_t, F_{t,i}, S'_{t,i}, \pi'_i, T'_i\rangle$ *les STTI définis par la sémantique des nœuds temporisés. Pour les nœuds temporisés* $\mathcal{N}$ *et* $\mathcal{N}'$*, on utilise le théorème 4 page 76 qui permet de faire abstraction des priorités. Ainsi, pour* $[\![\mathcal{N}]\!] = (\langle E_t, F_t, S_t, \pi, T\rangle \restriction <_{\tilde{V}}) \restriction <_0 = (\mathcal{A} \restriction <_{\tilde{V}}) \restriction <_0$ *et* $[\![\mathcal{N}']\!] = (\langle E_t, F_t, S'_t, \pi', T'\rangle \restriction <_{\tilde{V}}) \restriction <_0 = (\mathcal{A}' \restriction <_{\tilde{V}}) \restriction <_0$*, nous montrons que* $\mathcal{A}$ *et* $\mathcal{A}'$ *sont bisimilaires temporellement.*

*Puisque les nœuds* $\mathcal{N}_i$ *and* $\mathcal{N}'_i$ *sont bisimilaires temporellement, on considère* $\forall i \in [0, n]$*, les homomorphismes de bisimulation temporisée interfacée* $h_i : [\![\mathcal{N}_i]\!] \longrightarrow [\![\mathcal{N}'_i]\!]$. *On pose* $h : S_t \longrightarrow S'_t$ *défini par* $h(\langle q_0, \cdots, q_n\rangle) = \langle h_0(q_0), \cdots, h_n(q_n)\rangle$. *On montre que* $h$ *est un homomorphisme de bisimulation temporisée interfacée de* $\mathcal{A}$ *dans* $\mathcal{A}'$.

1. *on montre d'abord que* $h$ *est surjective. Si* $q' = (q'_0, \cdots, q'_n) \in S'_t$*, on sait puisque chaque* $h_i$ *est surjectif que* $\exists q_0, \cdots q_n \in S_{0,t} \times \cdots \times S_{n,t}, q' = \langle h_0(q_0), \cdots, h_n(q_n)\rangle = h(\langle q_0, \cdots, q_n\rangle)$.
2. *ensuite, soit* $q \in S_t$*, on montre que* $\pi(q) = \pi'(h(q))$.
$$\begin{aligned} \pi(q) &= \pi_0(q_0) \wedge \cdots \wedge \pi_n(q_n) \\ &= \pi'_0(h_0(q_0)) \wedge \cdots \wedge \pi'_n(h_n(q_n)) \\ &= \pi'(h(q)) \end{aligned}$$
3. *Soit* $(q, g, e, q_1, g') \in T$*, comme il n'y a pas de priorité temporelle, on a naturellement* $(h(q), g, e, h(q_1), g') \in T'$.
4. *Soit* $q \in S_t, q'_1 \in S'_t$ *avec* $(h(q), g, e, q'_1, g') \in T'$. *Alors* $\exists q_1 \in S_t, q'_1 = h(q_1)$ *tel que* $\forall i \in [0, n], (q_i, g_i, e_i, q_{1,i}, g'_i) \in T_i$*,* $g_0 = (g, g_1, \cdots, g_n)$ *et* $g'_0 = (g, g'_1, \cdots, g'_n)$ *puisque chaque* $h_i$ *satisfait le point 4 de la définition 17. On conclut que* $(q, g, e, q_1, g') \in T$.

*On a montré que $h$ est un homomorphisme de bisimulation temporisée interfacée de $\mathcal{A}$ dans $\mathcal{A}'$. D'après le théorème 4 page 76 qui assure la conservation de l'homomorphisme en appliquant les opérateurs de restriction de priorité, on déduit que $h$ est un homomorphisme de bisimulation temporisée interfacée de $(\mathcal{A} \lceil<_{\tilde{V}}) \lceil<_0$ dans $(\mathcal{A}' \lceil<_{\tilde{V}'}) \lceil<'_0$, i.e. de $\mathcal{N}$ dans $\mathcal{N}'$.* □

**Réécriture d'un nœud temporisé en un composant temporisé**

Dans cette partie nous montrons que tous les opérateurs que nous avons rajoutés sont des artifices syntaxiques et tout nœud temporisé peut être remplacé par un composant temporisé ayant les mêmes comportements. Nous donnons dans un premier temps la procédure de construction d'un composant temporisé équivalent à un nœud temporisé. Ensuite, nous prouvons l'équivalence de leur sémantique.

**Définition 29 (Sémantique temporisée symbolique [CPR02])** *Si $\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ est un nœud temporisé, on note $\mathcal{C}_{\mathcal{N}} = \langle V_S, C_S, V_F, C_F, E, A, M\rangle$ le composant temporisé construit comme suit :*

1. $\forall i = 0 \cdots n$,
   - (a) *si $\mathcal{N}_i$ est un composant temporisé, alors on pose $\mathcal{N}'_i = \mathcal{N}_i \lceil<^i$ (cf résolution syntaxique de la partie 4.2.3 page 77) ;*
   - (b) *si $\mathcal{N}_i$ est un nœud temporisé, alors on définit $\mathcal{N}'_i = \mathcal{C}_{\mathcal{N}i}$ ;*
   - (c) *on note $\mathcal{N}'_i = \langle V'_{S_i}, C'_{S_i}, V'_{F_i}, C'_{F_i}, E'_i, A'_i, M'_i, \emptyset\rangle$ ;*
2. $V_S = V'_{S_0} \cup \cdots \cup V'_{S_n}$ *et* $C_S = C'_{S_0} \cup \cdots \cup C'_{S_n}$ ;
3. $A = (\exists_{i=1..n}(V'_{F_i} \cup C'_{F_i})). \bigwedge_{i=0..n} A'_i$ ;
   *où la notation $\exists_{i=1..n}(W_i).\phi$ signifie $\forall i, \exists \eta_i \in W_i, \phi(\eta_i)$. Alors pour $((s,\nu),(f,\mu)) \in \mathcal{D}^{V_S} \times \mathbb{R}^{C_S} \times \mathcal{D}^{V_F} \times \mathbb{R}^{C_F}$ on définit :*
   - $((s,\nu),(f,\mu)) \in [\![A]\!] \iff \forall i \in [1..n], \exists \eta_i \in \mathcal{D}^{V'_{F_i}} \times \mathbb{R}^{C'_{F_i}}$ *tel que* $((s,\nu),(f,\mu),\eta_1,\cdots,\eta_n)) \in [\![\bigwedge_{i=0..n} A'_i]\!]$,
   - $\forall i \in [1..n], ((s,\nu),(f,\mu),\eta_1,\cdots,\eta_n)) \in [\![A'_i]\!] \iff ((s_i,\nu_i),\eta_i) \in [\![A'_i]\!]$.
4. *l'ensemble des macro-transitions $M \subseteq (\mathbb{F} \times \mathcal{B}(C_T)) \times E \times (\mathbb{E}(V_T)^{V_T} \times \mathcal{A}(C_T))$ est défini par $M = (M' \lceil<_{\tilde{V}}) \lceil<_0$, où $<_0$ est la relation de priorité temporelle donnée à la définition 28 page 82, avec $M'$ défini par $((g,\gamma),e,(a,R)) \in M'$ si, et seulement si :*
   - $\forall i \in [0 \cdots n], \exists((g_i,\gamma_i),e_i,(a_i,R_i)) \in M'_i, g = (\exists_{i=1..n}V_{F_i}).g_0 \wedge \cdots \wedge g_n$, *et* $\gamma = (\exists_{i=1..n}C_{F_i}) .\gamma_0 \wedge \cdots \wedge \gamma_n$,
   - $\forall x \in V_S, x \in V'_{S_i} \implies a(x) = a_i(x)$ *and* $\forall c \in C_S, c \in C'_{S_i} \implies R(c) = R_i(c)$.
   - $e = (e_0, e_1, \cdots, e_n) \in \tilde{V}$.

**Théorème 6 (Réécriture en composant temporisé [CPR02])** *Soit $\mathcal{N}$ un nœud temporisé. Alors $\mathcal{N}$ peut se réécrire en composant temporisé $\mathcal{C}_{\mathcal{N}}$ bisimilaire temporellement.*

**Preuve du théorème 6**. *Soit $\mathcal{N} = \langle V_F \cup C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ un nœud temporisé. On note $\mathcal{C}_{\mathcal{N}} = \langle V_S \cup C_S, V_F \cup C_F, E, A, M, <\rangle$ sa réécriture en composant.*

**On suppose que tous les $\mathcal{N}_i$ sont des composants temporisés.** *On considère les STTI $[\![\mathcal{N}]\!] = \langle E_t, F_t, S^0_t, \pi^0, T^0\rangle$ et $[\![\mathcal{C}_{\mathcal{N}}]\!] = \langle E_t, F_t, S^1_t, \pi^1, T^1\rangle$ qui donnent les sémantiques de $\mathcal{N}$ et $\mathcal{C}_{\mathcal{N}}$.*

*On vérifie d'abord que $S^1_t = S^0_t$ et donc que $\pi^1 = \pi^0$. D'après la définition 28 page 82, on a $S^0_t = \{q \in S_{0_t} \times S_{1_t} \cdots \times S_{n_t}, \pi^0(q) \neq \emptyset\}$ avec $\pi^0(q_0, q_1, \cdots, q_n) = \{g \in F_t | \forall i \in [1,n], \exists g_i \in F_{t,i},\ g_i \in$*

*$\pi_i(q_i)$ et $(g, g_1, \cdots, g_n) \in \pi_0(q_0)\}$. De plus, toujours d'après la définition de la sémantique du nœud, on a : $\pi_i(q_i) = \{g_i \in F_{t,i} | (q_i, g_i) \in [\![A'_i]\!]\}$ et $\pi_0(q_0) = \{(g, \eta_1, \cdots, \eta_n) | (q_0, g, \eta_1, \cdots, \eta_n) \in [\![A'_0]\!]\}$. On en déduit :*

$$\begin{aligned}
\pi^0(q) &= \{g \in F_t | \forall i \in [1, n], \exists \eta_i \in F_{t,i}, (q_i, \eta_i) \in [\![A'_i]\!] \text{ et } (q, g, \eta_1, \cdots, \eta_n) \in [\![A'_0]\!]\} \\
&= \{g \in F_t | \forall i \in [1, n], (q, g) \in [\![\exists (V'_{F_i} \cup C'_{F_i}).A'_i]\!] \text{ et } (q, g) \in \\
&\quad [\![\exists_{i=1..n} (V'_{F_i} \cup C'_{F_i}).A'_0]\!]\} \\
&= \pi^1(q)
\end{aligned}$$

*D'où $\pi^1 = \pi^0$ et $S^0_t = S^1_t$.*

*On prouve ensuite que $T^0 = T^1$. D'après le théorème 4 page 76 de préservation de la bisimulation par opérateur de restriction de priorité, il est suffisant de montrer le résultat pour la relation $T_{\mathcal{N}}$ explicitée dans la définition 28 page 82 est la même que la relation de transitions $M'$ donnée dans la définition 29 page 84.*

*On prouve que $T_{\mathcal{N}} \subseteq M'$. Soit $\langle (q_0, q_1, \cdots, q_n), f, e, (q'_0, q'_1, \cdots, q'_n), f' \rangle \in T^0$. Il y a deux cas :*

1. *$e = (e_0, e_1, \cdots, e_n)$, la transition est un pas discret. Par définition 28, $\exists f_0 = (f, f_1, \cdots, f_n) \in \pi_0(q_0)$, $\exists f'_0 = (f', f'_1, \cdots, f'_n) \in \pi_0(q'_0)$, $\forall i \in [0..n], (q_i, f_i, e_i, q'_i, f'_i) \in T_i$. Cela entraîne $\forall i \in [0..n], \exists ((g_i, \gamma_i), e_i, (a_i, R_i)) \in M'_i$ tel que $(q_i, f_i) \in [\![A'_i \wedge g_i \wedge \gamma_i]\!]$ et $q'_i = (a_i, R_i)(q_i, f_i)$. On considère alors la transition $((g, \gamma), e, (a, R))$ créée dans $\mathcal{C}_{\mathcal{N}}$ définie par : $g = (\exists_{i=1..n} V_{F_i}).g_0 \wedge \cdots \wedge g_n$, $\gamma = (\exists_{i=1..n} C_{F_i}).\gamma_0 \wedge \cdots \wedge \gamma_n$, $e = (e_0, e_1, \cdots, e_n)$, $(a, R)$ est donné dans la définition 29. Il est clair que $(q, f, f_1, \cdots, f_n) \in [\![A]\!]$. Il reste à montrer que $(q, f, f_1, \cdots, f_n) \in [\![g \wedge \gamma]\!]$. Comme $\forall i \in [1..n], (q_i, f_i) \in [\![g_i \wedge \gamma_i]\!]$ et $(q, f, f_1, \cdots, f_n) \in [\![A'_0]\!]$, $(q, f) \in [\![g = (\exists_{i=1..n} V_{F_i}).g_0 \wedge \cdots \wedge g_n]\!]$. On fait de même pour $\gamma$. Comme $(a, R)$ coïncide avec $a_i, R_i$ sur les ensembles $V'_{S_i} \cup C'_{S_i}$, les nouveaux états $q''_i$ coïncident avec les $q'_i$. De plus, $(e_0, e_1, \cdots, e_n) \in \tilde{V}$, donc est aussi dans les événements de $M'$.*
2. *$e = \delta \in \mathbb{T}$, le temps progresse. La preuve est à nouveau la même : les mises à jour sont simplement des mises à jour temporelles où les valeurs des horloges sont incrémentées de $\delta$ unités de temps.*

*Réciproquement $M' \subseteq T_{\mathcal{N}}$ se montre sensiblement de la même manière.*

**Cas général.** *L'hypothèse d'induction est que tous les sous nœuds temporisés $\mathcal{N}_i$ peuvent se réécrire en composants temporisés $\mathcal{N}'_i$ : on applique alors le théorème 5 page 83 puis le cas précédent et on obtient le résultat.* □

**Deuxième procédure de construction de $\mathcal{A} \upharpoonright_{\mathcal{U}_g(E)}$.** On propose une deuxième solution en sus de celle donnée dans la section 4.2.2 page 68 basée sur l'encapsulation d'un composant temporisé contenant des événements urgents. On considère un composant temporisé $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ avec $\mathcal{U}_g(E)$ l'ensemble des événements urgents.

On a déjà vu qu'il fallait scinder les configurations urgentes en 3, on a pour cela introduit une variable de flot discrète $U$. De manière équivalente, on peut modifier l'invariant du composant en $g_u \implies \big((\gamma_u \wedge \neg \uparrow \gamma_u) \implies Y = 0\big)$ pour toute configuration urgente $g_u$. Ensuite l'horloge $Y$ est une horloge globale qui est resettée pour toutes les transitions possibles, même les $\varepsilon$-transitions. On remarque que pour cette méthode, nous sommes obligés de réduire le type de garde accepté sur les transitions urgentes, on ne peut pas traiter les gardes de la forme $x = 5$.

Pour cela, on construit un nœud temporisé $\mathcal{N}_{urgent} = \langle V_F, C_F, E, \mathcal{N}_0, \tilde{A}, \mathcal{C}_u, V \rangle$ :

1. $\mathcal{N}_0 = (\emptyset, \emptyset, V_F, C_F, E, true, \emptyset)$,
2. $\mathcal{C}_u = (\emptyset, \mathcal{C}_u.C_S = \{y\}, \emptyset, \mathcal{C}_u.C_F = \{Y\}, \mathcal{C}_u.E = \{e_u\}, \mathcal{C}_u.A, \mathcal{C}_u.M)$ avec pour assertion $\mathcal{C}_u.A \equiv y = Y$ et unique transition :

   ```
   true |- e_u -> y:=0
   ```
3. $\tilde{\mathcal{A}} = (V_S, C_S, V_F, C_F \cup \{Y\}, E, A \wedge (\bigwedge g_u \wedge Pre_{t_u}([\![A]\!]) \implies (\gamma_u \wedge \neg \uparrow \gamma_u \implies Y = 0)), M)$,
4. les vecteurs de synchronisation sont $V = \forall e \in E,\ < e, \tilde{\mathcal{A}}.e, \mathcal{A}_u.e_u >$ et $< \epsilon, \tilde{\mathcal{A}}.\epsilon, \mathcal{A}_u.e_u >$.

**Exemple 26** *On reprend l'exemple 18 et on encode l'urgence selon le deuxième procédé. On obtient le nœud temporisé $N_{urgent}$ suivant :*

```
node Ex_tilde
state s : [0,1]; x :clock;
flow Y: clock
event e, f, g
trans s=1 & 10 <= x <=20 |- e -> s:=0;
   vrai |- f -> s:=0;
   vrai |- g -> s:=1;
tinvariant s=1 =>(  10 < x <=20 => Y=0)
edon
```

```
node C_u
state y :clock;
flow Y: clock
event e_u
trans
   true |- e_u -> Y:=0;
tinvariant y=Y
edon
```

```
node N_urgent
event e,f,g
sub C: C_u, C2: Ex_tilde
synchro <e,C2.e,C.e_u>,
<f,C2.f,C.e_u>,
<g,C2.g,C.e_u>,
<-,C2.-,C.e_u>
tinvariant C.Y=C2.Y
edon
```

Lorsqu'on réécrit le nœud temporisé $\mathcal{N}_{urgent}$, on retrouve composant temporisé bisimilaire temporellement à celui calculé à l'aide de l'algorithme 4.2 page 77. Les sémantiques ne sont pas égales car l'horloge privée $Y$ n'est pas remise à zéro sur chaque transition. Soit la réécriture syntaxique de $\mathcal{N}$, $\mathcal{C}_\mathcal{N} = (V_S, C_S \cup \{y\}, V_F, C_F, E, A \wedge (\bigwedge g_u \wedge Pre_{t_u}([\![A]\!]) \implies (\gamma_u \wedge \neg \uparrow \gamma_u \implies Y = 0)), M \wedge y := 0)$. On pose $[\![\mathcal{C}_\mathcal{N}]\!] = \langle E_t, F_t, S_\mathcal{N}, \pi_\mathcal{N}, T_\mathcal{N} \rangle$ et $[\![\mathcal{A} \upharpoonright_{\mathcal{U}_g(E)}]\!] = \langle E_t, F_t, S_t, \pi, T \rangle$. On considère la relation définie par $(\overline{s}, y) \simeq \overline{s}$ avec $\overline{s} \in S_t$. On montre que c'est une relation de bisimulation temporisée interfacée :

1. Premièrement $(\overline{s}, 0) \in S_\mathcal{N}$ si, et seulement si $\overline{s} \in S_t$.
2. Ensuite, $\pi_\mathcal{N} = [\![A]\!] \wedge \exists Y.[\![\bigwedge g_u \wedge Pre_{t_u}([\![A]\!]) \implies (\gamma_u \wedge \neg \uparrow \gamma_u \implies Y = 0)]\!] = \pi$.
3. Soit $(\overline{s}, y, \overline{f}, \overline{\mu}, e, \overline{s}', y', \overline{f}', \overline{\mu}') \in T_\mathcal{N}$,
   (a) $e \in E_+$, alors $(\overline{s}, y, \overline{f}, \overline{\mu}, e, \overline{s}', 0, \overline{f}', \overline{\mu}') \in T_\mathcal{N}$ se projette immédiatement sur $T$,
   (b) $e = \varepsilon$ alors soit le vecteur de $V$ est $< \varepsilon, \varepsilon, \varepsilon >$, on a $(\overline{s}, y, \overline{f}, \overline{\mu}, e, \overline{s}, y, \overline{f}', \overline{\mu}') \in T_\mathcal{N}$, soit le vecteur de $V$ est $< \varepsilon, \varepsilon, e_u >$, on a $(\overline{s}, y, \overline{f}, \overline{\mu}, e, \overline{s}, 0, \overline{f}', \overline{\mu}') \in T_\mathcal{N}$. Dans les deux cas les transitions projetées sont dans $T$,
   (c) $e \in \mathbb{T}$, si $e = 0$ c'est évident et si $e = t > 0$, $(\overline{s}, y, \overline{f}, \overline{\mu}, t, \overline{s} + t, y + t, \overline{f}, \overline{\mu} + t) \in T_\mathcal{N}$ alors nécessairement $\forall t' \leq t,\ (\overline{s} + t, \overline{f}, \overline{\mu} + t') \notin [\![\bigwedge g_u \wedge Pre_{t_u}([\![A]\!]) \implies \gamma_u \wedge \neg \uparrow \gamma_u]\!]$. Donc $(\overline{s}, \overline{f}, \overline{\mu}, t, \overline{s} + t, \overline{f}, \overline{\mu} + t) \in T$.
   (d) Le dernier point se prouve de la même manière.

Nous pouvons à présent traiter l'urgence dans n'importe quel nœud temporisé et en particulier dans le nœud *Pompe* présenté dans l'exemple 24 page 80.

**Exemple 27 (Un système de réservoir critique - IV)** *Nous réécrivons syntaxiquement le nœud temporisé* Pompe *en un composant temporisé selon le procédé de réécriture présenté à le Définition 29. Pour cela, nous utilisons l'outil a-semantics [Poi03], la réécriture étant entièrement syntaxique, les domaines des horloges ont simplement été transformés en un intervalle entier et les invariants introduits dans l'assertion. Nous obtenons alors le composant temporisé donné en Annexe B page 139.*

*On peut ensuite appliquer le procédé d'encodage d'urgence sur ce composant temporisé.*

Le langage Timed AltaRica est entièrement défini. Nous nous intéressons dans la suite à déterminer son expressivité et connaître les propriétés décidables qui pourront être vérifiées sur les modèles.

## 4.4 Equivalence Timed AltaRica et automate temporisé

Si le domaine des variables discrètes est fini, le pouvoir d'expression de Timed AltaRica en terme de bisimilarité est exactement celui des automates temporisés. On expose dans cette partie comment passer d'un formalisme à un autre. Puisque le langage Timed AltaRica est hiérarchique et puisque les priorités temporelles peuvent s'encoder syntaxiquement, nous ne raisonnerons que sur des composants, sans sous nœuds et sans priorité.

### 4.4.1 Traduction d'un automate temporisé en un programme Timed AltaRica

Cette traduction est assez naturelle. Les états de contrôle sont encodés par une variable entière. Soit $\mathcal{A} = (Q, E, X, q_0, I, \rightarrow)$ un automate temporisé avec $Q = \{q_0, \cdots, q_n\}$, on construit le composant temporisé $\mathcal{A}_{TAR} = \langle V_S, X, \emptyset, \emptyset, E, A, M\rangle$ comme suit :

1. $V_S = \{s \in [0, n]\}$ est réduit à une variable,
2. l'assertion ne contient que des invariants temporels $A \equiv \bigwedge_i s = i \implies I(q_i)$,
3. la formule initiale est $s = 0$ et toutes les horloges à zéro,
4. pour toute transition $(q_i, g, e, r, q_j) \in\rightarrow$, on a $s = i \wedge g| - e- > s := j, r$ dans $M$.

Ces deux systèmes sont équivalents : à tout $(q_i, \nu)$ on associe $(s = i, \nu)$ et réciproquement. Pour toute transition dans $\mathcal{A}$ issue de $(q_i, \nu)$ étiquetée $e$ menant à $(q_j, r(\nu))$, il en existe une dans $\mathcal{A}_{TAR}$ de $(s = i, \nu)$ étiquetée $e$ dans $(s = j, r(\nu))$. Ces deux systèmes sont bisimilaires temporellement.

**Exemple 28** *Considérons l'exemple de l'automate temporisé donné à la Figure 3.2 page 43. La traduction de cet automate en suivant la procédure précédente donne le composant temporisé suivant, Figure 4.19.*

```
node Exemple
 event   a,b,c;
 state   s :[0,1]; x: clock;
 trans   s=0 & x<1 |- a -> ;
     s=0 & x <= 2 |- b -> s:=1;
     s=1 |- c -> s:=0, x:=0;
 init    x:=0, s:=0
 tinvariant s=0 => x <= 2;
edon
```

FIG. 4.19 – L'Exemple 8 en Timed AltaRica

### 4.4.2 Traduction d'un modèle Timed AltaRica en un automate temporisé

On considère un composant Timed AltaRica $\mathcal{A}_{TAR} = \langle V_S, C_S, V_F, C_F, E, A, M\rangle$ avec $|C_S| = n$ et $|C_F| = m$. Lorsqu'il y a des horloges de flux, le système peut être indéterministe et certaines horloges de flux peuvent être remises à jour aléatoirement dans $\mathbb{R}$ (cf. page 58). Ce cas va engendrer des problèmes pour la procédure de traduction d'un modèle Timed AltaRica en automate temporisé.

Nous donnons dans la suite deux algorithmes : le premier, l'Algorithme 4.3, est basé sur le calcul sémantique du modèle Timed AltaRica, il ne termine pas toujours à cause des horloges de flux ; le deuxième, l'Algorithme 4.4, est symbolique et termine toujours.

Le deuxième Algorithme est une version symbolique du premier : ainsi si l'Algorithme sémantique termine, les automates temporisés obtenus par les deux procédures sont identiques. La représentation graphique résultant de l'Algorithme 4.4 est beaucoup plus concise que celle de l'Algorithme 4.3, En réalité, l'automate temporisé résultant de la Procédure 4.4 est un quotient de l'automate temporisé résultant de la Procédure 4.3.

**Algorithme sémantique : Timed AltaRica → automate temporisé**

On donne une première Procédure de traduction selon l'idée naturelle de combiner le calcul de la sémantique du composant temporisé et le calcul de l'automate temporisé : on associe à chaque configuration discrète un état de contrôle puis on transcrit les transitions. L'Algorithme 4.3 construit l'automate temporisé bisimilaire temporellement en se basant sur le calcul de la sémantique.

**Donnée :** $\mathcal{A}_{TAR} = (V_S, C_S, V_F, C_F, E, A, M)$, on note $p = |V_S| + |V_F|$, les domaines des variables discrètes $D_i$, i.e. $\forall i,\ s_i \in D_i$ et $[\![\mathcal{A}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$
**Sortie :** $\mathcal{A} = (Q, E, X, Q_0, I, \rightarrow)$
**Init :** on pose $Q = D_1 \times \cdots \times D_p \cap [\![A]\!]$\{chaque valuation admissible des variables discrètes devient un état de contrôle\}
  on pose $X = C_S \cup C_F$, $\rightarrow = \emptyset$
  on pose $Q_0 = [\![Init]\!]$\{ensemble des valuations discrètes initiales\}
**pour tout** $q \in Q$ **faire**
  Si $q = (s, f)$, alors $I(q) = \{(\nu, \mu) | (f, \mu) \in \pi(s, \nu)\}$
**fin pour**
**pour tout** $t = ((g, \gamma), e, (a, R)) \in M$ **faire**
  **pour tout** $(s, f) \in Q$ **faire**
    **pour tout** $(s', f') \in Q$ **faire**
      **si** $(s, f, e, s', f') \in T$ **alors**
        $\rightarrow = \rightarrow \cup \{\big((s, f), (\gamma, e, R), (s', f')\big)\}$
      **fin si**
    **fin pour**
  **fin pour**
**fin pour**

**Algorithme 4.3:** Algorithme de traduction sémantique d'un programme Timed AltaRica en automate temporisé

L'Algorithme 4.3 ne termine pas toujours : en effet s'il existe au moins une horloge de flux $Y$ non contrainte par une formule $Y = f(x_1, \cdots, x_n)$ où $x_1, \cdots, x_n$ sont des horloges d'états alors le calcul des invariants et le calcul de la relation de transition sont infinis. Il faut donc introduire des opérations symboliques, ce qui sera fait dans l'Algorithme 4.4.

**Proposition 5** *Etant donné un composant Timed AltaRica $\mathcal{A}_{TAR} = (V_S, C_S, V_F, C_F, E, A, M)$, l'automate temporisé $\mathcal{A} = (Q, E, X, q_0, I, \rightarrow)$ obtenu à partir de l'Algorithme 4.3, s'il termine, est bisimilaire temporellement à $\mathcal{A}_{TAR}$.*

**Preuve de la proposition 5**. *Il suffit d'exprimer la sémantique de l'automate temporisé par un*

*STTI et de vérifier que les STTI sont bisimilaires. On note* $[\![\mathcal{A}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$*. L'automate temporisé obtenu* $\mathcal{A} = (Q, E, X, Q_0, I, \rightarrow)$ *admet pour sémantique le système de transition temporisé* $[\![\mathcal{A}]\!]$.

*Si le composant temporisé ne possède pas de variable de flux, la sémantique de* $\mathcal{A}$ *est également le TITS* $\langle E'_t, F'_t, S'_t, \pi', T'\rangle$ *de dimension* $(|X|, 0)$*, avec* $E'_t = E \cup \mathbb{R}_{\geq 0}$*,* $F_t = \{tt?\}$*,* $S'_t = \{(l, \nu) \,|?\, \nu \in [\![inv(l)]\!]?\}$*,* $\pi(q) = tt$ *et* $T'$ *est donné par* $\rightarrow$*. Alors les deux STTI sont bisimilaires interfacés.*

*Plus généralement, on définit une relation de bisimulation entre un programme Timed AltaRica et un automate temporisé. On associe une configuration à un état de contrôle et une valuation des horloges, i.e. la relation de bisimulation* $R \subseteq (S_t \times F_t) \times (Q \times \mathcal{U}(X))$.

*Par construction, il est évident que si l'Algorithme termine, les deux systèmes seront bisimilaires.* □

### Algorithme symbolique : Timed AltaRica → automate temporisé

On considère un composant Timed AltaRica $\mathcal{A}_{TAR} = \langle V_S, C_S, V_F, C_F, E, A, M\rangle$ avec $|C_S| = n$ et $|C_F| = m$. Dans l'Algorithme 4.3, les problèmes provenaient des invariants et de la relation de transition. On se propose ici un calcul symbolique, à partir de la syntaxe, de ces ensembles :

**Invariants :** regrouper les valuations discrètes vérifiant des mêmes invariants temporels. Pour cela,

1. l'assertion est scindée en deux prédicats : une formule sur les variables discrètes $A_Z$ (correspondant à l'assertion dans AltaRica) et une formule exprimant les invariants temporels $A_C \equiv \bigwedge_{j=1}^{p} P_j \implies I_j$ avec $vlib(P_j) \subseteq Z$.
2. on transforme la formule $A_C$ en $A'_C \equiv \bigwedge_{j=1}^{p} P'_j \implies I'_j$ de sorte que les formules discrètes $P'_j$ soient disjointes deux à deux.

Alors les états de contrôle sont les ensembles de valuations discrètes $[\![P'_j]\!]$ et l'invariant associé est immédiatement $I'_j$.

**Transitions :** les transitions sont énumérées et chaque horloge de flux est automatiquement remise à jour dans $\mathbb{R}$. L'invariant éliminera les valeurs inadmissibles des horloges.

L'Algorithme de traduction est donnée dans la Figure 4.4.

L'Algorithme 4.4 est symbolique donc ne s'appuie que sur la syntaxe d'un composant temporisé et nous obtenons un automate semi interprété. Cette procédure termine toujours puisque toutes les opérations sont réalisées sur des ensembles finis.

On peut rajouter des optimisations par exemple :

**tant que** $\exists q, q' \in Q, q \neq q' \wedge [\![I(q'')]\!] = [\![I(q)]\!]$ **faire**
$q'' = q \vee q'$
$I(q'') = I(q)$
$Q = Q \setminus \{q, q'\} \cup \{q''\}$
**fin tant que**

**Remarque 5** *Soit* $\mathcal{A}_{TAR} = (V_S, C_S, V_F, C_F, E, A, M)$ *un composant Timed AltaRica, l'automate temporisé généré par l'Algorithme 4.3 est grosso modo la sémantique* $[\![\mathcal{A}_{TAR}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$*. On considère la relation d'équivalence sur l'ensemble des valuations discrètes* $\mathcal{D}^Z$ *: soient* $v_1, v_2 \in \mathcal{D}^Z$*,* $v_1 \sim v_2 \iff \big(\forall \mu \in \mathbb{R}^X_{\geq 0},\ (v_1, \mu) \in S_t \times F_t \Leftrightarrow (v_2, \mu) \in S_t \times F_t\big)$*. On peut alors quotienter l'ensemble des valuations par* $\sim$*. Alors* $Q$ *l'ensemble des états de contrôle engendrés par l'Algorithme 4.4 est égal à l'ensemble des classes d'équivalence* $\mathcal{D}^Z/\sim\, = Q$*. D'où l'automate temporisé* $\mathcal{A} = (Q, E, X, q_0, I, \rightarrow)$ *obtenu par l'Algorithme 4.4 est l'automate quotient* $[\![\mathcal{A}_{TAR}]\!]/\sim$.

**Donnée :** $\mathcal{A}_{TAR} = (V_S, C_S, V_F, C_F, E, A, M)$ avec $A_C \equiv \bigwedge_{j=1}^{p} P_j \implies I_j$, $V_F = \{f_1, \cdots, f_l\}$ et $C_F = \{Y_1, \cdots, Y_m\}$
**Sortie :** $\mathcal{A} = (Q, E, X, q_0, I, \rightarrow)$
**Init :** on pose $Q = \emptyset$, $\rightarrow = \emptyset$, $X = C_S \cup C_F$
on pose $Flux = (f_1, \cdots, f_l)$ et $\vec{Y} = (Y_1, \cdots, Y_m)$
**pour tout** $T, F \subseteq [1,p], T \cap F = \emptyset, T \cup F = [1,r]$ **faire**
$l_T^F = A_Z \wedge \left(\bigwedge_{j \in T} P_j\right) \wedge \left(\bigwedge_{j \in F} \neg P_j\right)$ {on construit les états de contrôle}
$Q = Q \cup \{l_T^F\}$
si $T \neq \emptyset$, $I(l_T^F) = \bigwedge_{k \in T} I_k$, sinon $I(l_\emptyset^{[1..p]}) = tt$
**fin pour**
**pour tout** $T, F \subseteq [1,p], T \cap F = \emptyset, T \cup F = [1,r]$ **faire**
$\rightarrow = \rightarrow \cup \bigcup_{v \in \mathcal{D}^{V_F}} \{(l_T^F, (tt, \epsilon, (Flux := v, \vec{Y} \in \mathbb{R}^m)), l_T^F)\}$
**pour tout** $T', F' \subseteq [1,p], T' \cap F' = \emptyset, T' \cup F' = [1,r]$ **faire**
**pour tout** $t = ((g, \gamma), e, (a, R)) \in M$ **faire**
$\rightarrow = \rightarrow \cup \bigcup_{v \in \mathcal{D}^{V_F}} \{(l_T^F, (g \wedge Pre_t(l_{T'}^{F'}) \wedge \gamma, e, (a, R, Flux := v, \vec{Y} \in \mathbb{R}^m)), l_{T'}^{F'})\}$
**fin pour**
**fin pour**
**fin pour**

**Algorithme 4.4:** Algorithme symbolique de traduction d'un programme Timed AltaRica en automate temporisé

**Proposition 6** *Etant donné un composant Timed AltaRica $\mathcal{A}_{TAR} = (V_S, C_S, V_F, C_F, E, A, M)$, l'automate temporisé $\mathcal{A} = (Q, E, X, q_0, I, \rightarrow)$ obtenu à partir de l'Algorithme 4.4 est bisimilaire temporellement.*

**Preuve de la proposition 6**. *On note $[\![\mathcal{A}]\!] = \langle E_t, F_t, S_t, \pi, T \rangle$ et on reprend la relation de bisimulation définie dans la preuve 4.4.2 page 88 : $R \subseteq (S_t \times F_t) \times (Q \times \mathcal{U}(X))$. Soit $(s, \nu, f, \mu) \in S_t \times F_t$ une configuration, on a $(s, f) \in V_S \times V_F$ et alors $\exists G, F$ tels que $(s, f) \in [\![l_G^F]\!]$ par construction de l'automate temporisé, puis par respect des invariants $(\nu, \mu) \in [\![I(l_G^F)]\!]$. Respectivement, si on considère un état de contrôle $q$ et une valuation d'horloge $\nu$, il existe $\exists G, F, q = l_G^F$. On choisit une valuation dans $(s, f) \in q$, $(\nu, \mu) \in I(q)$ et nécessairement $(s, \nu, f, \mu) \in S_t \times F_t$. Ensuite, il reste à montrer que pour toute transition issue d'une valuation de $(s, f) \in S_t \times F_t$ il existe une transition étiquetée par le même événement issu de tout couple $(q, \nu)$ tel que $((s, f), (q, \nu)) \in R$ dont les buts sont également en relation, et réciproquement. Par construction, c'est évident.* □

**Exemple 29 (Un système de réservoir critique - V)** *Nous allons traduire le composant temporisé Reservoir, Figure 4.2 page 57, avec l'Algorithme 4.4. L'invariant temporel*

```
tinvariant (s=1) => (x <= 100);
```

*nous assure qu'il y aura exactement deux états de contrôle puisqu'il y a deux partitions. Le deuxième invariant $(x = Y) \equiv (vrai \Rightarrow x = Y)$ n'influe pas sur les états de contrôle.*

| partition | | état de contrôle | | invariant |
|---|---|---|---|---|
| $T = \emptyset$ | $F = \{1\}$ | $l_T^F \equiv s \neq 1 \equiv s = 0 \wedge s = 2$ | $\equiv q_0$ | $tt$ |
| $T = \{1\}$ | $F = \emptyset$ | $l_T^F \equiv s = 1$ | $\equiv q_1$ | $x \leq 100$ |

*On parcourt ensuite l'ensemble des transitions spécifiées dans le composant et on compare les extrémités (origine et but) aux états de contrôle calculés précédemment. La première transition est*

```
s=0 & R=vrai |- marche -> s:=1, x:=0;
```

*Cette transition* $t_1$ *génère les 4 transitions* $\forall i, j = 0, 1, \big(q_i, (s = 0 \wedge R = vrai \wedge Pre_{t_1}(q_j), a, s := 1, x := 0), q_j\big)$. *Or, la garde assure que l'origine de la transition ne peut être que* $q_0$. *L'affectation modifie* $s$ *et* $x$, *seule* $q_1$ *peut éventuellement être le but et* $Pre_{t_1}(q_1) = \{(s, x, R, Y) | (1, 0, R, Y) \in [\![A]\!]\}$ *puisque la transition modifie toutes les valeurs de manière déterministe, aucune condition initiale n'est requise. On obtient alors la transition* $\big(q_0, (s = 0 \wedge R = vrai, marche, s := 1 \wedge x := 0), q_1\big)$.

*De même les 2 autres transitions* $t_2$ *et* $t_3$ *sont des affectations discrètes déterministes*

```
s=1 & x=100|- arret -> s:=2;
s!=2 |- fuite -> s:=2, x:=101;
```

*donc l'opérateur de précondition n'intervient pas. Les gardes et les affectations déterminent les buts et origines de ces transitions. Nous obtenons ainsi* $\big(q_1, (s = 1 \wedge x = 100, arret, s := 2), q_0\big)$ *à partir de* $t_2$ *et* $\big(q_1, (s! = 2, fuite, s := 2 \wedge x := 101), q_0\big)$ *et* $\big(q_0, (s! = 2, fuite, s := 2 \wedge x := 101), q_0\big)$ *à partir de* $t_3$.

*Enfin, on énumère toutes les transitions* $\epsilon$.

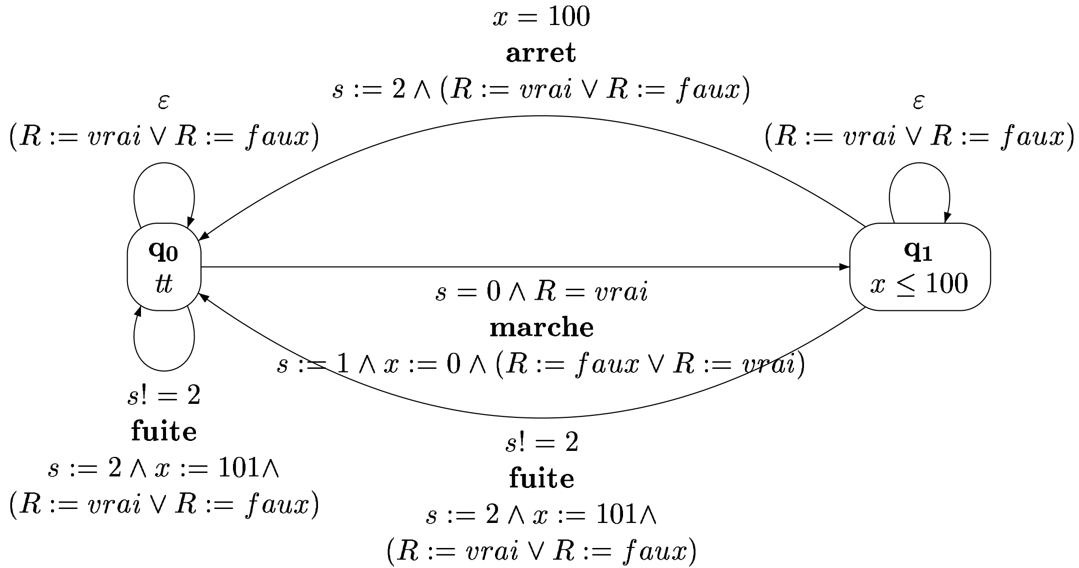



FIG. 4.20 – Automate temporisé du composant temporisé *Reservoir*

*L'automate temporisé obtenu est représenté à la Figure 4.20 et on constate qu'à priori il diffère de celui obtenu à la Figure 4.6 page 63. En fait ils sont parfaitement équivalents en taille puisque la taille de l'automate temporisé obtenu ici est celle des états de contrôle multipliée par les variables discrètes et leur domaine à savoir : 2 (taille de* $R \in \mathbb{B}$*)* $\times$ $\big(2$ *(taille de* $q_0 \equiv s \in [0, 2] \wedge s! = 1$*)* $+ 1$ *(taille de* $q_1$*)*$\big) = 6$ *et une horloge dans les deux cas.*

### 4.4.3 Résultats de décidabilité des modèles Timed AltaRica

Le pouvoir d'expression du langage Timed AltaRica est celui des automates temporisés avec mise à jour [BDFP00] et transitions inobservables [BGP96], cela résulte des deux Algorithmes précédents.

**Proposition 7 (Expressivité de Timed AltaRica)** *Si le domaine des variables discrètes est fini, le pouvoir d'expression de Timed AltaRica (et des automates à contraintes temporisés) est celui des automates temporisés avec mise à jour [BDFP00] et transitions inobservables [BGP96].*

On peut récapituler les résultats de décidabilité pour Timed AltaRica dans le tableau 4.1. Comme dans les cas des automates temporisés [BDFP00] la décidabilité dépend :

1. des gardes des transitions,
2. des affectations des transitions. Dans le Tableau 4.1 les affectations de type $(i)$ correspondent à celles du Tableau 3.1 page 44.

   Or dans Timed AltaRica, il y a les affectations spécifiées des horloges d'états et les affectations non déterministes selon l'assertion des horloges de flux. Il faut donc prendre en compte ce nouveau paramètre : les invariants temporels portant sur les variables de flux reviennent à déterminer le type d'affectation des horloges de flux dans les transitions discrètes, y compris $\epsilon$. Ainsi, soit $Y$ une horloge de flux,

   (a) si $Y$ est toujours égale à une horloge d'état, i.e. $Y = x$, la décidabilité dépend uniquement des affectations sur les transitions discrètes ;

   (b) si $Y = x + c$, nous obtenons le Tableau 4.1. On remarque que des problèmes surgissent dès lors qu'il y a une affection du type $x := y + c$ ;

   (c) la dernière possibilité est que $Y$ satisfait $Y \leq d$ alors les affectations pour $Y$ sont de type 6.

| Type des affectations pour $x$ | | Type des affectations pour $Y$ | |
|---|---|---|---|
| $x := c$ | (1) | $Y := c + c'$ | (1) |
| $x := y$ | (2) | $Y := y + c'$ | (4) |
| $x := x + 1$ | (3) | $Y := Y + 1$ | (3) |
| $x := y + c$ | (4) | $Y := y + c + c'$ | (4) |
| $x := x - 1$ | (5) | $Y := Y - 1$ | (5) |
| $x :< c$ | (6) | | |
| $x :> c$ | (7) | | |
| $x :\backsim y + c$ | (8) | $\epsilon$-trans. $Y := x + c'$ | (4) |
| $y + c <: x :< y + d$ | (9) | | |
| $y + c <: x :< z + d$ | (10) | | |

TAB. 4.1 – Horloge de flux $Y$ contrainte par $Y = x + c'$

**Remarque 6** *Le tableau 4.1 semble pessimiste quant aux propriétés vérifiables s'il y a des horloges de flux. Dans la plupart des systèmes, les horloges de flux correspondront à des horloges d'états transportés par un autre système. Ainsi, lorsqu'un modèle Timed AltaRica est mis à plat, les horloges de flux coïncideront souvent à des horloges d'états et n'interviendront pas dans la décidabilité.*

On peut ensuite se demander quelle est la complexité de la mise à plat.

**Conclusion du chapitre.** Dans ce chapitre sur Timed AltaRica, nous avons construit un langage hiérarchique temporisé de haut niveau basé sur le modèle des automates à contraintes temporisés. L'expressivité de ce langage est celle des automates temporisés avec mise à jour [BDFP00] et

transitions inobservables [BGP96]. On connaît ainsi, selon la syntaxe du modèle les propriétés décidables sur ce modèle. On a produit un algorithme de traduction d'un modèle Timed AltaRica en un automate temporisé, combinaison de la mise à plat et de l'algorithme de traduction symbolique. On peut ensuite appliquer des algorithmes existants pour les automates temporisés et décider de nombreuses choses comme l'accessibilité ou des propriétés de logique temporelle. Notamment, trouver les états accessibles d'un modèle Timed AltaRica revient à faire un calcul de point fixe sur les composantes connexes de l'automate temporisé généré contenant au moins une valuation initiale. L'outil présenté dans le chapitre 6 sera utilisé de la sorte. Nous avons également ajouté des opérateurs de modélisation comme l'urgence et les priorités temporelles.

Dans le prochain chapitre, nous proposons une extension hybride d'AltaRica en nous inspirant de la méthode de construction suivie pour Timed AltaRica : nous assemblons les caractéristique d'AltaRica au modèle des automates hybrides [Hen96].

# Chapitre 5

# Hybrid AltaRica

Dans ce chapitre, nous présentons une généralisation de l'extension temporisée précédente. A nouveau, nous conservons les caractéristiques inhérentes aux modèles AltaRica comme les coordinations de flux, les broadcast et les priorités. Notre construction se fera en deux temps : nous étudierons une version hybride des composants en nous inspirant du modèle des automates hybrides [Hen96] puis nous nous intéresserons à l'aspect hiérarchique.

Nous étendons en premier lieu la syntaxe : de même que pour Timed AltaRica, nous introduisons un nouveau type *analog* symbolisant les variables continues. Les variables continues seront manipulées de la même manière que les horloges : test sur les transitions, remises à jour et invariants temporels. Nous spécifions en outre des conditions d'évolution. Nous nous restreignons à des conditions d'évolution rectangulaires : on se place dans la sous classe des automates hybrides telle que l'application $F$ est un champ de vecteurs défini par : $\forall x \in X,\ \dot{x} \in [p, q]$ avec $p, q \in \mathbb{Z}$.

Dans la section 5.1, nous exprimons les ajouts syntaxiques. Il apparaît nécessaire de ne spécifier des conditions d'évolution que sur les variables d'état : les lois vérifiées par les variables de flux découleront des conditions données par l'assertion. Dans la section 5.1, nous exprimons la sémantique des composants hybrides en terme de STTI. Une fois le modèle des automates à contraintes hybrides étudiés, nous construisons la hiérarchie dans la section 5.2 et nous prouvons un théorème de réécriture sur les automates hybrides. Enfin, dans la section 5.3, nous proposons un Algorithme de traduction d'un composant hybride en un automate hybride.

## 5.1 Une extension hybride des composants AltaRica

### 5.1.1 Extension syntaxique hybride

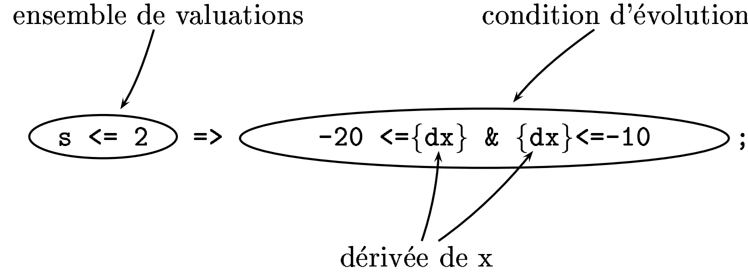
Les variables continues sont une généralisation des horloges et on applique les extensions syntaxiques à ces nouvelles variables.

**Déclarations.** D'abord, on ajoute le mot clé *analog*, ce type touchera les variables de flux et d'état.

```
state s : [0,5];  x : analog;
```

**Transitions.** Les transitions sont gardées par des formules discrètes et des contraintes de variables continues définies dans le paragraphe 3.2 page 47. Les affectations concernent toutes les variables d'états, y compris les variables continues.

**Contraintes.** Les assertions discrètes dans le champ *assert* vérifient la syntaxe d'AltaRica. On peut spécifier des invariants dans le *tinvariant*, toujours de la forme $P \implies I$ où $I$ est convexe. Il faut en outre exprimer les conditions d'évolutions des variables continues. On introduit des accolades pour distinguer les variables pointées des autres variables. Avec les mêmes motivations que les invariants temporels, nous les représentons de la manière suivante :




### Les composants hybrides

Nous présentons à présent la syntaxe générale d'un composant hybride et nous obtenons la Figure 5.1.

```
node Composant_hybride
 flow   f : type ...; y : analog ...
 event  e,h < g
 state  s : type ...; x : analog ...
 trans  formule(s,f) & contrainte_linéaire(x,y) |- e -> s:= expression(s,f),
                   x:=expression_linéaire(x,y);
       ...
 init   formule (s,f,x);
 assert formule(s,f);
 tinvariant formule(s,f) => contrainte_linéaire(x,y);
    formule(s,f) => contrainte_linéaire({dx});
edon
```

FIG. 5.1 – Syntaxe d'un composant hybride

Les expressions linéaires et les contraintes linéaires ont été définies dans la section 3.2 page 47. Ainsi, une variable continue de flux $y$ est une fonction affine des variables continues d'état $x_i$ : $y = \Sigma a_i x_i$ en chaque configuration.

**Exemple 30 (Jeu de Marienbad - version blitz - I)** *On reprend les règles du jeu données dans l'Exemple 2 page 24. Cette fois, une partie est jouée avec une pendule et donc à une cadence rapide. Un délai est donné à chaque joueur et le temps total qu'il passe à réfléchir alors que c'est à son tour de jouer doit être inférieur à ce délai. Un joueur dont le délai expire a perdu. La spécification d'un paquet reste inchangée. En revanche, l'arbitre mesure le temps et s'assure que le délai p imparti à chaque joueur est respecté.*

*Nous donnons la spécification de l'arbitre du blitz dans la Figure 5.2. Les joueurs ne peuvent jouer que si $x_a < p$ et $x_b < p$, c'est-à-dire quand ils n'ont pas encore utilisé tout leur temps. Quand la variable $AvaJouer = 0$ c'est au tour de $A$, quand elle vaut $1$ c'est à $B$ et quand elle vaut $2$ le jeu est terminé car l'un des deux joueurs soit a perdu en prenant la dernière allumette ou son horloge est tombée.*

```
node Arbitre_B
  state AvaJouer : [0,2]; xa,xb: analog;
  event joueurA, joueurB, stop
  trans AvaJouer=0 & (xa<p & xb<p) |- joueurA -> AvaJouer := 1;
      AvaJouer=1 & (xa<p & xb<p) |- joueurB -> AvaJouer := 0;
      AvaJouer<=1 & (xa=p | xb=p) |- stop -> AvaJouer := 2;
  init AvaJouer := 0,xa:=0,xb:=0;
  tinvariant (AvaJouer<=1) => (xa <= p & xb <= p);
       (AvaJouer=0) => ({dxa}=1 & {dxb}=0);
       (AvaJouer=1) => ({dxa}=0 & {dxb}=1);
       (AvaJouer=2) => ({dxa}=0 & {dxb}=0);
edon
```

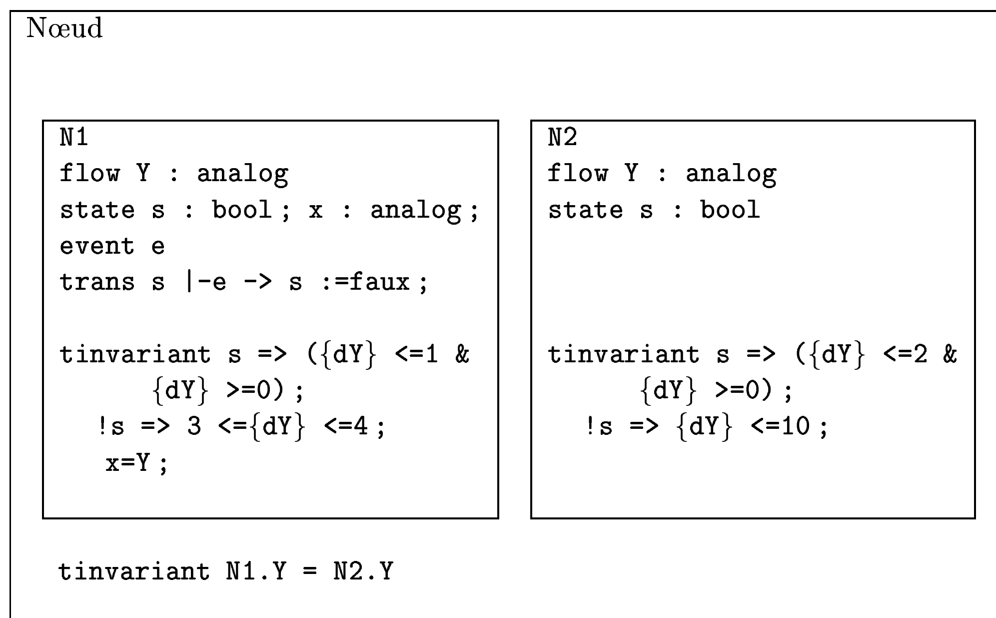
Fig. 5.2 – Arbitre tenant compte du temps

### La grammaire de Hybrid AltaRica

Le grammaire concrète est formellement définie en Annexe D page 151. Pour cela, nous avons repris la grammaire concrète de Timed AltaRica définie dans l'Annexe C et nous avons ajouté les extensions syntaxiques présentées ici.

### Interface des composants hybrides

On illustre dans un premier cas de figure la raison pour laquelle donner des conditions d'évolution sur les variables de flux est absurde :

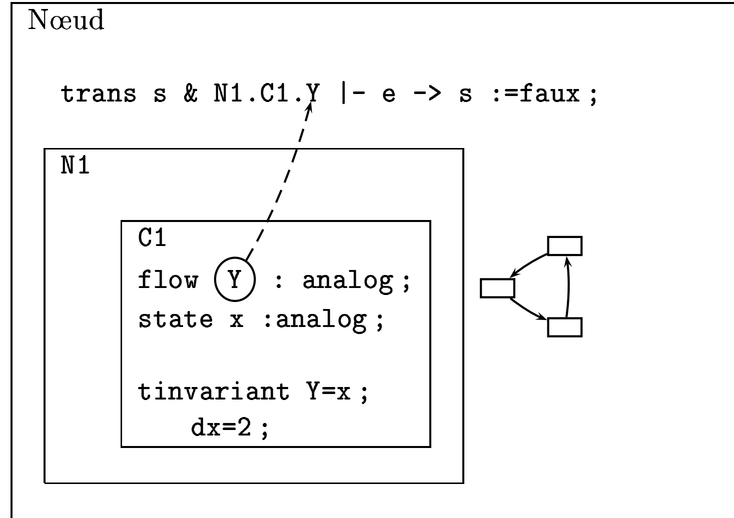



Deux sous-nœuds partagent la même variable de flux et donc spécifient chacun des conditions d'évolution. Alors, le nœud hybride est le produit des deux sous-nœuds et on doit calculer les conditions d'évolution en chaque configuration de ce nœud hybride : chaque variable continue suit la loi d'évolution satisfaisant toutes les conditions d'évolution spécifiées dans chaque nœud hybride.

| Configuration $(N_1.s, N_2.s)$ | condition d'évolution |
|---|---|
| (vrai,vrai) | $0 \leq dY \leq 1$ |
| (vrai,faux) | $0 \leq dY \leq 1$ |
| (faux,vrai) | faux |
| (faux,faux) | $3 \leq dY \leq 4$ |

Les conditions d'évolution agissent également sur les configurations admissibles : la configuration $(faux, vrai)$ n'admet aucune dérivée pour la variable de flux $Y$, donc elle n'admet aucune valeur de flux $Y$ admissible.

Dans AltaRica, Timed AltaRica et Hybrid AltaRica, on suppose qu'une variable de flux n'est "possédée" (dans le sens fonction linéaire des variables continues d'états) que par un seul nœud et est uniquement lisible par les nœuds qui la voit. Une modélisation attendue est la suivante :




La variable continue de flux $Y$ appartient au composant $C1$ et est remontée dans la hiérarchie. Sa valeur peut être testée comme c'est le cas dans la transition spécifiée dans Nœud. Sa dérivée est connue par la condition $Y = x$, on a alors $\dot{Y} = \dot{x}$.

### 5.1.2 Sémantique des composants hybrides

Nous exprimons à présent la sémantique d'un composant Hybrid AltaRica. De même que dans Timed AltaRica, on définit d'abord formellement la syntaxe : un composant correspond à un automate à contraintes hybride, puis on en donne la sémantique sous forme de STTI.

**Définition 30 (Composant hybride - syntaxe abstraite)** *Un composant hybride est un octuplet $\mathcal{H} = \langle V_S, H_S, V_F, H_F, E, A, \dot{A}, M \rangle$ où :*

1. *$V_S, V_F$ sont des ensembles finis disjoints appelés respectivement ensembles des variables d'états et de flux. $C_S, C_F$ sont des ensembles finis disjoints appelés respectivement ensembles des variables* continues d'état *et* continues de flux. *On note $Z = V_S \cup V_F$, $C = H_S \cup H_F$ et $\dot{C}$ l'ensemble des variables continues pointées. On suppose que $Z \cap C = \emptyset$ ;*
2. *$E = E_+ \cup \{\epsilon\}$ est un ensemble fini d'événements et $\epsilon \notin E_+$ est l'action inobservable ;*
3. *l'*assertion *du composant $A$ est une formule telle que : $A = A_Z \wedge A_I \in \mathcal{F}(\mathcal{L}_{Z \cup C})$ avec*
   *(a) $A_Z \in \mathcal{F}(\mathcal{L}_Z)$,*

   (b) $A_I = \bigwedge_{k \in K} P_k \Longrightarrow I_k$ *où* $K$ *est un ensemble fini d'indices,* $P_k \in \mathcal{F}(\mathcal{L}_Z)$ *et* $I_k \in \mathcal{C}_l$ *définit une région convexe de* $\mathbb{R}^p$ *si* $|C| = p$ *et définit également les variables de flux en une fonction affine des variables d'état continues ;*

4. *la* condition d'évolution *du composant* $\dot{A}$ *est une formule telle que* $\dot{A} = \bigwedge Q \Longrightarrow J \in \mathcal{F}(\mathcal{L}_{Z \cup \dot{H}_S})$ *avec*
   (a) $Q \in \mathcal{F}(\mathcal{L}_Z)$,
   (b) $J \in \mathcal{C}(\dot{H}_S)$ *est une contrainte sur les dérivées des variables d'état de la forme* $\bigwedge_{x \in H_S} \dot{x} \in [p, q]$ *avec* $p, q \in \mathbb{N}$ *et* $p \leq q$.
5. $M \subseteq (\mathcal{F}(\mathcal{L}_Z) \times \mathcal{C}(C)) \times E \times (\mathcal{T}(\mathcal{L}_Z)^{V_S} \times \mathcal{U}_l(H_S))$ *est un ensemble de* macro-transitions $((g, \gamma), e, (a, R))$ *telles que :*
   (a) $(g, \gamma)$ *est une* garde *où* $g \in \mathcal{F}(\mathcal{L}_Z)$ *est une contrainte sur les variables discrètes et* $\gamma \in \mathcal{C}(C)$ *est une contrainte de variables continues ;*
   (b) $e \in E$ *est l'événement déclenchant la transition ;*
   (c) $a : V_S \to \mathcal{T}(\mathcal{L}_Z)$ *est une affectation pour les variables de* $V_S$ *et* $R \in \mathcal{U}_l(H_S)$ *est l'ensemble des variables continues remises à jour par la transition*[1] *;*
6. $<$ *est une relation de priorité (cf Définition 4 page 27).*

**Remarque 7** *Si* $\dot{A} = \bigwedge_{i \in [1,r]} Q_i \Longrightarrow J_i$, *soit* $(s, \nu, f, \mu)$ *une configuration dans* $[\![A]\!]$, *lorsque l'on donne la sémantique de la condition d'évolution, il faut résoudre les deux problèmes suivant :*

1. *si* $(s, f) \notin \bigcup_i [\![Q_i]\!]$, *cela signifie qu'aucune condition d'évolution n'a été spécifiée en cette configuration. Il y a deux possibilités : soit on choisit d'imposer de spécifier toutes les conditions d'évolution et alors* $(s, f)$ *n'est pas admissible, soit on décide de mettre une valeur par défaut* $\bigwedge \dot{x} \in \mathbb{R}$. *Nous avons choisi cette solution.*
2. *si* $(s, f) \in [\![Q_i]\!] \cap [\![Q_j]\!]$ *alors on doit avoir* $\dot{\nu} \in [\![J_i]\!] \cap [\![J_j]\!]$. *En particulier, si* $[\![J_i]\!] \cap [\![J_j]\!] = \emptyset$, *les conditions de flot sont contradictoires donc* $f$ *n'est pas dans l'interface de* $s$. *Par exemple, si on a* `(s=1 & f=1) => (dx = 2)` *et* `(f=1) => (dx <1)` *alors* $f = 1 \notin \pi(1)$.

La sémantique d'un composant hybride est exprimée par un STTI et associe des valeurs aux variables, dans $\mathcal{D}$ pour les discrètes et $\mathbb{R}$ pour les variables continues.

**Définition 31 (Sémantique d'un composant hybride)** *Soit* $\mathcal{H} = \langle V_S, H_S, V_F, H_F, E, A, \dot{A}, M \rangle$ *un composant hybride avec* $|H_S| = n$, $|H_F| = m$, $A = \bigwedge_{i \in [1,l]} P_i \Longrightarrow I_i$ *et* $\dot{A} = \bigwedge_{i \in [1,r]} Q_i \Longrightarrow J_i$. *La sémantique de* $\mathcal{T}$ *sur le domaine temporel* $\mathbb{T}$ *est le système de transitions hybride interfacé* $[\![\mathcal{H}]\!] = \langle E_t, F_t, S_t, \pi, T \rangle$ *de dimension* $(n, m)$ *construit de la manière suivante :*

1. $E_t = E \cup \mathbb{T}$ *;*
2. $F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$ *;*
3. $S_t = \{(s, \nu) \in \mathcal{D}^{V_S} \times \mathbb{R}^n \mid \pi(s, \nu) \neq \emptyset\}$ *;*
4. $\pi : S_t \times \mathbb{R}^n \to 2^{F_t \times \mathbb{R}^m}$ *est définie par :* $(f, \mu) \in \pi(s, \nu)$ *si, et seulement si*
   (a) $(s, \nu, f, \mu) \in [\![A]\!]$ *;*
   (b) $\exists \dot{\nu} \in \mathbb{R}^n$ *tel que que pour tout* $i \in [1, r]$, $(s, f) \in [\![Q_i]\!]$ *alors* $\dot{\nu} \in [\![J_i]\!]$ *;*
   (c) $\exists \dot{\mu} \in \mathbb{R}^m$ *tel que pour tout* $i \in [1, l]$ *tel que* $(s, f) \in [\![P_i]\!]$, *on a* $(\nu, \mu) \in [\![I_i]\!]$ *et on en déduit les valeurs possibles de* $\dot{\mu}$ *en fonction de* $\dot{\nu}$ *(cf paragraphe page 100) ;*

[1] Il s'agit d'affectations linéaires définies 3.2 page 47.

5. *$T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ tel que $T = [\![M]\!] \upharpoonright_<$ avec :*
   (a) *$\upharpoonright_<$ est donné dans la Définition 5 page 27;*
   (b) *soit la transition $t = ((g, \gamma), e, (a, R))$, on définit $[\![t]\!]$ par :*

$$((s,\nu),(f,\mu),e,(s',\nu'),(f',\mu')) \in [\![t]\!] \text{ si } \begin{cases} ((s,\nu),(f,\mu)) \in [\![A \wedge g \wedge \gamma]\!] \\ \wedge \quad ((s',\nu'),(f',\mu')) \in [\![A]\!] \\ \wedge \quad s' = a(s,f) \ \wedge \ \nu' = R(\nu,\mu) \end{cases}$$

   (c) *soit $\delta \in \mathbb{T}$, on définit $[\![\delta]\!]$ par :*

$$((s,\nu),(f,\mu),\delta,(s,\nu'),(f,\mu')) \in [\![\delta]\!] \text{ si } \begin{cases} ((s,\nu),(f,\mu)) \in [\![A]\!] \\ \wedge \quad \exists \overline{v} \in [\![J(s,f)]\!], \ \nu' = \nu + \delta \times \overline{v} \\ \wedge \quad ((s,\nu'),(f,\mu')) \in [\![A]\!] \\ \wedge \quad \forall \delta' \le \delta, \ \exists \mu_{\delta'}, \ (\nu + \delta' \times \overline{v}, \mu_{\delta'}) \in [\![I(s,f)]\!] \end{cases}$$

   *avec $I(s,f) = \bigwedge_{k \in [1,l],(s,f) \in P_k} I_k$ et $J(s,f) = \bigwedge_{k \in [1,r],(s,f) \in Q_k} J_k$.*
   (d) *$[\![M]\!] = \cup_{t \in M} [\![t]\!] \bigcup \cup_{\delta \in \mathbb{T}} [\![\delta]\!]$.*

**Remarque 8** *La fonction $\pi$ en une configuration $(s, \nu)$ est non vide s'il existe des valeurs $(f, \mu)$ qui satisfont l'invariant et s'il existe des valeurs des dérivées admissibles : si par exemple dans l'invariant temporel $y = c + \Sigma a_i x_i$, nécessairement on a la relation $\dot{y} = \Sigma a_i \dot{x}_i$.*

*Dans un composant hybride, on peut laisser passer un temps $\delta$ dans une configuration $((s,\nu),(f,\mu))$, c'est-à-dire $\big((s,\nu),(f,\mu),\delta,(s,\nu'),(f,\mu')\big) \in T$ si, et seulement si $(f,\mu) \in \pi(s,\nu)$, $(f,\mu') \in \pi(s,\nu')$, s'il existe un vecteur vitesse $\overline{v} = \dot{\nu}$ qui satisfait la condition d'évolution en $(s,f)$, qu'il existe un vecteur $\dot{\mu}$ respectant les conditions induites sur les dérivées par l'invariant temporel et si pour tout pas intermédiaire $\delta'$ il existe $\mu_{\delta'}$ tel que $(f, \mu_{\delta'}) \in \pi(s, \nu + \delta' \times \overline{v})$.*

**Exemple 31 (Jeu de Marienbad - version blitz - II)** *On donne la sémantique du composant hybride Arbitre_B de la Figure 5.2 page 97. Il y a trois configurations discrètes AvaJouer $= 0, 1$ ou $2$. En chacune de ces configurations, une condition d'évolution est donnée pour les variables $x_a$ et $x_b$ qui détermine les transitions continues. Un invariant temporel est donné pour les configurations $0$ et $1$. On a représenté le STTI $[\![$Arbitre_B$]\!]$ dans la Figure 5.3 : les transitions discrètes sont dessinées par des traits continus et les transitions temporelles par des traits pointillés, on suppose que les quantités $x_a < p$, $x_a + \delta < p$, $x_b < p$ et $x_b + \delta < p$.*

*Depuis une configuration $(0, x_a, x_b)$ telle que $x_a < p$ et $x_b < p$, on peut laisser passer un temps $\delta$ si, et seulement si $x_a + \delta \le p$ et alors on atteint la configuration $(0, x_a + \delta, x_b)$. Depuis une configuration $(0, x_a, x_b)$ telle que $x_a < p$ et $x_b < p$, on peut faire une transition étiquetée* JoueurA *et alors on atteint la configuration $(1, x_a, x_b)$.*

### Réduction des flux

On n'a spécifié aucune condition d'évolution sur les variables de flux car elles sont déduites des invariants temporels. Ces invariants temporels sont des contraintes linéaires. Pour pouvoir implanter ces variables, nous devons réduire les invariants autorisés. En effet, une condition nécessaire pour pouvoir implanter une variable de flux $Y$ appartenant à un composant hybride, c'est-à-dire qu'elle n'est pas seulement en lecture, est que $\dot{Y}$ existe et soit bornée en toute configuration.

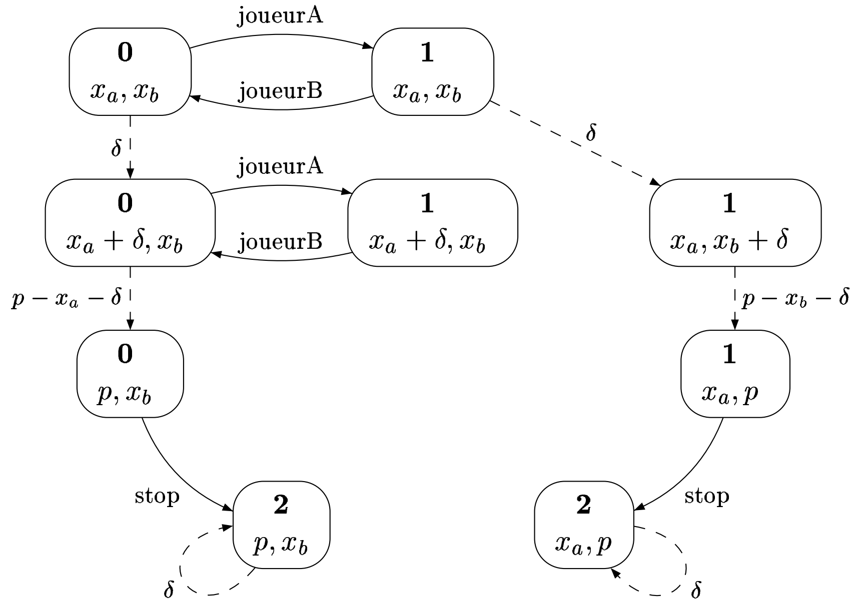



FIG. 5.3 – Sémantique du composant hybride *Arbitre_B*

1. Si $Y \leq x$ avec $x$ variable continue d'état alors $\dot{Y}$ n'est pas bornée, donc cette condition n'est pas suffisante,
2. Si $x - 1 \leq Y \leq x$ avec $x$ variable continue d'état alors $\dot{Y}$ n'est pas bornée,
3. Plus généralement, si $Y \in I$ avec $\mathring{I} \neq \emptyset$ et $I$ dépend ou non des variables continues, alors $\dot{Y}$ n'est pas borné.
4. Donc si $Y \in I$ nécessairement $\mathring{I} = \emptyset$. Si $Y \in \{0, 1, 2\}$ alors $Y$ peut osciller n'importe quand entre ces trois valeurs et $Y$ n'est pas dérivable.
5. Enfin, si on a des relations entre plusieurs variables flux et d'état, par exemple $y_1 + y_2 = x$, on n'assure pas que $\dot{Y}$ est borné.

D'après ces constats, on doit avoir en chaque configuration $Y = F(X)$ et on cherche une condition nécessaire sur $F$ pour que $Y(t)$ existe et $\dot{Y}(t) \in [a, b]$. Soit $(s, f, X, F(X))$ une configuration,

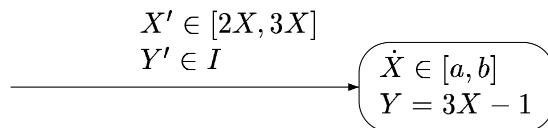
**Premier cas :** $F(X) = \Lambda X + B$**.** On a alors $\dot{Y} = \Lambda\dot{X}$.

Condition d'évolution sur $X$ :

- si $\dot{X} = a$ alors $\dot{Y} = \Lambda a$,
- si $\dot{X} \in [a, b]$, on a alors $\dot{Y} = \Lambda\dot{X}$ et on peut calculer les valeurs accessibles de $X$ et $Y$ en $(s, f)$ en éliminant la variable $X_0$ dans les équations $at + X_0 \leq X(t) \leq bt + X_0$ et $Y = \Lambda X + B$.

Si de la configuration $(s', f', X, Y)$, on tire une transition $((g, \gamma), e, (a, R))$ et on atteint la configuration $(s, f, X', Y')$, on doit pouvoir calculer les valeurs possibles de $Y'$ :

- si on a une affectation déterministe $X' = AX + b$ alors $Y' = \Lambda RX + \Lambda b + B$,
- si on a une affectation indéterministe $AX + b \leq X' \leq cX + d$ alors $Y$ doit être indépendant de $X$ en $(s, f)$ sinon on en déduit que $Y \in [F(AX + b), F(cX + d)]$ donc on obtient trop de comportements.



Dans le cas de figure ci-dessus, on doit trouver automatiquement $I$. Or $I$ peut dépendre de $X$ mais pas de $X'$, donc la seule valeur possible est $I = [6X-2, 9X-3]$. Ainsi l'invariant $Y = 3X - 1$ n'est pas toujours vérifié. Ce comportement n'est donc pas implantable.

**Deuxieme cas :** $F(X) = \begin{cases} \Lambda_1 X + B_1 & si\ T_1 \in \mathbb{F}(H_S) \\ \cdots & \\ \Lambda_n X + B_n & si\ T_n \in \mathbb{F}(H_S) \end{cases}$ avec $\cup T_i = \wedge_{k \in K|(s,f)\in P_k} I_k$, $T_i \cap T_j = \emptyset$ et $\uparrow T_i \subseteq T_i$[2]. En effet, il faut que :

- $\cup T_i = \wedge_{k\in K|(s,f)\in P_k} I_k$ afin que $Y$ soit définie pour toutes les valuations possibles de $X$,
- $T_i \cap T_j = \emptyset$ pour ne pas donner des relations contradictoires,
- $\uparrow T_i \subseteq T_i$ nous permet d'assurer que le changement se fera en un instant précis.

On a alors $\dot{Y} = \Sigma_{i=1,\cdots,n} \chi_{I_i} \Lambda_1 \dot{X}$ où $\chi_I$ est la fonction indicatrice de $I$, c'est-à-dire $\chi_I(t) = 1$ si $I$ est vérifiée en $t$ et 0 sinon.

Par exemple, on peut avoir `s=1 => ((x<10 => Y=x) & (x>=10 => Y=11))`.

Pour les mêmes raisons que dans le premier cas $F(X) = \Lambda X + B$, la loi d'évolution des variables continues d'états n'intervient pas, i.e. on doit seulement avoir $\dot{x} \in [a, b]$. De même, en cas d'affectations non déterministes, $Y$ doit être indépendant de $X$ dans les configurations discrètes atteintes.

**Cas général.** Finalement, on considère des variables de flux telles que :

1. $Y$ est une fonction continue affine par morceaux et continue à droite fonction des variables continues d'état. Elle est donc dérivable presque partout et les dérivées à gauche et à droite sont bornées.
2. Dans les configurations atteintes par des transitions dont les mises à jour sont non déterministes pour certaines variables continues, $Y$ doit être indépendantes de ces variables en ces configurations.

Alors, en reprenant l'item 4 de la Définition 31 page 99, $(f, \mu) \in \pi(s, \nu)$ si, et seulement si

1. $(s, \nu, f, \mu) \in [\![A]\!]$ ;
2. $\exists \dot{\nu} \in \mathbb{R}^n$ tel que que pour tout $i \in [1, r]$, $(s, f) \in [\![Q_i]\!]$ alors $\dot{\nu} \in [\![J_i]\!]$ ;
3. si $\mu = \Sigma \chi_{I_i}(\Lambda_i \nu + b_i)$ alors $\dot{\mu}$ existe si, et seulement si $\dot{\mu} = \Sigma \chi_{I_i}(\Lambda_i \dot{\nu} + b_i)$ existe.

**Exemple 32** *On considère le composant hybride suivant :*

```
node N1
 state s : bool, x : analog;
 flow Y : analog;
 event e
 trans s |- e-> s:=false;
 tinvariant s => ((x<10 => Y=x) & (x>=10 => Y=11));
            ~s => Y=x+1;
            s=> {dx}=2;
            ~s => -1<= {dx}<= 3;
edon
```

*Alors on représente sa sémantique dans la Figure 5.4. Le saut de la fonction $Y(t)$ est vu comme le résultat d'une action interne du composant hybride.*

[2] Il s'agit du front montant de la définition 21 page 66

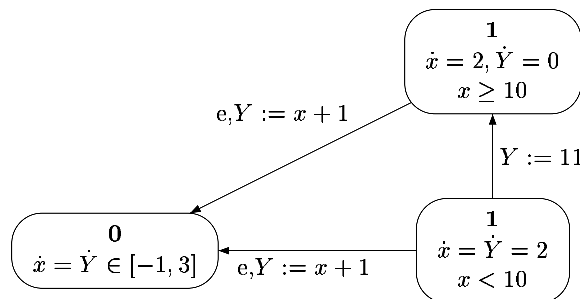



FIG. 5.4 – Sémantique de $N_1$

## 5.2 Les nœuds hybrides

On peut alors définir la syntaxe des nœuds hybrides, les ajouts seront les mêmes que pour les composants hybrides, la différence est que les formules et les contraintes peuvent porter sur les variables de flux des sous composants.

```
node Nœud
 flow   f : type ...; y : analog ...
 event  e, h < g
 state  s : type ...; x : analog ...
 sub C: Composant; ...
 trans  formule(s,f,C.f) & contrainte(x,y,C.y) |- e -> s:= expression,
                  x:=expression_linéaire(x,y,C.y);
     ...
 sync <e,C.h ...>,
     <h,C.e? ...> ...
      ...
 init   formule (s,f,x);
 assert formule(s,f,C.f);
 tinvariant formule(s,f,C.f) => contrainte_linéaire(x,y,C.y);
      formule(s,f,C.f) => contrainte_linéaire({dx});
edon
```

FIG. 5.5 – Syntaxe d'un nœud hybride

**Exemple 33 (Jeu de Marienbad - version blitz - III)** *On spécifie entièrement le jeu de Marienbad version blitz. On suppose qu'on reste un temps minimal $\delta$ en un état de contrôle car physiquement un joueur doit au moins retirer les allumettes d'un paquet. Ainsi pour un blitz de temps $p$ octroyé à chaque joueur, il y a au plus $2p/\delta$ coups. On reprend le nœud* Paquet *donné dans l'exemple 2 page 24 et le nœud hybride* Arbitre_B *représenté à la Figure 5.2 page 97.*

```
node Marienbad_B
  state x :clock;
  event d
  trans x>=delta |- d -> x:=0;
  sub R : Arbitre_B; L1, L2, L3, L4 : Paquet;
  sync <d,R.joueurA, L1.joue?, L2.joue?, L3.joue?,L4.joue?> = 1;
    <d,R.joueurB, L1.joue?, L2.joue?, L3.joue?,L4.joue?> = 1;
  init L1.n := 1, L2.n := 3, L3.n := 5, L4.n := 7;
edon
```

*On considère une constante* delta *qui sera le temps minimum en chaque configuration et on ajoute un événement* d *qui contraindra les transitions de jeu selon la garde $x \geq \delta$.*

### 5.2.1 Sémantique des nœuds hybrides

On définit à présent formellement les nœuds hybrides. Cette définition est la même que celle des nœuds temporisés, Définition 27 page 81. La seule différence réside dans le remplacement de temporisé par hybride.

**Définition 32 (Nœud hybride - syntaxe abstraite)** *Un* nœud hybride *est un uplet $\mathcal{N} = \langle V_F, H_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ avec :*

1. *$V_F$ est un ensemble de variables de flux et $H_F$ est l'ensemble des* ***variables continues*** *de flux ;*
2. *$E = E_+ \cup \{\epsilon\}$ est un ensemble fini d'événements et $\epsilon \notin E_+$ est l'action inobservable ;*
3. *$<$ est une relation de priorité sur $E$ ;*
4. *pour tout $i \in [1, n]$, $\mathcal{N}_i$ est un* ***composant hybride*** *ou un* ***nœud hybride****, $V_{F_i}$ (resp. $H_{F_i}$) est l'ensemble des variables (resp. des* ***variables continues****) de flux de $\mathcal{N}_i$ et $E_i$ est l'ensemble des événements. On suppose $\forall i \neq j \in [1, n], V_{F_i} \cap V_{F_j} = H_{F_i} \cap H_{F_j} = \emptyset$ ;*
5. *$\mathcal{N}_0$ est un* ***composant hybride*** *spécial appelé le* contrôleur*, il s'agit du comportement local du nœud. L'ensemble des événements de ce nœud est $E_0 = E$ et aucune relation de priorité n'est spécifiée pour ce nœud. L'ensemble des variables de flux de $\mathcal{N}_0$ est $F_0 = F \cup F_1 \cup \cdots \cup F_n$, et l'ensemble des* ***variables continues de flux*** *est $H_{F_0} = H_F \cup F_1 \cup \cdots \cup H_{F_n}$ ;*
6. *$\tilde{V} \subseteq E_0 \times E_{1,v} \times \cdots \times E_{n,v}$ est une expansion de l'ensemble des vecteurs de synchronisation avec la relation de priorité $<_{\tilde{V}}$ (cette relation exprime l'ordre induit par les broadcast).*

La sémantique d'un nœud hybride est exprimée à l'aide d'un STTI.

**Définition 33 (Nœud hybride - sémantique)** *Soit un nœud hybride $\mathcal{N} = \langle V_F, H_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$. On suppose que pour tout $i \in [0, n]$, $[\![\mathcal{N}_i]\!] = \langle E_{t,i}, F_{t,i}, S_{t,i}, \pi_i, T_i\rangle$ de dimension $(n_i, m_i)$. Alors la sémantique de $\mathcal{N}$ est le STTI $[\![\mathcal{N}]\!] = \langle E_t, F_t, S_t, \pi, T\rangle$ de dimension $(\Sigma_{k=0}^{n} n_i, |H_F|)$ défini par :*

1. *$F_t = \mathcal{D}^{V_F} \times \mathbb{R}^m$ où $m = |H_F|$ ;*
2. *$\pi(s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n) = \{(f, \mu) \in F_t |\ \forall i \in [1, n],\ \exists (f_i, \mu_i) \in F_{t,i},\ (f_i, \mu_i) \in \pi_i(s_i, \nu_i)$ et $(f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n) \in \pi_0(s_0, \nu_0)\}$ ;*
3. *$S_t = \{q \in S_{t,0} \times S_{t,1} \times \cdots \times S_{t,n} |\ \pi(q) \neq \emptyset\}$ ;*
4. *$T \subseteq S_t \times F_t \times E_t \times S_t \times F_t$ est défini par :*
   (a) *on considère $<_0$ la relation de priorité engendrée par $<$ et définie par :* $(e_0, e_1, \cdots, e_n) <_0 (e'_0, e'_1, \cdots, e'_n) \iff e_0 < e'_0$,
   (b) *soit $T_{\mathcal{N}}$ l'ensemble des transitions définies par :*
   – *pas discret : si $e = (e_0, e_1, \cdots, e_n) \in \tilde{V}$*

$$\langle (s_0, \nu_0, \cdots, s_n, \nu_n), f, \mu, e, (s'_0, \nu'_0 \cdots, s'_n, \nu'_n), f', \mu'\rangle \in T_{\mathcal{N}} \iff$$

$$\begin{cases} \exists (f_0, \mu_0) = (f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n) \in \pi_0(s_0, \nu_0), \\ \exists (f'_0, \mu'_0) = (f', \mu', f'_1, \mu'_1, \cdots, f'_n, \mu'_n) \in \pi_0(s'_0, \nu'_0), \\ \forall i \in [0, n], \big((s_i, \nu_i), (f_i, \mu_i), e_i, (s'_i, \nu'_i), (f'_i, \mu'_i)\big) \in T_i \end{cases}$$

– *pas continu :* $\delta \in \mathbb{T}$

$$\langle(s_0, \nu_0, \cdots, s_n, \nu_n), f, \mu, e, (s_0, \nu'_0 \cdots, s_n, \nu'_n), f, \mu'\rangle \in T_{\mathcal{N}} \iff$$

$$\begin{cases} \exists(f_0, \mu_0) = (f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n) \in \pi_0(s_0, \nu_0), \\ \exists(f_0, \mu'_0) = (f, \mu', f_1, \mu'_1, \cdots, f_n, \mu'_n) \in \pi_0(s_0, \nu'_0), \\ \forall i \in [0, n], \big((s_i, \nu_i), (f_i, \mu_i), \delta, (s_i, \nu'_i), (f, \mu'_i)\big) \in T_i \end{cases}$$

(c) *alors* $T = \left(T_{\mathcal{N}} \restriction <_{\tilde{V}}\right) \restriction <_0$.

**Exemple 34** *On reprend l'exemple 32 page 102 et on considère un système décrit comme suit :*

```
node Noeud
 sub nn : N1; pp : N2;
 tinvariant nn.Y = pp.Y;
edon
```

```
node N2
 flow Y : analog;
 state s : bool;
 event f,g
 trans s & Y <= 10 |-f -> s:=false;
   ~s |-g -> s:=true;
edon
```

*On peut alors calculer la sémantique du nœud hybride. Elle est représentée dans la Figure 5.6. Les variables Y n'apparaissent plus puisqu'elles apparaissent dans la portée d'un quantificateur existentiel. Comme les domaines sont finis, on peut éliminer les quantificateurs.*

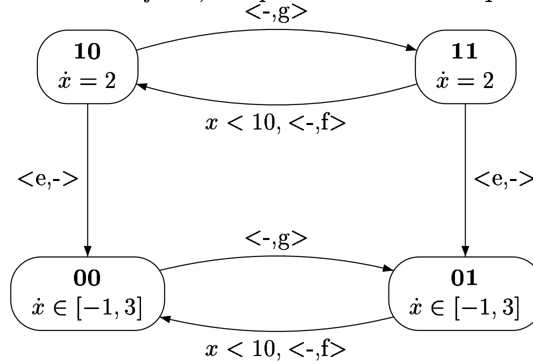



FIG. 5.6 – Sémantique du nœud hybride *Noeud*

### 5.2.2 Réécriture d'un nœud hybride en un composant hybride

Hybrid AltaRica conserve la propriété de modularité suivante : si on considère $n$ composants hybrides bisimilaires respectivement à $n$ autres, alors le produit synchronisé des $n$ premiers composants hybrides reste bisimilaire au produit synchronisé des $n$ seconds, avec la même contrainte de synchronisation.

**Théorème 7 (Bisimulation et composition hybride)** *Soient* $\mathcal{N} = \langle V_F, H_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ *et* $\mathcal{N}' = \langle V_F, H_F, E, <, \mathcal{N}'_0, \cdots, \mathcal{N}'_n, (\tilde{V}, <_{\tilde{V}})\rangle$ *deux nœuds hybrides tels que* $\forall i \in [0..n]$ *il existe un homomorphisme de bisimulation temporisée interfacée* $h_i$ *de* $[\![\mathcal{N}_i]\!]$ *dans* $[\![\mathcal{N}'_i]\!]$. *Alors il existe un homomorphisme de bisimulation temporisée interfacée* $h$ *de* $[\![\mathcal{N}]\!]$ *dans* $[\![\mathcal{N}']\!]$.

**Preuve du théorème 7**. *La démonstration est identique à la preuve 4.1.3 page 60, même pour l'application* $\pi$. □

Le procédé de réécriture d'un nœud hybride en un composant hybride est le même que pour un nœud temporisé, cf. Définition 29 page 84. On écrit en caractères **gras** les différences d'avec la sémantique symbolique temporisée.

**Définition 34 (Sémantique hybride symbolique)** *Si $\mathcal{N} = \langle V_F, H_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ est un nœud hybride, on note $\mathcal{C}_\mathcal{N} = \langle V_S, H_S, V_F, H_F, E, A, \dot{A}, M\rangle$ le composant hybride construit comme suit :*

1. $\forall i = 0 \cdots n$
   (a) *si $\mathcal{N}_i$ est un composant hybride, alors on pose $\mathcal{N}'_i = \mathcal{N}_i \upharpoonright <^i$*[3] *;*
   (b) *si $\mathcal{N}_i$ est un nœud hybride, alors on définit $\mathcal{N}'_i = \mathcal{C}_{\mathcal{N}i}$ ;*
   (c) *on note $\mathcal{N}'_i = \langle V'_{S_i}, H'_{S_i}, V'_{F_i}, H'_{F_i}, E'_i, A'_i, \dot{A}'_i, M'_i, \emptyset\rangle$ ;*
2. $V_S = V'_{S_0} \cup \cdots \cup V'_{S_n}$ *et* $H_S = H'_{S_0} \cup \cdots \cup H'_{S_n}$ *;*
3. $A = (\exists_{i=1..n}(V'_{F_i} \cup C'_{F_i})). \bigwedge_{i=0..n} A'_i$ *;*
   *où la notation $\exists_{i=1..n}(W_i).\phi$ signifie $\forall i, \exists \eta_i \in W_i,\ \phi(\eta_i)$. Alors pour $((s,\nu),(f,\mu)) \in \mathcal{D}^{V_S} \times \mathbb{R}^{C_S} \times \mathcal{D}^{V_F} \times \mathbb{R}^{C_F}$ on définit :*
   – $((s,\nu),(f,\mu)) \in [\![A]\!] \iff \forall i \in [1..n], \exists \eta_i \in \mathcal{D}^{V'_{F_i}} \times \mathbb{R}^{C'_{F_i}}$ *tel que* $((s,\nu),(f,\mu),\eta_1,\cdots,\eta_n)) \in [\![\bigwedge_{i=0..n} A'_i]\!]$,
   – $\forall i \in [1..n], ((s,\nu),(f,\mu),\eta_1,\cdots,\eta_n)) \in [\![A'_i]\!] \iff ((s_i,\nu_i),\eta_i) \in [\![A'_i]\!]$.
4. $\dot{\mathbf{A}} = \exists_{\mathbf{i=1..n}}(\mathbf{V}'_{\mathbf{F_i}} \cup \mathbf{C}'_{\mathbf{F_i}})). \bigwedge_{\mathbf{i=0..n}} \dot{\mathbf{A}}'_{\mathbf{i}}$ *;*
5. *l'ensemble des macro-transitions $M \subseteq (\mathbb{F} \times \mathcal{B}(C_T)) \times E \times (\mathbb{E}(V_T)^{V_T} \times \mathcal{A}(C_T))$ est défini par $M = (M' \upharpoonright <_{\tilde{V}}) \upharpoonright <_0$, où $<_0$ est la relation de priorité donnée à la définition 33 avec $M'$ défini par $((g,\gamma),e,(a,R)) \in M'$ si, et seulement si :*
   – $\forall i = 0 \cdots n, \exists ((g_i,\gamma_i),e_i,(a_i,R_i)) \in M'_i, g = (\exists_{i=1..n} V_{F_i}).g_0 \wedge \cdots \wedge g_n$, *et* $\gamma = (\exists_{i=1..n} H_{F_i}) .\gamma_0 \wedge \cdots \wedge \gamma_n$,
   – $\forall x \in V_S, x \in V'_{S_i} \implies a(x) = a_i(x)$ *and* $\forall c \in H_S, c \in C'_{S_i} \implies R(c) = R_i(c)$.
   – $e = (e_0, e_1, \cdots, e_n) \in \tilde{V}$.

**Exemple 35** *On reprend l'exemple 34 page 105 et on exprime le nœud hybride réécrit.*

```
node Noeud_reecrit
 state N1.x : analog; N1.s, N2.s : bool;
 event N1.e, N2.e
 trans N1.s |- N1.e -> N1.s:=false;
    N2.s & N1.s & x < 10 |- N2.e -> N2.s := false;
    N2.s & ~N1.s & x <= 9 |- N2.e -> N2.s := false;
   ~N2.s |- N2.e -> s:= true
 tinvariant N1.s => {dx}=2;
      ~N1.s => -1<= {dx}<= 3;
edon
```

*On a éliminé le quantificateur $\exists$ : en $N_1.s = ff$, $Y = x + 1$ donc $Y \leq 10 \iff x \leq 9$ et en $N1.s = tt$, $Y \leq 10 \iff Y < 10 \iff x < 10$.*

**Théorème 8 (Réécriture en composant hybride)** *Soit $\mathcal{N}$ un nœud hybride. Alors $\mathcal{N}$ peut se réécrire en composant hybride $\mathcal{C}_\mathcal{N}$ selon la définition 34 et alors $[\![\mathcal{N}]\!] = [\![\mathcal{C}_\mathcal{N}]\!]$.*

**Preuve du théorème 8**. *On considère les STTI $[\![\mathcal{N}]\!] = \langle E_t, F_t, S^0_t, \pi^0, T^0\rangle$ et $[\![\mathcal{C}_\mathcal{N}]\!] = \langle E_t, F_t, S^1_t, \pi^1, T^1\rangle$, les sémantiques de $\mathcal{N}$ et $\mathcal{C}_\mathcal{N}$.*

[3] On ne considère ici que des priorités statiques ou broadcast, alors les résultats et les résolutions syntaxiques proposées pour AltaRica s'appliquent, cf proposition 1 page 27

**On suppose que tous les $\mathcal{N}_i$ sont des composants hybrides.** *On vérifie d'abord que* $S_t^1 = S_t^0$ *et donc que* $\pi^1 = \pi^0$.

*D'après la définition 32 page 104, on a* $S_t^0 = \{q \in S_{0_t} \times S_{1_t} \cdots \times S_{n_t} | \ \pi^0(q) \neq \emptyset\}$ *avec* $\pi^0(s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n) = \{(f, \mu) \in F_t | \ \forall i \in [1, n], \ \exists (f_i, \mu_i) \in F_{t,i}, \ (f_i, \mu_i) \in \pi_i^0(s_i, \nu_i)$ *et* $(f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n) \in \pi_0^0(s_0, \mu_0)\}$ *et* $\pi_0^0(s_0, \nu_0) = \{(f_0, \mu_0, f_1, \mu_1, \cdots, f_n, \mu_n) | \ (s_0, \nu_0, f_0, \mu_0, f_1, \mu_1, \cdots, f_n, \mu_n) \in [\![A_0]\!] \ \wedge \dot{\nu}_0 \in \bigcap_{(s,f_0) \in [\![Q^0]\!]} [\![J^0]\!] \ \wedge \ \exists \dot{\mu}_i \in \mathbb{R}^{m_i} \ \wedge \ \big($*si* $(s, f_0) \in [\![P]\!]$ *entraîne* $\mu = \Sigma \chi_{I_i}(\Lambda_i \nu + b_i)$ *alors* $\dot{\mu} = \Sigma \chi_{I_i}(\Lambda_i \dot{\nu})\big)\}$

$$
\begin{aligned}
\pi^0(s, \nu) &= \{(f, \mu) \in F_t | \ \exists (f_0, \mu_0) = (f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n), \ \forall i \in [0, n], \ \exists \dot{\mu}_i \in \mathbb{R}^{m_i}, \ (s_i, \nu_i, f_i, \mu_i) \\
&\qquad \in [\![A_i']\!] \ \wedge \ (s_i, \dot{\nu}_i, f_i) \in [\![\dot{A}_i']\!] \ \wedge \ (s_i, \nu_i, f_i, \mu_i = \Sigma \chi_{I_j}(\Lambda_j \nu_i + b_j) \in [\![A_i']\!] \implies \\
&\qquad \dot{\mu}_i = \Sigma \chi_{I_j}(\Lambda_j \dot{\nu}_i)\} \\
&= \{(f, \mu) \in F_t | \ \forall i \in [1, n], \ (s, \nu, f, \mu) \in [\![\exists (V'_{F_i} \cup H'_{F_i}).A_i']\!] \ \wedge \ (s, \dot{\nu}, f) \in [\![\exists V'_{F_i}.\dot{A}_i']\!] \\
&\qquad \text{et } (s, \nu, f, \mu) \in [\![\exists_{i=1..n} (V'_{F_i} \cup H'_{F_i}).A_0']\!] \ \wedge \ (s, \dot{\nu}, f) \in [\![\exists_{i=1..n} V'_{F_i}.\dot{A}_0']\!]\} \\
&= \pi^1(s, \nu)
\end{aligned}
$$

*On prouve ensuite que* $T^0 = T^1$. *Le théorème de préservation de la bisimulation par opérateur de restriction de priorité 4 page 76 s'étend au cas hybride puisque les priorités interviennent sur la relation de transition et non sur les dérivées. Il suffit alors que la relation* $T_{\mathcal{N}}$ *explicitée dans la définition 33 page 104 est la même que la relation de transitions* $M'$ *donnée dans la définition 34 page 106.*

*On prouve que* $T_{\mathcal{N}} \subseteq M'$. *Soit* $\langle (s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n), f, \mu, , e, (s_0', \nu_0', s_1', \nu_1', \cdots, s_n', \nu_n'), f', \mu' \rangle \in T^0$. *Il y a deux possibilités :*

1. *le cas discret* $e = (e_0, e_1, \cdots, e_n)$ *est le même que dans la démonstration 4.3.2 ;*
2. *le cas continu* $e = \delta \in \mathbb{T}$ *est différent : par définition 33,* $\exists (f_0, \mu_0) = (f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n) \in \pi_0(s_0, \nu_0)$, $\exists (f_0, \mu_0') = (f, \mu', f_1, \mu_1', \cdots, f_n, \mu_n') \in \pi_0(s_0, \nu_0')$, *tel que* $\forall i \in [0, n], (s_i, \nu_i, f_i, \mu_i, \delta, s_i, \nu_i', f_i, \mu_i') \in T_i \ \wedge \ \exists \overline{v}_i, (s_i, f_i, \overline{v}_i) \in [\![\dot{A}_i]\!] \ \wedge \ \nu_i' = \nu_i + \delta \times \overline{v}_i \ \wedge \ \forall \delta' \leq \delta, \exists \mu_{\delta'}, \ (f_i, \mu_{\delta'}) \in \pi_i(s_i, \nu_i + \delta' \times \overline{v}_i)$.

   *On a bien* $(s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n, f, \mu) \in [\![A]\!]$. *On pose* $\overline{v} = (\overline{v}_0, \cdots, \overline{v}_n) \in \mathbb{R}^{\Sigma n_i}$, *alors* $(s_0, s_1, \cdots, s_n, f, \overline{v}) \in [\![\dot{A}]\!]$, $(s_0, s_1, \cdots, s_n) = (s_0', s_1', \cdots, s_n')$, $(\nu_0', \nu_1', \cdots, \nu_n') = (\nu_0, \nu_1, \cdots, \nu_n) + \delta \times \overline{v}$ *et* $\forall \delta' \leq \delta, \ \exists \mu_{\delta'}, \ (f, \mu_{\delta'}) \in \pi(s, \nu + \delta' \times \overline{v})$. *Donc* $\langle (s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n), f, \mu, , e, (s_0, \nu_0', s_1, \nu_1', \cdots, s_n, \nu_n'), f, \mu' \rangle \in M'$

*Pour la réciproque, seules les transitions continues nous intéressent. Soit* $\langle (s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n), f, \mu, , e, (s_0, \nu_0', s_1, \nu_1', \cdots, s_n, \nu_n'), f, \mu' \rangle \in M'$, *on a* $(s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n, f, \mu) \in [\![A]\!]$, $\exists \overline{v}$ *tel que* $(s_0, \cdots, s_n, f, \overline{v}) \in [\![\dot{A}]\!]$, $(\nu_0', \nu_1', \cdots, \nu_n') = (\nu_0, \nu_1, \cdots, \nu_n) + \delta \times \overline{v}$ *et* $\forall \delta' \leq \delta, \ \exists \mu_{\delta'}, \ (f, \mu_{\delta'}) \in \pi(s, \nu + \delta' \times \overline{v})$.

$(s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n, f, \mu) \in [\![A]\!]$ *entraîne* $\exists \mu_1, \cdots, \mu_n$ *tel que* $(s_0, \nu_0, s_1, \nu_1, \cdots, s_n, \nu_n, f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n) \in [\![\bigwedge A_i]\!]$.

$(s_0, \cdots, s_n, f, \overline{v}) \in [\![\dot{A}]\!]$ *entraîne* $\exists \dot{\mu}_1, \cdots, \dot{\mu}_n \in \mathbb{R}^m$ *fonction de* $\overline{v}$. *Donc* $(f, \mu, f_1, \mu_1, \cdots, f_n, \mu_n) \in \pi_0(s_0, \nu_0, \dot{\nu}_0)$.

*On projette les composantes du vecteur* $\overline{v}$ *sur chaque sous composant pour avoir les comportements locaux.*

□

Les résultats obtenus sur Timed AltaRica se transportent dans le cas hybride : ainsi, Hybrid AltaRica est un langage hiérarchique dont tout nœud peut être automatiquement réécrit en un

composant hybride. Un autre résultat s'applique au cas hybride : il s'agit de la traduction d'un composant hybride en un automate hybride. Dans la prochaine partie, nous montrons comment traduire un automate hybride en un composant hybride et réciproquement.

## 5.3 Hybrid AltaRica et les automates hybrides

Dans cette section, nous nous intéressons à faire le lien entre l'extension hybride d'AltaRica et les automates hybrides. La traduction d'un automate hybride linéaire avec condition de flot rectangulaire en un composant hybride est assez directe, de même que dans le cas Timed AltaRica. Le contraire est un peu plus ardu puisqu'il faut résoudre le problème des variables de flux. En chaque état de contrôle, il faut évaluer la loi d'évolution de ces variables en fonction des lois induites par les invariants temporels.

### 5.3.1 Traduction d'un automate hybride en un modèle Hybrid AltaRica

Cette traduction est assez naturelle. Les états de contrôle sont encodés par une variable entière. Soit $\mathcal{H} = (Q, E, X, q_0, init, I, F, \rightarrow)$ un automate hybride avec $Q = \{q_0, \cdots, q_n\}$, on construit le composant hybride $\mathcal{H}_{TAR} = \langle V_S, X, \emptyset, \emptyset, E, A, \dot{A}, M\rangle$ comme suit :

1. $V_S = \{s \in [0, n]\}$ est réduit à une variable,
2. l'assertion ne contient que des invariants temporels $A \equiv \bigwedge_i s = i \implies I(q_i)$ et les conditions d'évolution $\dot{A} \equiv \bigwedge_i s = i \implies F(q_i)$,
3. la formule initiale est $s = 0$ et toutes les horloges à zéro,
4. pour toute transition $(q_i, g, e, r, q_j) \in\rightarrow$, on a $s = i \wedge g| - e- > s := j, r$ dans $M$.

Ces deux systèmes sont équivalents : à tout $(q_i, \nu)$ on associe $(s = i, \nu)$ et réciproquement. Pour toute transition dans $\mathcal{H}$ issue de $(q_i, \nu)$ étiquetée $e$ menant à $(q_j, r(\nu))$, il en existe une dans $\mathcal{H}_{TAR}$ de $(s = i, \nu)$ étiquetée $e$ dans $(s = j, r(\nu))$. Ces deux systèmes sont bisimilaires temporellement.

**Exemple 36** *On reprend l'exemple du thermostat donné à la Figure 3.1 page 40. On modifie la condition d'évolution $\dot{x} = -x + 4$ en $\dot{x} \in [-4, 0]$ afin d'avoir un automate hybride linéaire avec conditions de flots rectangulaires.*

```
node thermostat
 state s :[0,1]; x : analog;
 event e
 trans s=0 |- e -> s:=1;
    s=1 |- e -> s:=0;
 tinvariant (s=0) => ({dx} <=4 & {dx}>=0);
    (s=1) => ({dx} <=0 & {dx}>= 4);
    (s=0) => (x <=Tmax);
    (s=1) => (x >=Tmin);
edon
```

### 5.3.2 Traduction d'un modèle Hybrid AltaRica en un automate hybride

On considère un composant Hybrid AltaRica $\mathcal{H}_{ARH} = \langle V_S, H_S, V_F, H_F, E, A, \dot{A}, M\rangle$ avec $|H_S| = n$ et $|H_F| = m$. On adapte l'Algorithme symbolique 4.4 page 90. On propose un calcul symbolique pour calculer les ensembles suivants :

**Etats de contrôle :** on regroupe les valuations discrètes vérifiant les mêmes invariants temporels et **les mêmes conditions d'évolution**. Pour cela,

1. de même que dans l'Algorithme 4.4, on a transformé $A$ en $A' \equiv \bigwedge_{j=1}^{p} P'_j \implies I'_j$ de sorte que les formules discrètes $P'_j$ soient disjointes deux à deux.
2. on réalise la même opération pour $\dot{A}$, c'est-à-dire on transforme $\dot{A}$ en $\dot{A}' \equiv \bigwedge_{j=1}^{l} Q'_j \implies J'_j$ de sorte que les formules discrètes $Q'_j$ soient disjointes deux à deux.

Alors les états de contrôle sont les ensembles de valuations discrètes $[\![P'_j \wedge Q'_k]\!]$, l'invariant associé est immédiatement $I'_j$. La condition d'évolution est $J'_j$ et les valeurs admissibles des dérivées des variables de flux dépendant linéairement des variables d'états.

**Transitions :** de même que dans l'Algorithme 4.4, les transitions sont énumérées et chaque variable continue de flux est automatiquement remise à jour comme combinaison linéaire des remises à jour de $X$ (cf paragraphe page 100). Eventuellement elle est remise à jour dans $\mathbb{R}_{\geq 0}$ si elle est indépendante de $X$ et non contrainte.

**Donnée :** $\mathcal{H}_{ARH} = (V_S, H_S, V_F, H_F, E, A, M)$ avec $A \equiv \bigwedge_{j=1}^{p} P_j \implies I_j$, $\dot{A} \equiv \bigwedge_{j=1}^{l} Q_j \implies J_j$, $V_F = \{f_1, \cdots, f_l\}$ et $H_F = \{Y_1, \cdots, Y_m\}$
**Sortie :** $\mathcal{H} = (Q, E, X, q_0, init, I, F, \to)$
**Init :** on pose $Q = \emptyset$, $\to = \emptyset$, $X = H_S \cup H_F$
on pose $Flux = (f_1, \cdots, f_l)$ et $\vec{Y} = (Y_1, \cdots, Y_m)$
**pour tout** $T, F \subseteq [1,p], G, U \subseteq [1,l], T \cap F = \emptyset, T \cup F = [1,r], G \cap U = \emptyset, G \cup U = [1,l]$ **faire**
$l_{T,G}^{F,U} = A_Z \wedge \left(\bigwedge_{j \in T} P_j\right) \wedge \left(\bigwedge_{j \in F} \neg P_j\right) \wedge \left(\bigwedge_{j \in G} Q_j\right) \wedge \left(\bigwedge_{j \in U} \neg Q_j\right)$ {on construit les états de contrôle}
$\vec{Y} = \Sigma \chi_{K_i}(\Lambda_i X + b_i)$
**si** $G \neq \emptyset \wedge \bigwedge_{j \in G} J_j \neq ff \wedge \dot{\vec{Y}} \neq \emptyset$ **alors**
$Q = Q \cup \{l_{T,G}^{F,U}\}$
$F(l_{T,G}^{F,U}) = \bigwedge_{j \in G} J_j \wedge \dot{\vec{Y}} \in \mathbb{R}_{\geq 0}^{m} \wedge \dot{\vec{Y}} \in \Sigma \chi_{K_i}(\Lambda_i \dot{X})$ avec $\dot{X}_i = proj_{dx_i}(\bigwedge_{j \in G} J_j)$,
si $T \neq \emptyset$, $I(l_{T,G}^{F,U}) = \bigwedge_{k \in T} I_k$, sinon $I(l_{\emptyset,G}^{[1..p],U}) = tt$ {on évalue les invariants}
**fin si**
**fin pour**
**pour tout** $q, q' \in Q$ **faire**
**pour tout** $t = ((g, \gamma), e, (a, R)) \in M$ **faire**
$\to = \to \cup \left(q, (g \wedge Pre_t(q') \wedge \gamma, e, (a, R, \vec{Y} = \Sigma \chi_{K_i}(\Lambda_i X + b_i))), q'\right)$
**fin pour**
**fin pour**

**Algorithme 5.1:** Algorithme symbolique de traduction d'un programme Hybrid AltaRica en automate hybride

L'Algorithme 5.1 est symbolique donc ne s'appuie que sur la syntaxe d'un composant hybride et termine toujours puisque toutes les opérations sont réalisées sur des ensembles finis.

**Proposition 8** *Etant donné un composant Hybrid AltaRica $\mathcal{H}_{ARH} = (V_S, H_S, V_F, H_F, E, A, \dot{A}, M)$ , l'automate hybride $\mathcal{H} = (Q, E, X, q_0, init, I, F, \to)$ obtenu à partir de l'Algorithme 5.1 est bisimilaire temporellement.*

**Preuve de la proposition 8**. *On note $[\![\mathcal{H}]\!] = \langle E_t, F_t, S_t, \pi, T \rangle$ et on considère la relation de bisimulation $R \subseteq (S_t \times F_t) \times (Q \times \mathcal{U}(X) \times \mathcal{C}(\dot{X}))$ qui associe à une configuration un triplet (état de contrôle, valuation d'horloge satisfaisant l'invariant,vecteur vitesse satisfaisant la condition d'évolution).*

*Soit une configuration* $(s, \nu, f, \mu) \in S_t \times F_t$*, alors* $\exists \dot{\nu}, \dot{\mu}$ *et* $(f, \mu) \in \pi(s, \nu)$*. Comme* $(s, f) \in V_S \times V_F$*,* $\exists T_1, F, G, U$ *tels que* $(s, f) \in [\![ l^{F,U}_{T_1,G} ]\!]$ *par construction de l'automate hybride, puis par respect des invariants* $(\nu, \mu) \in [\![ I(l^{F,U}_{T_1,G}) ]\!]$ *et de la condition d'évolution* $(\dot{\nu}, \dot{\mu}) \in [\![ F(l^{F,U}_{T_1,G}) ]\!]$*.*

*Respectivement, si on considère un état de contrôle* $q$*, une valuation d'horloge* $\nu$ *et une dérivée* $(\dot{\nu}, \dot{\mu})$*,* $\exists T_1, F, G, U,\ q = l^{F,U}_{T_1,G}$*. On choisit une valuation dans* $(s, f) \in q$ *alors nécessairement* $(s, f) \in S_t \times F_t$*,* $(\dot{\nu}) \in [\![ \dot{A} ]\!]$ *et* $\dot{\nu}$ *dérive d'un invariant temporel et des dérivées* $\dot{\nu}$*.*

*Ensuite, il reste à montrer que pour toute transition issue d'une configuration de* $(s, f) \in S_t \times F_t$ *il existe une transition étiquetée par le même événement issu de tout couple* $(q, \nu)$ *tel que* $((s, f), (q, \nu)) \in R$ *dont les buts sont également en relation, et réciproquement. Par construction, c'est évident.* □

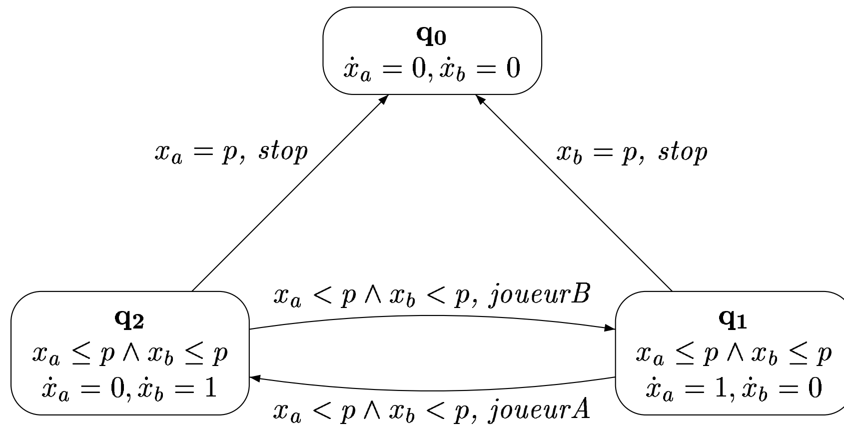
**Exemple 37 (Jeu de Marienbad - version blitz - IV)** *On reprend le nœud hybride* Arbitre_B *représenté à la Figure 5.2 page 97. Nous le traduisons en automate hybride selon l'Algorithme 5.1. Il y a un invariant temporel* `(AvaJouer<=1) => (xa <= p & xb <= p);` *et trois conditions d'évolution*

```
(AvaJouer=0) => ({dxa}=1 & {dxb}=0);
(AvaJouer=1) => ({dxa}=0 & {dxb}=1);
(AvaJouer=2) => ({dxa}=0 & {dxb}=0);
```

*Ainsi,* $p = 1$ *et* $l = 2$*, d'où :*

| partition | | état de contrôle | | invariant | flot |
|---|---|---|---|---|---|
| $T = \emptyset$<br>$G = \{3\}$ | $F = \{1\}$<br>$U = \{1,2\}$ | $AvaJouer \neq 0,1 \wedge AvaJouer = 2$<br>$\equiv AvaJouer = 2$ | <br>$\equiv q_0$ | $tt$ | $\dot{x}_a = 0$<br>$\dot{x}_b = 0$ |
| $T = \{1\}$<br>$G = \{1\}$ | $F = \emptyset$<br>$U = \{2,3\}$ | $AvaJouer = 0,1 \wedge AvaJouer = 0$<br>$\equiv AvaJouer = 0$ | <br>$\equiv q_1$ | $x_a \leq p$<br>$x_b \leq p$ | $\dot{x}_a = 1$<br>$\dot{x}_b = 0$ |
| $T = \{1\}$<br>$G = \{2\}$ | $F = \emptyset$<br>$U = \{1,3\}$ | $AvaJouer = 0,1 \wedge AvaJouer = 1$<br>$\equiv AvaJouer = 1$ | <br>$\equiv q_2$ | $x_a \leq p$<br>$x_b \leq p$ | $\dot{x}_a = 0$<br>$\dot{x}_b = 1$ |

*On parcourt ensuite l'ensemble des transitions spécifiées dans le composant et on compare les extrémités (origine et but) aux états de contrôle calculés précédemment. La première transition est* `AvaJouer=0 & (xa<p & xb<p)` *- joueurA -¿ AvaJouer := 1;—. Le seul état atteint par cette transition* $t_1$ *est* $q_2$ *et* $pre_t(q_2) = x_a \leq p \wedge x_b \leq p$*, d'où* $t_1$ *génère la transition* $(q_1, xa < p \wedge xb < p, joueurA, (1, 0), q_2)$*. Et ainsi de suite, finalement, on obtient alors l'automate hybride suivant :*



### 5.3.3 Résultats de décidabilité des modèles Hybrid AltaRica

Le pouvoir d'expression du langage Hybrid AltaRica est celui des automates hybrides linéaires avec conditions de flot rectangulaires [HM00] d'après les deux Algorithmes de traduction précédents. En effet, si dans un état de contrôle la loi d'évolution est définie par morceaux pour une variable de flux, on divise l'état en autant d'états que de sauts et on rajoute des transitions pour représenter l'évolution du temps comme cela est fait dans l'exemple 32 page 102.

Les résultats de décidabilité, de même que pour Timed AltaRica, dépendent essentiellement des variables de flux : la condition d'initialisation ne peut être vérifiée que pour les variables continues d'états. L'initialisation pour une variable de flux ne peut être montrée que si la variable de flux conserve toujours la même dérivée quelle que soit la configuration ou si quand la variable est contrainte par l'invariant temporel à être une combinaison linéaire de variables d'états (qui elles sont contrôlables), i.e. la variable de flux appartient au composant hybride.

| Transitions discrètes ($\neq \varepsilon$) | Invariants temporels | |
|---|---|---|
| Variables d'états $\{x_i\}$ | rectangulaires et $Y = \Sigma\chi_{I_i}(\Sigma a_i x_i + b_i)$ | $Y$ quelconque |
| rectangulaires, initialisées et indépendance des remises à jour non déterministes et des flux | Décidable | Indécidable |
| quelconques | Indécidable | |

FIG. 5.7 – Résultats d'accessibilité des modèles Hybrid AltaRica

Nous avons proposé un langage Hybrid AltaRica : ce langage hiérarchique de modélisation hybride de haut niveau est basé sur le modèle des automates à contraintes hybrides. L'expressivité de ce langage est celle des automates hybrides linéaires avec conditions de flot rectangulaires. Contrairement au cas temporisé, on ne connaît pas les propriétés décidables sur ce modèle uniquement en regardant la syntaxe. Dans certains cas, il faut chercher des conditions de réinitialisation dans les boucles. Nous avons produit un algorithme de traduction d'un composant hybride en un automate hybride.

Dans cette deuxième partie, nous avons étendu les modèles AltaRica afin qu'ils manipulent quantitativement des grandeurs évoluant en fonction du temps qui passe. Le premier langage Timed AltaRica, en sus des caractéristiques classiques des modèles AltaRica, intègre des priorités temporelles. Le deuxième langage Hybrid AltaRica propose des modèles dont le pouvoir d'expression est plus riche.

Dans la prochaine partie, nous présentons l'implantation d'un outil pour Timed AltaRica : ce prototype, Tarc, prend en entrée des modèles Timed AltaRica, les met à plats et les transforme en automates temporisés en format UPPAAL et HYTECH. Le langage Hybrid AltaRica n'est pas encore implanté car la prise en charge des dérivées est plus compliquée.

# Troisième partie

# De la théorie à la pratique

# Chapitre 6

# Un outil pour Timed AltaRica : Tarc

Dans cette partie nous nous intéressons à l'implantation d'un prototype, Tarc, dont l'objectif est de générer des automates temporisés à partir de modèles Timed AltaRica. Le langage Timed AltaRica et ses caractéristiques ont été amplement étudiés dans le Chapitre 4. L'outil est l'implantation du procédé 29 de réécriture et de l'Algorithme 4.4 symbolique de traduction d'un composant temporisé en un automate temporisé. Pour cela, nous avons repris des modules de l'outil Mec V [ABC94, Arn95, Vin03a, Vin03b] qui implante, entre autre, le procédé de réécriture dans le cas non temporisé.

Dans ce chapitre, après avoir rappelé quelques généralités sur la vérification, nous présentons le model-checker Mec V dans la section 6.2 puis nous détaillons le codage des modèles Timed AltaRica, leur mise à plat et leur réécriture dans la section 6.3.

Dans le prochain chapitre, nous utilisons l'outil Tarc afin d'étudier des propriétés sur un système complet.

## 6.1 Rappels sur le model-checking

Dans la partie précédente, nous avons construit un langage de modélisation de haut niveau, la deuxième étape de l'analyse des systèmes consiste à assurer une certaine fiabilité sur les modèles obtenus. Nous avons défini la notion de vérification [Pnu81, CWA$^+$96, CGL94, SBB$^+$99, CGP99] dans l'Introduction page 10 et nous avons présenté plusieurs model-checkers dans les sections 3.1.4 et 3.2.4.

Nous allons appliquer cette méthode formelle sur nos modèles Timed AltaRica : nous allons donc traduire nos modèles dans le format de model-checkers existants : Uppaal et Hytech (en prévision de l'implantation du cas hybride).

La propriété $\phi$ à vérifier est exprimée dans une *logique temporelle* [Pnu77, SBB$^+$99, CGP99]. Pratiquement une logique temporelle est une logique à laquelle on ajoute la notion de temps. Le temps décrit dans la partie 3.1.1 n'est pas explicitement manipulé dans le cas des logiques temporelles et se résume essentiellement à des relations de précédence. On exprime des propriétés comme *si une propriété P est vérifiée, alors plus tard Q sera vérifiée*, ce qui s'écrit $\forall t,\ P(t) \implies \exists t' > t,\ Q(t)$. Il est possible de manipuler le temps quantitatif à l'aide de *logiques temporelles temporisées* : ce sont des logiques temporelles étendues avec des primitives permettant d'exprimer des conditions sur les dates auxquelles les événements se produisent. On peut alors exprimer des propriétés comme : *si une propriété P est vérifiée, alors dans moins de n unités de temps Q sera vérifiée*, ce qui s'écrit $\forall t,\ P(t) \implies \exists t < t' \leq t + n,\ Q(t')$.

On distingue deux grandes familles de logiques temporelles (resp. temporisées) dépendant de la nature du temps : celles en temps linéaire dont *LTL* [Pnu81, SBB+99] (resp. *TLTL* [AH89]) et celles en temps arborescent dont *CTL [EH82, SBB+99]*, *$\mu$-calcul [CGP99, AN01]* (resp. *TCTL [AD90]*, *T-$\mu$-calcul [HNSY92, LLW95]*). Nous rappelons les résultats de décidabilité du model-checking dans le cas temporisé dans le Tableau 6.1.

| | Complexité (temporisée) |
|---|---|
| Accessibilité | PSPACE complet |
| TCTL | PSPACE complet |
| TLTL | si $\mathbb{T} = \mathbb{R}$, indécidable<br>si $\mathbb{T} = \mathbb{N}$, EXPSPACE |
| $\mu$-calculus | ? |

FIG. 6.1 – Résumé des résultats connus

## 6.2 Mec V

Dans cette partie, nous développons quelques aspects d'implantation de l'outil Mec V réalisé par Aymeric VINCENT [Vin03b]. Nous nous attachons particulièrement à la manière dont il manipule les modèles AltaRica puisqu'il s'agit des programmes que nous réutiliserons.

### 6.2.1 Fonctionnement

Les caractéristiques du model-checker Mec V sont détaillées dans [GV03, Vin03a]. Le fonctionnement de Mec V est schématisé dans la Figure 6.2. Les spécifications prises en entrée sont des modèles AltaRica et la logique utilisée est du $\mu$-calcul relationnel. On peut alors calculer des relations sur les modèles AltaRica en entrée : les relations de transitions des systèmes mis à plat, les configurations accessibles depuis les configurations initiales ... Plus généralement, on exprime des relations sur les configurations ou sur les événements des différents nœuds ainsi que sur les types de base (booléens, intervalles, énumérations).

La logique utilisée est le $\mu$-calcul sur des relations avec quantificateurs du premier ordre et égalité. Une relation peut être vue comme une solution d'un système d'équations de points fixes avec un nombre quelconque d'alternances de plus petit et plus grand points fixes. Cette logique permet d'exprimer directement et simplement la relation de bisimulation interfacée [GV03]. Le model-checker est donc très expressif et peut vérifier les propriétés de logiques temporelles de $CTL^\star$ [Dam94]. Pour toute formule de $CTL^\star$, il faudra néanmoins la traduire "à la main", pour l'instant, en $\mu$-calcul ce qui sera parfois difficile, notamment dans le cas de LTL.

L'algorithme utilisé pour vérifier si un modèle satisfait des formules de $\mu$-calcul implanté n'est pas l'algorithme classique de marquage des états [AC88, CS92] comme dans MEC 4. Dans Mec V, les relations et les états vérifiant les sous propriétés sont stockés dans une structure de données particulièrement efficace : les *BDDs* [Bry86].

### 6.2.2 Structure de données : les BDDs [Bry86]

Pour représenter un système, on code les états et/ou la relation de transitions. Un tel codage rencontre vite des problèmes d'espace car rapidement la taille de l'espace d'états est très importante. Une solution est apparue dans la représentation symbolique de l'ensemble des états et de l'ensemble

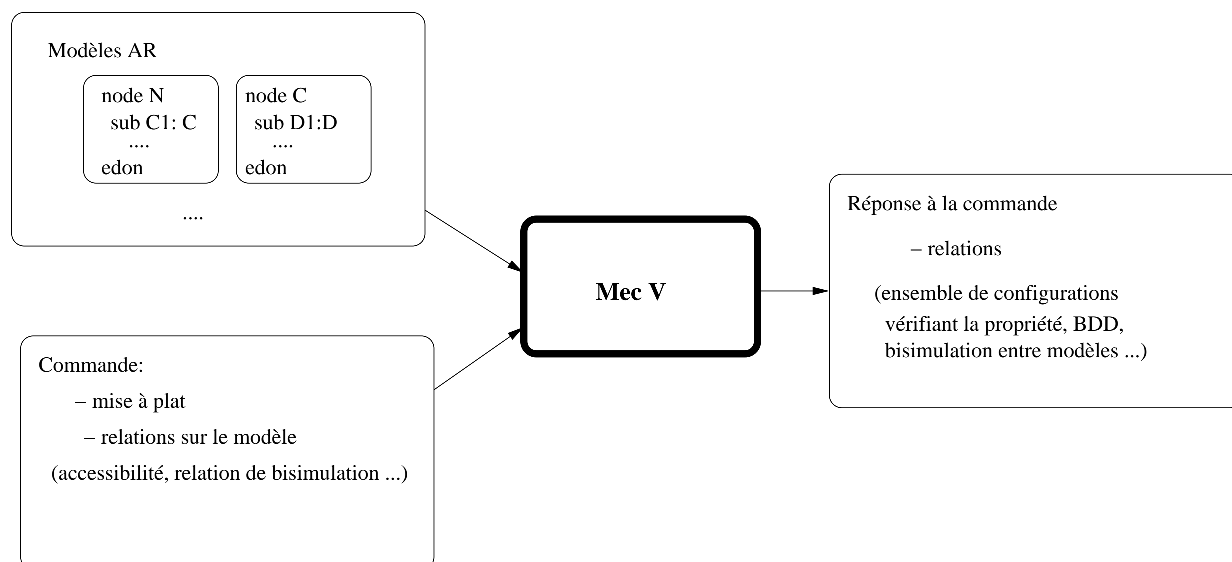



FIG. 6.2 – Fonctionnement de Mec V

des transitions. Ainsi, on regroupe un ensemble d'états satisfaisant une même formule (cette formule pouvant par exemple être exprimée comme un point fixe). Ces représentations symboliques doivent ensuite pouvoir être utilisées dans les Algorithmes : être stables par *intersection, union, Pre, Post* et on doit pouvoir décider si deux représentations correspondent au même ensemble.

Le BDD est apparu dans [Bry86] à la fois efficace et facile à implanter. Il permet les opérations de base nécessaires à l'implantation des algorithmes de vérification (en temps polynômial ou quadratique). Il est utilisé dans Mec V. Nous présentons brièvement ce formalisme puisque nous coderons les expressions de Timed AltaRica avec cette structure de données.

On considère un ensemble de variables booléennes $Z = \{x_1, \cdots, x_n\}$. On peut représenter toute fonction booléenne $f(x_1, \cdots, x_n)$ par un BDD : il s'agit d'un graphe acyclique orienté avec une racine et des sommets de deux types, *terminaux*, c'est-à-dire 0 ou 1, ou *non terminaux*. Dans la pratique on n'utilise pas les BDDs mais les ROBDDs [And98] (ordered binary decision diagram) car ils admettent une représentation unique une fois l'ordre des variables choisi.

La librairie des ROBDDs dont nous nous servons a été entièrement écrite par Aymeric VINCENT [Vin03a].

### 6.2.3 Les modules de mise à plat

La partie algorithme de Mec V qui nous intéresse est l'implantation du théorème 2 de réécriture d'un nœud AltaRica en un composant AltaRica, page 33. Nous reprenons dans la Figure 6.3 une partie du schéma de l'architecture de Mec V réalisé dans [Vin03a]. L'ensemble des modules décrits correspond à la prise en compte d'un modèle AltaRica et sa mise à plat. Ces modules sont réutilisés, avec éventuellement quelques modifications mineures, dans Tarc.

Dans la Figure 6.3, le sommet AltaRica regroupe l'analyse lexicale, l'analyse syntaxique et la gestion de l'arbre syntaxique. Le module *ar-node* transforme l'arbre syntaxique en une structure de données encodant le modèle AltaRica, c'est-à-dire :

- les variables du modèles sont représentées par un emplacement regroupant les informations (nom de la variable, type de la variable, flux ou état, indice numérotant la variable). Cette structure de données est gérée par le module *context* et les types des variables par le module

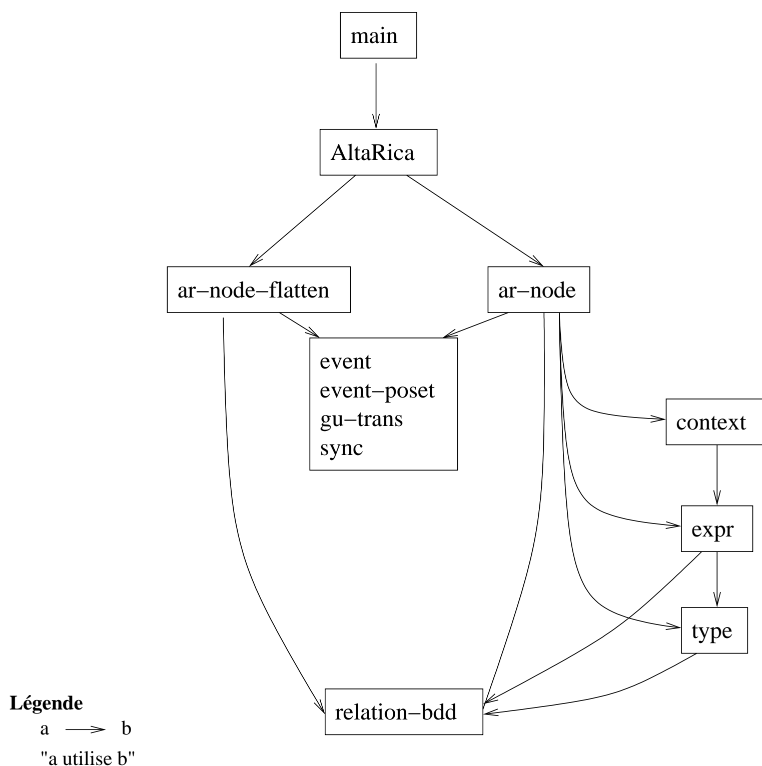



FIG. 6.3 – Partie de l'architecture de Mec V

*type* ;
- les contraintes du modèle, comme l'assertion ou la condition initiale, sont des expressions. La structure de données pour les expressions est gérée par le module *expr* ;
- la relation de transitions est stockée dans un BDD et est assez complexe à encoder. La relation de transitions locale est gérée par les modules intermédiaires :

  1. *event* associe à un événement l'ensemble des transitions étiquetées par cet événement ;
  2. *event-poset* gère les relations d'ordre induites par les priorités et les broadcasts entre les événements et les vecteurs d'événements ;
  3. *gu-trans* est la structure de données stockant la garde et l'affectation de chaque transition ;
  4. *sync* crée les vecteurs d'événements produits par la hiérarchie et les synchronisations.

  Ensuite, le module *ar-node-flatten* est appelé récursivement pour actualiser la relation de transitions prenant en compte celle des sous nœuds ;
- le module *relation-bdd* est au cœur du model-checker puisque il gère la structure de données BDDs. Les structures précédentes (expression et gu-trans) sont mises sous forme de BDDs et on utilise les opérations BDD pour calculer les relations.

### 6.2.4 Mise à plat des modèles AltaRica

Dans cette section, nous détaillons l'implantation du théorème 2 de réécriture dans Mec V. Une description détaillée et plus technique se trouve dans la thèse d'Aymeric VINCENT [Vin03a]. Pratiquement, un nœud AltaRica est représenté dans Mec V par quatre données :

- le type de ses configurations ($c$) : il s'agit du produit cartésien des variables ;
- le type des vecteurs d'événements ($e$) : il s'agit du produit cartésien du type des événements locaux et du type des vecteurs d'événements des sous-nœuds ;
- la relation de transition ($\subseteq c \times e \times c$) : c'est un BDD dont la hauteur maximale en nombre de bits est $log_2(2|c| + |e|)$ ;
- la condition initiale ($\subseteq c$).

La mise à plat consiste à calculer la relation de transitions et la condition initiale du nœud. Pour cela, on s'intéresse aux quatre données définies ci-dessus et Mec V réalise les étapes suivantes pour les évaluer :

1. la mise à plat des sous-nœuds est réalisée récursivement et de la mémoire est allouée dans le nœud courant pour toutes les variables apparaissant dans les sous-nœuds ;
2. le type configuration $c$ est calculé lors du parcours des sous-nœuds et est fonction des assertions : les variables locales utiles sont concaténées à $c$. Si dans l'assertion, deux variables sont toujours égales, une seule est représentée dans la configuration. La taille d'une variable dans $c$ est fonction de son domaine de définition ;
3. l'état initial est la conjonction des états initiaux spécifiés dans le nœud et des conditions initiales des variables des sous-nœuds non spécifiées dans le nœud ;
4. pour chaque événement spécifié, la relation de transitions locale associée à cet événement est calculée : il s'agit de l'union des BDDs des transitions étiquetées par l'événement. La relation de transitions locales est alors la réunion des BDDs associés à chacun des événements ;
5. l'événement local $\epsilon$ est traité, c'est-à-dire que le BDD de l'événement est généré par respect pour l'assertion et est ajouté à la relation de transitions ;
6. tous les vecteurs d'événements sont calculés, de même que les vecteurs d'instanciation dus aux broadcast ;
7. la dernière étape consiste à intégrer la relation de transitions des sous-nœuds en fonction des vecteurs d'événements. A un vecteur d'événement on associe l'intersection des relations de transitions locales des sous-nœuds liées à l'événement spécifié dans le vecteur. Cette relation devient la relation de transitions locale à l'événement local du nœud parent. La relation liée à cet événement est l'union des BDDs des vecteurs dans lesquels il apparaît ;
8. les priorités sont résolues.

**Exemple 38 (Jeu de Marienbad)** *Nous illustrons en partie le procédé de mise à plat implanté dans* Mec V *sur le nœud* Marienbad *donné dans l'Exemple 5 page 33. L'ensemble des variables est :*

```
local variables
sub variables
        state AvaJouer : bool;
        flow L1.N : [ 0, 7 ];
        state L1.n : [ 0, 7 ];
        flow L2.N : [ 0, 7 ];
        state L2.n : [ 0, 7 ];
        flow L3.N : [ 0, 7 ];
        state L3.n : [ 0, 7 ];
        flow L4.N : [ 0, 7 ];
        state L4.n : [ 0, 7 ];
```

*Les nombres entiers sont codés selon leur écriture binaire et un bit est ajouté pour le signe. Les variables de flux $L_i.N$ étant chacune égale à une variables d'état, elles ne sont pas encodées dans le BDD. Le type de configuration est alors le suivant :*

| *Variables* | *AvaJouer* | *L1.n* | *L2.n* | *L3.n* | *L4.n* |
|---|---|---|---|---|---|
| *Codage (en bits)* | *1* | *4* | *4* | *4* | *4* |

*La configuration initiale est alors* 0 0001 0011 0101 0111.

*On calcule la relation de transitions locale de* Marienbad *: le seul événement est* $\varepsilon$. *Toutes les transitions sont possibles donc le BDD est l'identité. Le vecteur* $\varepsilon$ *est toujours codé par* 0. *Dans le composant temporisé* Arbitre*, il y a 3 événements (*joueurA, joueurB *et* $\varepsilon$*), il faut 3 bits pour les énumérer. Le type de vecteurs d'événements est alors :*

| *Composants* | *local* | *R* | *L1* | *L2* | *L3* | *L4* |
|---|---|---|---|---|---|---|
| *Codage (en bits)* | *2* | *3* | *3* | *3* | *3* | *3* |

*Ensuite les vecteurs d'événements possibles sont construits, c'est-à-dire ceux respectant les contraintes de broadcast.*

```
event
  <epsilon,R.joueurA,L1.epsilon,L2.epsilon,L3.epsilon,L4.joue>,
  <epsilon,R.joueurB,L1.epsilon,L2.epsilon,L3.epsilon,L4.joue>,
  <epsilon,R.joueurA,L1.epsilon,L2.epsilon,L3.joue,L4.epsilon>,
  <epsilon,R.joueurB,L1.epsilon,L2.epsilon,L3.joue,L4.epsilon>,
  <epsilon,R.joueurA,L1.epsilon,L2.joue,L3.epsilon,L4.epsilon>,
  <epsilon,R.joueurB,L1.epsilon,L2.joue,L3.epsilon,L4.epsilon>,
  <epsilon,R.epsilon,L1.epsilon,L2.epsilon,L3.epsilon,L4.epsilon>,
  <epsilon,R.joueurA,L1.joue,L2.epsilon,L3.epsilon,L4.epsilon>,
  <epsilon,R.joueurB,L1.joue,L2.epsilon,L3.epsilon,L4.epsilon>;
```

*Considérons par exemple le vecteur :* $< \varepsilon$*,R.joueurA,L1.*$\varepsilon$*,L2.*$\varepsilon$*,L3.*$\varepsilon$*, L4.joue*$>$. *Il sera codé par* 00 001 000 000 000 001. *Pour calculer le BDD associé à ce vecteur, il faut calculer le BDD associé au vecteur R.joueurA. Dans le nœud* Arbitre*, le type des configurations est de* 1 *bit pour la variable booléenne AvaJouer. Il n'y a qu'une transition étiquetée joueurA :*

```
AvaJouer  |- joueurA -> AvaJouer := false;
```

*qui se code localement* 1 001 0 *et est stockée dans un BDD. Cette relation est ensuite étendue dans le type de configuration locale au nœud* Marienbad *:*

| *Codage dans* | *configuration* | *événement* | *configuration* |
|---|---|---|---|
| *Arbitre* | *1* | *001* | *0* |
| *Marienbad* | *1* $x_1$ $x_2$ $x_3$ $x_4$ | *00 001 000 000 000 001* | *0* $x'_1$ $x'_2$ $x'_3$ $x'_4$ |

*où* $x_i$ *et* $x'_i$ *sont codés sur 4 bits et prennent n'importe quelle valeur. On obtient alors le BDD représenté à la Figure 6.10 page 126, il contient 76 états et a été généré par* Mec V.

*De même pour l'événement* joue *dans* L4.Paquet*, on calcule la relation de transitions. Il s'agit de l'union des relations engendrées par les* 7 *transitions. On l'étend au nœud local. Pour les autres paquets, on considère la relation engendrée par la relation* $\varepsilon$ *étendue pour le nœud* Marienbad.

*Alors la relation pour l'événement* 00 001 000 000 000 001 *est l'intersection des* 5 *relations calculées précédemment. La relation de transitions du nœud* Marienbad *est l'union des relations de tous les vecteurs d'événements possibles.*

Nous venons de détailler certains aspects de l'outil Mec V que nous réutilisons dans l'outil de manipulation de programmes Timed AltaRica. Dans la suite, nous présentons cet outil et nous faisons le lien avec les modules de Mec V.

## 6.3 Tarc

Dans cette partie, nous présentons le prototype Tarc réalisé pendant la thèse pour traiter des modèles AltaRica temporisés. Contrairement à Mec V, il ne s'agit pas d'un model-checker mais d'un traducteur de modèles Timed AltaRica vers des model-checker temporisés existants. Ainsi, Tarc prend en entrée des modèles Timed AltaRica et aucune commande, met à plat les modèles et génère un automate temporisé. On peut représenter le fonctionnement général de l'outil dans la Figure 6.4. L'outil est écrit en C et tourne sous Linux.

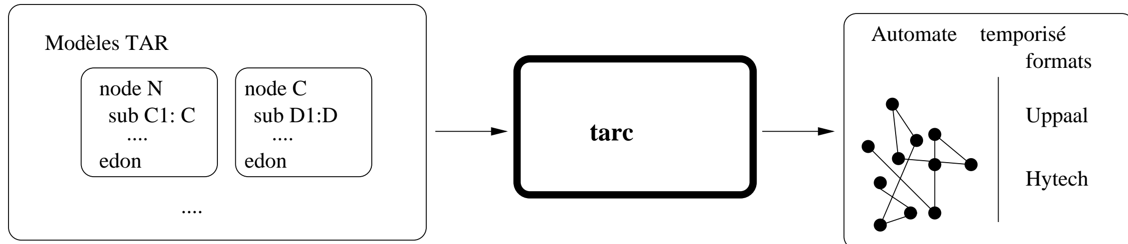



FIG. 6.4 – Fonctionnement de Tarc

### 6.3.1 Fonctionnement

Un modèle Timed AltaRica est pris en entrée de Tarc [Pag03a] et le transforme en automate temporisé de UPPAAL ou HYTECH. L'architecture générale du prototype est représentée dans la Figure 6.5. On retrouve les modules de Mec V présentés dans la section 6.2.3. Les analyseurs lexical et syntaxique sont ceux d'AltaRica étendus selon la grammaire concrète explicitée en Annexe C, avec entre autre les mots clés *clock* et *tinvariant*. L'arbre syntaxique est modifié en conséquence. Ces changements syntaxiques sont regroupés dans le sommet *Timed AltaRica* .

Une structure de données représente un automate temporisé et est gérée par le module *timed-automata*. Ce module crée les structures de données, fournit des fonctions d'accès à des champs de la structure. La structure représentant un automate temporisé contient :

1. le nom, en effet, un automate temporisé est associé à chaque nœud temporisé et on peut traduire n'importe quel sous modèle en automate temporisé,
2. trois champs renferment le noms des variables, des horloges et des vecteurs d'événements ;
3. deux tables de hachage contiennent les BDDs associés aux états de contrôle et les BDDs associés aux invariants temporels. Il y a un champ pour l'état initial ;
4. une sous structure gère les transitions. Une transition consiste en un état source, un état destination, une garde, un affectation des variables et pointe vers la transition suivante.

### 6.3.2 Abstraction

La première étape consiste à transformer un nœud Timed AltaRica en un nœud AltaRica : cette phase est gérée par le module *ar-node-timed*. Pour cela, les variables d'horloge disparaissent et toute formule temporelle élémentaire est remplacée par une nouvelle variable booléenne n'appartenant pas au modèle. La contrainte *tinvariant* devient alors une formule discrète et est incorporée dans l'assertion. Le module *expr-timed* intervient dans la manipulation des formules temporelles.

Formellement, un composant temporisé $\mathcal{A} = (V_S, C_S, V_F, C_F, E, A, M)$ est transformé en un composant $\mathcal{A} = \langle V_S \cup \{@\gamma_j\}_{j\in J}, V_F \cup \{@\gamma_j\}_{j\in K}, E, \tilde{A}, \tilde{M}\rangle$ avec :

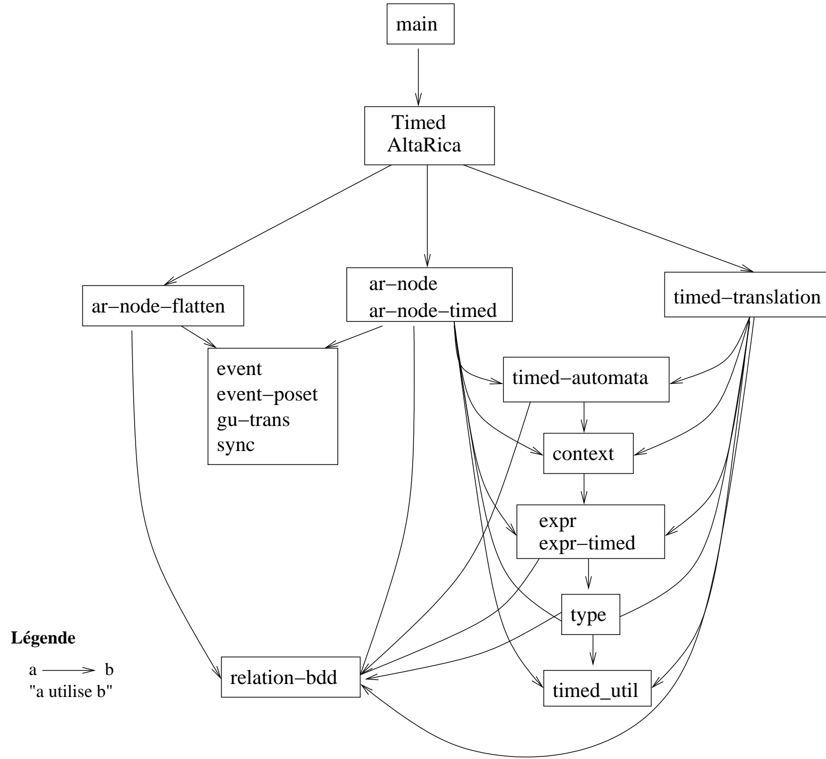



FIG. 6.5 – Architecture de Tarc

1. $@\gamma_i$ sont des variables booléennes n'appartenant pas à $V_S \cup V_F$. Ces variables sont ajoutées lors de la traduction de l'arbre syntaxique en la structure de données *ar-node* : chaque expression élémentaire *exp* contenant une horloge est transformée en $@\gamma_i$. Si $vlib(exp) \subseteq C_F$, alors $@\gamma_i$ est ajouté à $V_F$ et à $V_S$ sinon. Le nombre de variables ajoutées dépend du nombre d'expressions élémentaires distinctes.
2. $J, K$ deux intervalles entiers tels que $J \cap K = \emptyset$, $J$ correspond aux variables $@\gamma_i$ ajoutées à $V_S$ et $K$ à celles ajoutées à $V_F$ ;
3. les contraintes d'horloges sont modifiées comme suit :
   (a) $\big(A = A_Z \wedge \bigwedge P_j \implies I_j\big) \iff \big(\tilde{A} = A_Z \wedge \bigwedge P_j \implies (\wedge_{k \in L_j} @\gamma_k)\big)$ où $L_j \subseteq J \cup K$ et $I_j = \wedge$ formule_atomique ;
   (b) $((g, \gamma), e, (a, R)) \in M \iff (g \wedge \bigwedge_L @\gamma_k, e, a \wedge \bigwedge_T @\gamma_k) \in \tilde{M}$ avec $L \subseteq J \cup K$ et $T \subseteq J$.

**Exemple 39 (Un système de réservoir - VI)** *Reprenons l'exemple du nœud temporisé* Reservoir *de l'exemple 12 page 56. L'étape d'abstraction de ce nœud temporisé génère le nœud AltaRica représenté à la Figure 6.6.*

### 6.3.3 Mise à plat des modèles Timed AltaRica

La mise à plat est entièrement réalisée par Mec V selon le processus décrit dans la partie 6.2.4. Le modèle abstrait établi par l'étape précédente est injecté dans les modules de Mec V et nous obtenons la relation de transitions du système complet sous forme d'un BDD.

Les réécritures dans le cas sans temps (Définition 35 page 137) et dans le cas temporisé (Définition 29 page 84) sont des transformations syntaxiques. On considère un nœud temporisé

```
node Reservoir
        flow R : bool;
        state s : [ 0, 2 ];
        state @gamma0 : bool;
        state @gamma1 : bool;
        state @gamma2 : bool;
        state @gamma3 : bool;
 event marche , arret, fuite;
 trans s=0 & R=true |- marche -> s:=1, @gamma2:=true;
    s=1 & @gamma0 |- arret -> s:=2;
    s!=2 |- fuite -> s:=2, @gamma3:=true;
 assert (s=1) => (@gamma1);
edon
```

| formule atomique | booléen |
|---|---|
| $x = 100$ | $@\gamma_0$ |
| $x \leq 100$ | $@\gamma_1$ |
| $x = 0$ | $@\gamma_2$ |
| $x = 101$ | $@\gamma_3$ |

FIG. 6.6 – Abstraction du nœud *Reservoir*

$\mathcal{N} = \langle V_F, C_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ dont tous les sous-nœuds sont des composants temporisés. Nous avons obtenu pour chaque composant temporisé $\mathcal{N}_i = (V_S^i, C_S^i, V_F^i, C_F^i, E^i, A^i, M^i)$ un composant $\mathcal{A}_i = \langle V_S^i \cup \{@\gamma_j^i\}_{j \in J^i}, V_F^i \cup \{@\gamma_j^i\}_{j \in K^i}, E^i, \tilde{A}^i, \tilde{M}^i \rangle$.

Si on applique la réécriture d'AltaRica, sur le nœud AltaRica $\tilde{\mathcal{N}} = \langle V_F \cup \{@\gamma_j^0\}_{j \in K^0}, E, \emptyset, \mathcal{A}_0, \cdots, \mathcal{A}_n, (\tilde{V}, <_{\tilde{V}}) \rangle$ alors nous obtenons le composant $\tilde{\mathcal{C}_\mathcal{N}} = \langle V_S \bigcup \cup \{@\gamma_j^i\}_{j \in L^i}, V_F \cup \{@\gamma_j^0\}_{j \in K}, E \ , A, M \rangle$ avec $A = \wedge A_Z^i \bigwedge (\wedge P_j^i \implies (\wedge @\gamma_k^i))$ et $(G, e, A) \in M'$ si, et seulement si $G = \wedge g^i \bigwedge \wedge (\wedge @\gamma_k^i)$ et $A = \wedge a_i \bigwedge \wedge @\gamma_k^i$.

On n'a donc perdu aucune information et lorsque les $@\gamma_i$ sont retranscrits sous forme de formules temporelles, nous obtenons un composant temporisé qui est bisimilaire au composant temporisé résultant de la réécriture temporisée. Nous avions déjà appliqué ce résultat dans l'étude du nœud temporisé *Pompe* dans l'exemple 27 page 86. Le domaine *clock* avait été transformé en un domaine entier puis le nœud avait été réécrit syntaxiquement avec l'AltaTool A-SEMANTICS.

### 6.3.4 Génération de l'automate

Nous avons prouvé que l'Algorithme 4.3 page 88 ne termine pas toujours : cela correspond à une transformation de la relation de transitions calculée précédemment en un automate temporisé. Nous allons donc appliquer l'Algorithme 4.4 page 90 sur le `tinvariant` du modèle. Le module *timed-translation* récupère les informations $A_Z \bigwedge \big( \wedge P_j^i \implies (\wedge @\gamma_k^i) \big)$ et génère les états de contrôle de l'automate temporisé associé. Le module suivant l'Algorithme 4.4 page 90, recherche les formules disjointes représentant les plus grands ensembles satisfaisant le même invariant temporel. Le calcul des partitions d'un ensemble en deux éléments est conduit par le module *timed_util*.

Les états de contrôle sont donc représentés sous forme de BDD. Le parcours des transitions se fait en récupérant les transitions représentées dans la relation de transitions évaluées par la mise à plat. Cette étape se fait grâce aux opérations de projection et d'intersection des BDDs.

**Exemple 40 (Un système de réservoir - VII)** *Nous avons vu dans l'exemple 29 page 90 que nous obtenions exactement deux états de contrôle. Nous obtenons exactement le même automate, Figure 6.7 où les formules contenant des horloges sont abstraites.*

### 6.3.5 Concrétisation

La dernière étape consiste à remplacer les booléens $@\gamma_i$ par les valeurs qu'ils représentent. Dans le cas du composant temporisé *Reservoir*, Tarc génère l'automate temporisé d'UPPAAL donné à la la Figure 6.8 et on retrouve celui la Figure 4.20 page 91.

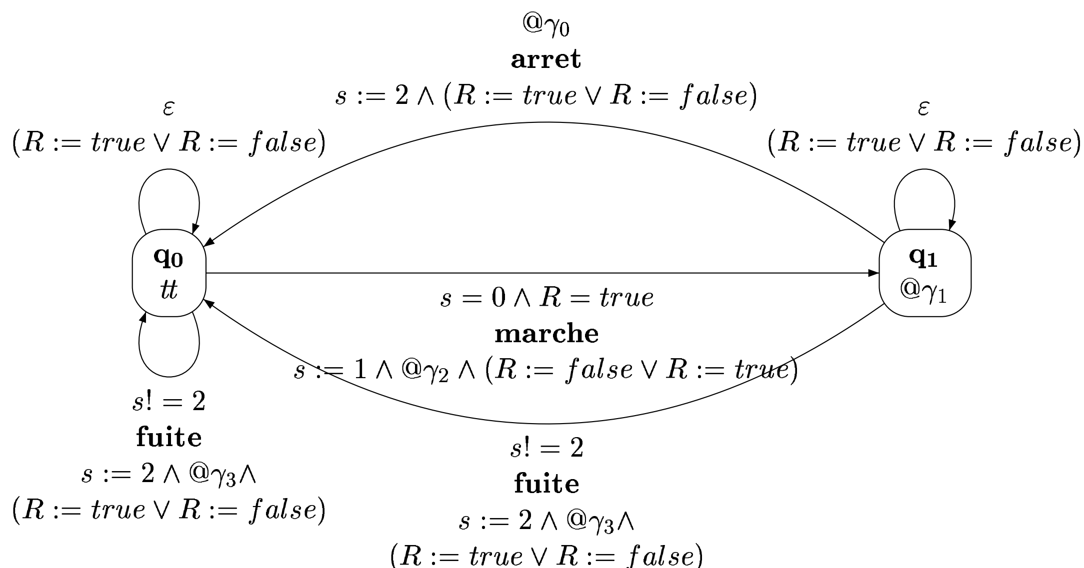



FIG. 6.7 – Automate intermédiaire de *Reservoir*

La dernière difficulté apparaît pour les horloges de flux non reliées à des horloges d'état. A chaque transition discrète, même $\varepsilon$, il faut mettre à jour ces horloges de manière indéterministe dans $\mathbb{R}$. Pour le nœud temporisé *Pompe*, si on ne représente que le contrôleur :

Ce nœud temporisé contient des horloges de flux qui peuvent être mises à jour à n'importe quel instant. Alors l'automate temporisé obtenu est représenté à la Figure 6.9.

Les formats de sortie de Tarc sont des fichiers à la syntaxe d'UPPAAL (en xml) et d'HYTECH (texte). On peut ensuite vérifier des propriétés de logiques temporelles temporisées sur les modèles Timed AltaRica. Une étude d'un système de contrôle de gouverne d'avion est proposée dans [PCR03] et nous la présentons au chapitre suivant ce qui nous permettra d'illustrer l'utilisation du prototype.

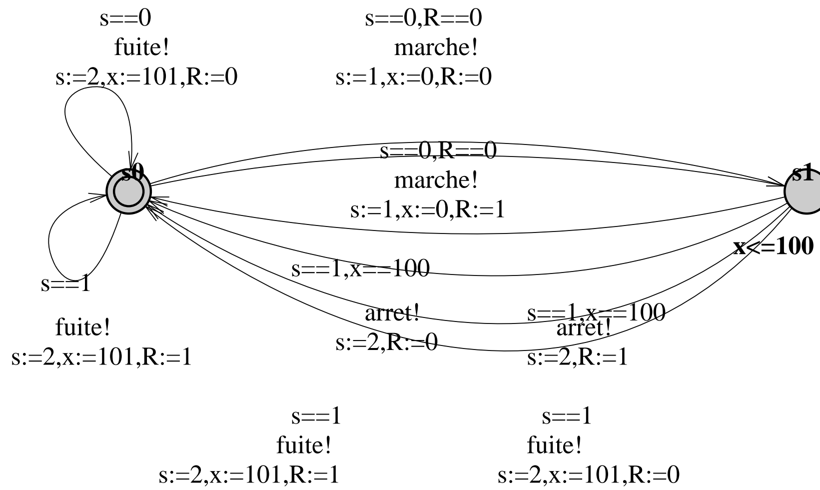



FIG. 6.8 – Traduction du nœud temporisé *Reservoir*

```
node Pompe_controleur
 state res : [0,2]; z,t:clock;
 flow R1Y, R2Y : clock;
 event marche,vide, ok, chgt
 trans
    res=0 & R1Y >= 90 |- chgt -> res:=1;
    true |- marche ->;
    res <2 & (R1Y!=t | R1Y>=100) & (R2Y!= t | R2Y>=100) |- vide -> res:=2;
    res=0 & R1Y < 90 & t=R1Y & z=10 |- ok -> z:=0;
tinvariant (res=0) => (R1Y <=100 & z<=10)
 init t:=0,res:=0,z:=0
edon
```

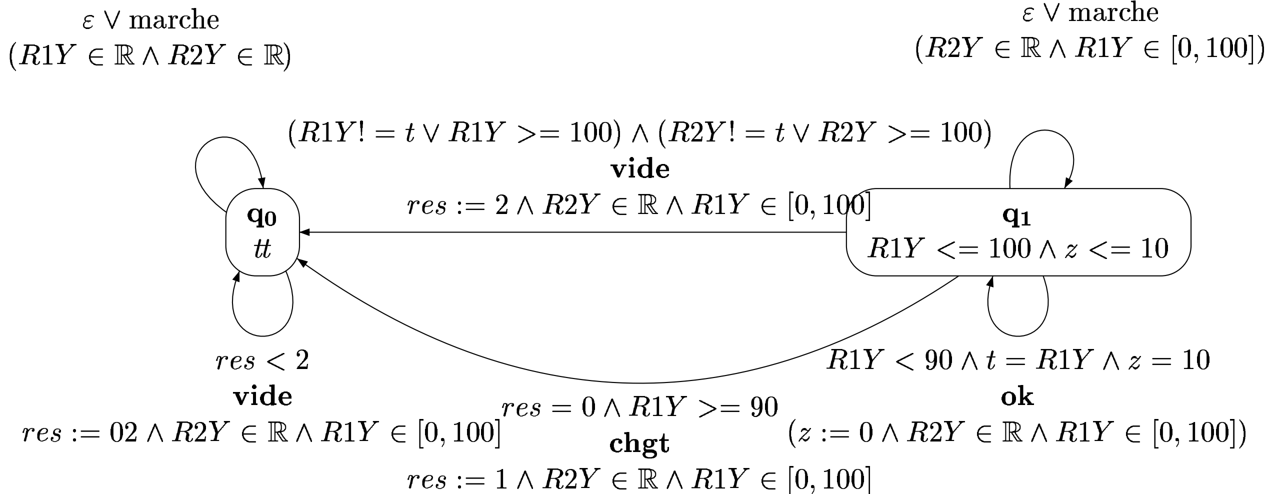



FIG. 6.9 – Automate temporisé de *Pompe_controleur*

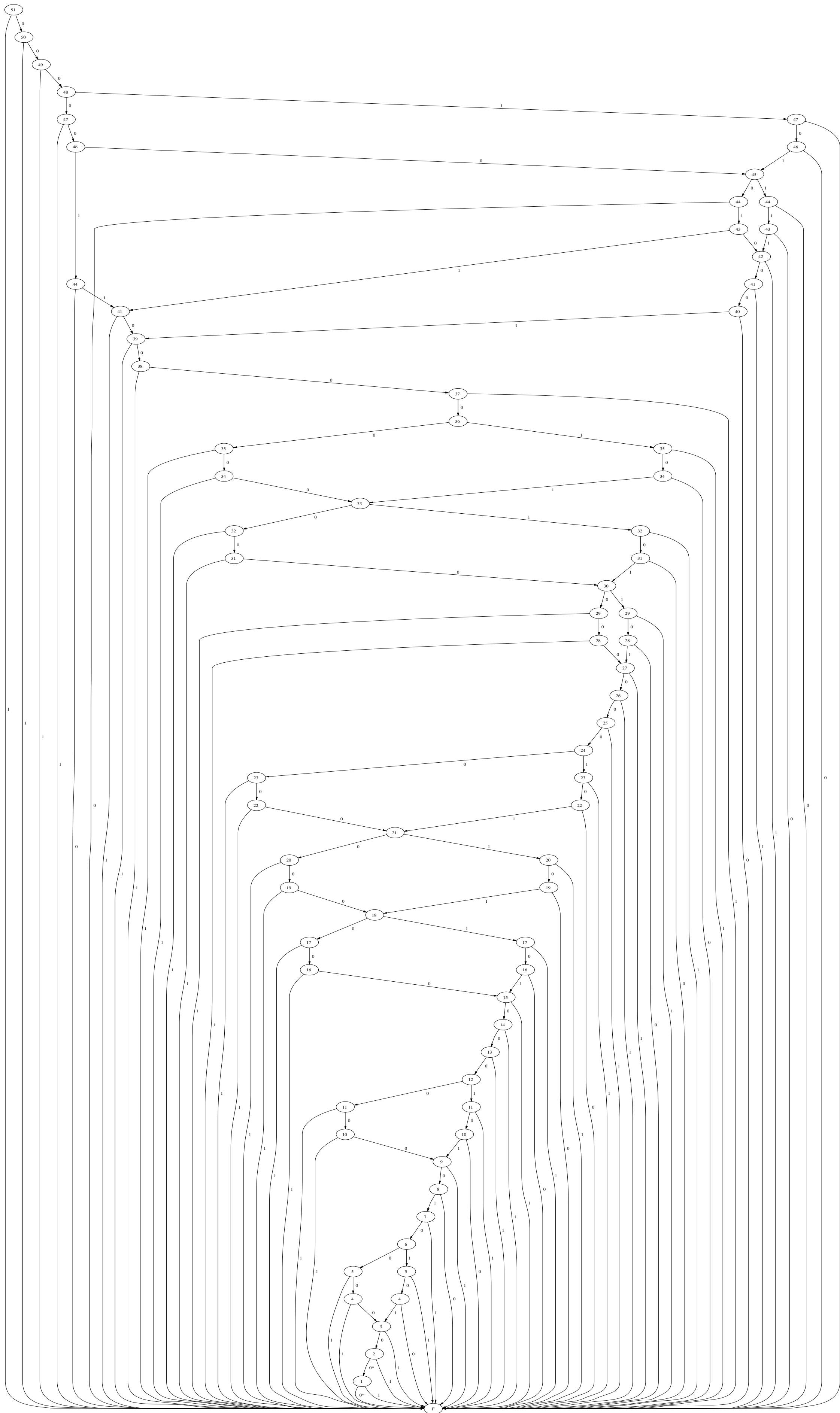


FIG. 6.10 – BDD représentant la relation de transition du vecteur < $\varepsilon$,R.joueurA,L1.$\varepsilon$,L2.$\varepsilon$,L3.$\varepsilon$, L4.joue>

# Chapitre 7

# Une étude de cas

Dans ce chapitre nous étudions un système d'avionique extrait de [BE02, BBE01]. Nous modélisons entièrement ce système en Timed AltaRica et nous le traduisons en utilisant le prototype Tarc en automate temporisé. On peut alors vérifier certaines propriétés de sûreté. On propose deux modélisations : la première étude a été menée dans [PCR03] et la deuxième, basée sur l'urgence, est réalisée dans [Pag03b].

## 7.1 Spécification du système

L'objectif est de modéliser un système de pilotage d'une gouverne à bord d'un aéronef représenté dans la Figure 7.1. Le rôle de ce système est de diriger la gouverne en fonction des instructions de pilotage émise pas l'équipage. Le système produit périodiquement une commande calculée par un calculateur sur un bus relié à la gouverne.

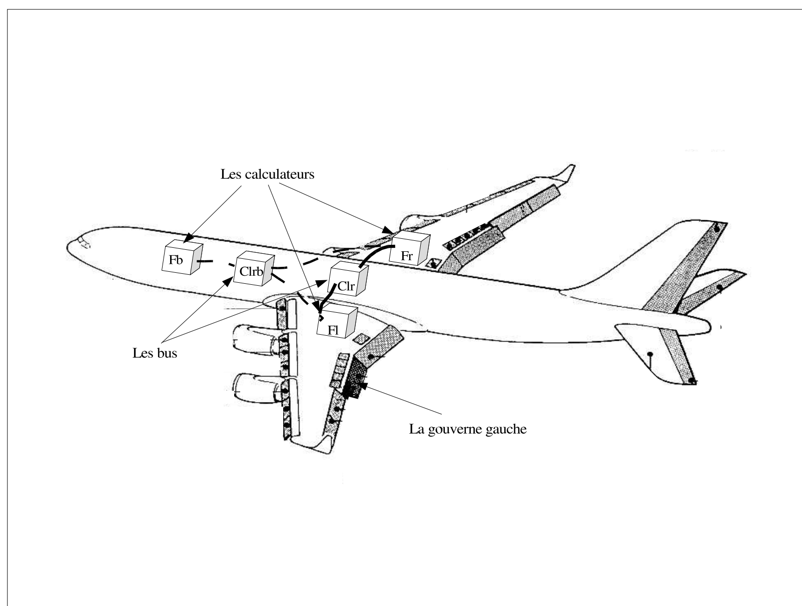



Fig. 7.1 – Schéma du système

Le système est composé de :

1. Trois calculateurs $F_L$, $F_R$ et $F_B$ se chargent de contrôler la gouverne ;
2. Deux bus principaux réalisent les communications : $C_{LR}$ connecte $F_R$ et $F_L$ (ligne continue dans la Figure 7.1) ; $C_{LRB}$ relie $F_R$ à $F_B$ et $F_L$ à $F_B$ (ligne pointillée dans la Figure 7.1). Un autre bus assure la liaison entre les calculateurs, $F_L$, $F_R$ et $F_B$, et la gouverne mais nous ne le représentons pas et nous supposons que la communication est instantanée sur ce bus.

Les spécifications informelles sont les suivantes :

1. La gouverne est contrôlée par l'événement *Cmd*. *Cmd* peut être émis par l'un des calculateurs et est envoyé par le bus instantané.
2. Initialement, le calculateur $F_R$ contrôle la gouverne tandis que $F_L$ et $F_B$ sont au repos, i.e. ils ne peuvent émettre l'événement *Cmd*.
3. Si $F_R$ tombe en panne, alors $F_L$ prend le relais et contrôle la gouverne. Si $F_L$ tombe également en panne, c'est finalement $F_B$ qui émet la commande. Dans le cas où $F_L$ est déjà défaillant lorsque $F_R$ tombe en panne, le calculateur $F_B$ prend le contrôle. On suppose que $F_B$ ne tombe jamais en panne.
4. Chaque calculateur, lorsqu'il contrôle la gouverne, émet périodiquement la commande, concrètement un événement *Cmd* est émis toutes les 20 ms.
5. Toutes les 20 ms $C_{LR}$ met à jour la valeur d'une variable d'état qui correspond à l'état de $F_R$ (0 en cas de marche et 1 en cas panne).
6. Le bus $C_{LRB}$ agit de manière similaire et met à jour toutes les 40 ms la valeur d'une variable d'état qui correspond au nombre de composants $F_R$ et $F_L$ qui sont tombés en panne (soit 0, 1 ou 2).
7. Le fonctionnement est le suivant : $F_L$ lit la dernière valeur représentant l'état de $F_R$ de $C_{LR}$ toutes les 20 ms ; quand cette valeur est à 1, il prend le relais. Il y un délai entre 10 et 20 ms avant que le composant ne devienne actif. $F_B$ de son côté récupère toutes les 20 ms les valeurs mises à jour dans $C_{LRB}$ et quand la somme vaut 2 il prend le contrôle.

Le système doit vérifier les propriétés suivantes :
$P_1$ : à tout instant, au plus un calculateur peut envoyer un événement *Cmd* ;
$P_2$ : le délai maximal entre deux événements *Cmd* doit être plus petit que 160 ms.
Dans la prochaine section, nous donnons une modélisation du système en Timed AltaRica.

## 7.2 Modélisation en Timed AltaRica

**Spécifications des bus** Les bus sont des composants Timed AltaRica : on utilise une variable d'état discrète *failure* qui se réinitialise toutes les 20 ms pour $C_{LR}$ (resp. 40 ms pour $C_{RLB}$) en la valeur $Failure_R$ représentant l'état du composant $F_R$ (resp. la somme des états de $F_R$ et $F_L$ pour $C_{LRB}$).

```
node CLR
  flow Failure_R , Failure_LR : [0,1];
  state failure : [0,2]; h : clock
  event refresh
  trans h=20 |- refresh -> h:=0,
      failure:= Failure_R ;
  init failure:=0,  h:=0;
  assert Failure_LR=failure;
  tinvariant h<=20;
 edon
```

```
node CLRB
  flow Failure_L,Failure_R: [0,1];Failure_RLB: [0,2]
  state failure : [0,2]; h : clock
  event refresh
  trans h=40 |- refresh -> h:=0, failure:=
          Failure_L + Failure_R
  init failure:=0, h:=0
  assert Failure_RLB=failure
  tinvariant h<=40;
 edon
```

**Spécifications des calculateurs** Chaque calculateur est un composant Timed AltaRica dont l'état est représenté par une variable *failure* qui est exportée par une variable de flux *Failure*. Tant que le composant vérifie $Failure_R = 0$, le calculateur $F_R$ envoie un événement $Cmd$ toutes les 20 ms.

Le calculateur $F_L$ lit, par l'événement *reset*, la variable de flux transmise par le bus $C_{LR}$. Lorsque $F_R$ tombe en panne et $F_L$ fonctionne correctement, le calculateur $F_L$ change de *mode* avec l'action *start_compute*. Lorsque le *mode* vaut 1, $F_L$ calcule la commande puis passe en $mode = 2$ pour prendre le contrôle en envoyant l'événement $Cmd$ toutes les 20 ms. A tout moment, $F_L$ peut tomber en panne avec l'action $breakdown_L$.

```
node FR
  flow Failure_R : [0,1]
  state  loc : [0,1];
     x : clock
  event breakdown_R, cmd
  trans
     x=20 & loc=0 |- cmd -> x:=0;
     x<=20 & loc=0 |- breakdown_R ->
                loc := 1
  assert Failure_R=loc
  tinvariant
       (loc=0) => (x <=20)
  init loc:=0, x:=0
 edon
```

```
node FL
  flow Failure_L , Failure_LR : [0,1];
  state failure: [0,1]; mode: [0,2] ;h : clock
  event reset, cmd, start_compute, breakdown_L
   trans
      mode=0 & Failure_LR=0 & h=20 & failure=0 |-
               reset -> h:=0 ;
      mode=0 & Failure_LR=1 & h=20 & failure=0 |-
               start_compute -> h:=0, mode:= 1;
      mode=1 & h<=20 & h>=10 & failure=0 |-
               cmd -> h := 0, mode:=2 ;
      mode=2 & h=20  &  failure=0 |- cmd -> h := 0;
      true |- breakdown_L -> failure := 1;
  init mode:=0,  failure:=0,  h:=0
  assert Failure_L=failure
  tinvariant (failure=0) => (h<=20);
edon
```

Le calculateur $F_B$ est similaire au calculateur $F_L$ : comme $F_B$ ne tombe jamais en panne, il n'y a pas d'événement *breakdown* ni de variable $failure$ ; $F_B$ lit toutes les 20 ms la valeur transmise par le bus $C_{LRB}$ et change de mode lorsque les deux autres calculateurs sont défaillants.

La spécification de la gouverne est un nœud temporisé dont les sous composants sont les calculateurs et les bus, le composant temporisé local est la gouverne elle-même qui peut se synchroniser à tout moment avec un événement $Cmd$. Avec cette spécification, vérifier la propriété $P_2$ revient à s'assurer que l'horloge locale $z$ est toujours inférieure à 160.

```
node FB
  flow Failure_RLB : [0,2];
  state mode : [0,2] ;
      h : clock
      event reset, cmd, start_compute
   trans
       mode=0 & Failure_RLB<2 & h=20 |-
                reset -> h:=0 ;
       mode=0 & Failure_RLB=2 & h=20 |-
                start_compute -> h:=0, mode:= 1;
       mode=1 & h<=20 & h>=10 |- cmd
                -> h := 0, mode:=2 ;
       mode=2 & h=20  |- cmd -> h := 0;
  init mode:=0,  h:=0
  tinvariant
    (mode = 2 | mode =1 | mode= 0) => (h<=20)
  edon
```

```
node STEERING
 state loc : [0,1]; z : clock
 event cmd
 trans loc=0 |- cmd -> z:=0
 init loc:=0,z:=0
  sub
    fr : FR;
    fl : FL;
    cr : CLR;
    cs : CLRB;
    fb : FB;
 sync
  <cmd,fr.cmd?,fl.cmd?,fb.cmd?> =2;
 assert
    fr.Failure_R=cr.Failure_R;
    fl.Failure_LR=cr.Failure_LR;
    fr.Failure_R=cs.Failure_R;
    fl.Failure_L=cs.Failure_L;
    fb.Failure_RLB=cs.Failure_RLB
 edon
```

Un vecteur de broadcast $< cmd, fr.cmd?, fl.cmd?, fb.cmd? >$ avec la contrainte $= 2$ assure qu'au plus un événement $Cmd$ est réalisé par les calculateurs. Enfin, les coordinations de flux

spécifiées dans l'invariant relie les différentes valeurs des variables.

## 7.3 Vérification sur le système

**Traduction en Uppaal.** Dans cette partie, nous appliquons Tarc sur le système afin de le transformer en Uppaal afin de vérifier les propriétés requises.

**Composants.** On peut d'abord transformer chaque composant.

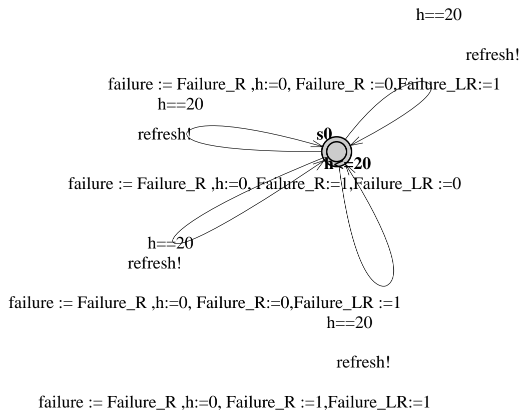



Fig. 7.2 – Automate temporisé $C_{LR}$ en Uppaal

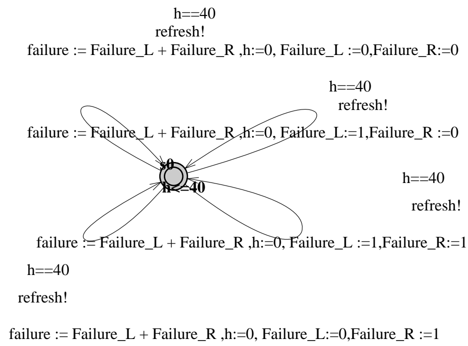



Fig. 7.3 – Automate temporisé $C_{LRB}$ en Uppaal

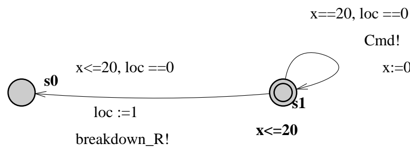



Fig. 7.4 – Automate temporisé $F_R$ en Uppaal

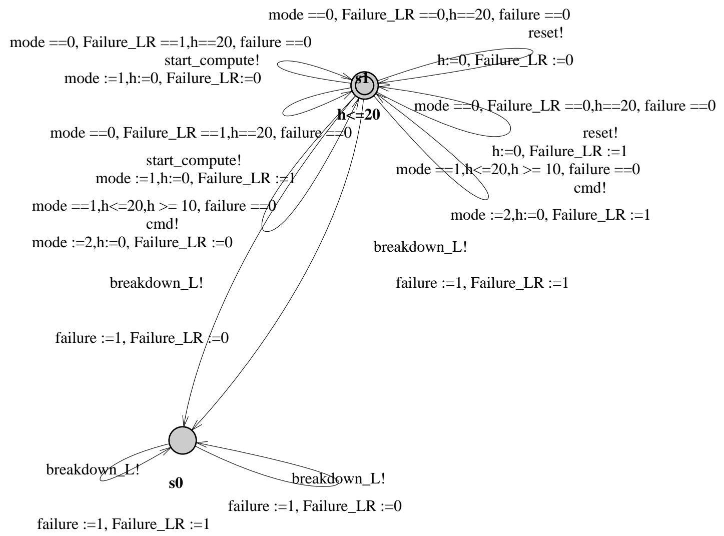



Fig. 7.5 – Automate temporisé $F_L$ en Uppaal

**Mise à plat.** Le renommage des actions pour la mise à plat n'est pas encore complètement au point.

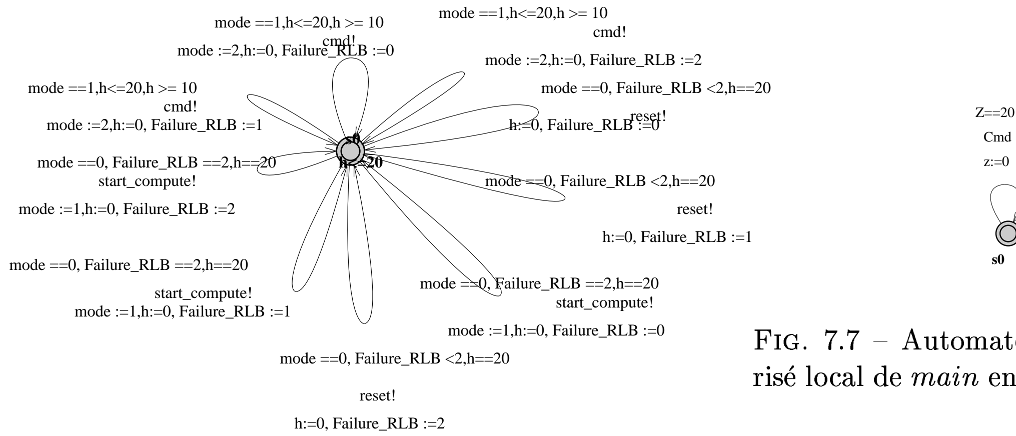



FIG. 7.6 – Automate temporisé $F_B$ en UPPAAL

FIG. 7.7 – Automate temporisé local de *main* en UPPAAL

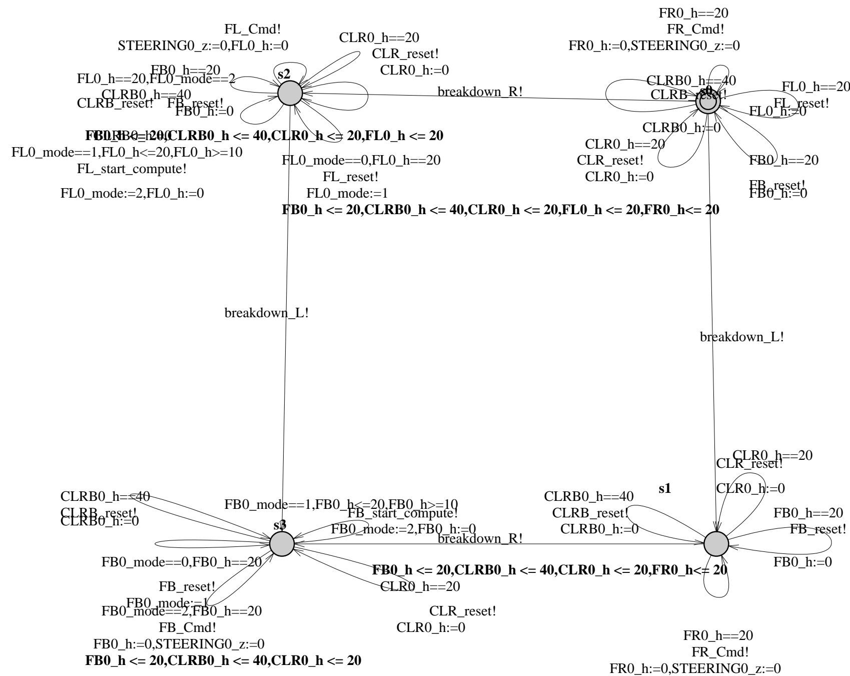



FIG. 7.8 – Automate temporisé du système mis à plat

**Vérification avec** UPPAAL. Nous pouvons maintenant vérifier les propriétés de sûreté exigées sur le système et définies section 7.1 page 128. Pour cela, nous vérifions les propriétés suivantes dans UPPAAL ;

- la propriété de sûreté `A[](not deadlock)` est vraie ce qui signifie que l'automate temporisé est sans deadlock ;
- on peut encoder $P_1$ en UPPAAL avec la formule de logique temporelle `A[](FR.s1 imply (FL.mode==0 and FB.mode==0) and ((FL.mode==2 and FL.s 1) imply FB.mode==0))`. Elle est également satisfaite par le système ;
- la dernière propriété consiste à vérifier que l'horloge locale de *main* est inférieure à 160 ms ce qui s'encode en logique temporelle de UPPAAL par `A[](steering.z <= 160)` et là encore elle est vérifiée.

De plus, on peut vérifier avec UPPAAL que la borne supérieure de 160 ms est en fait atteinte et on produit une trace d'un tel comportement.

## 7.4 Deuxième modélisation

On peut modéliser le système avec de l'urgence : plutôt que de rafraîchir les valeurs toutes les 20 ms on peut mettre de l'urgence sur les événements. Ainsi $F_R$ et le nœud temporisé *main* restent inchangés, en revanche les composants temporisés qui agissent en fonction de modifications de variables de flux sont différents. Par exemple, les calculateurs $F_L$ et $F_B$ deviennent :

```
node FL
  flow Failure_RL, Failure_L : [0,1]
  event breakdown_L, action_L, Cmd
  priority action_L > time
  state loc,failure : [0,1]; y:clock
  trans Failure_RL=1 & failure=0 & loc=0
       |- action_L -> loc:=1, y:=20;
    loc=0 |- breakdown_L -> failure:=1;
    loc=1 |- breakdown_L -> failure:=1,
       loc:=0;
    loc=1 & y=20 |- Cmd ->y:=0
  init loc:=0, failure:=0
  assert Failure_L=failure;
  tinvariant (loc=1 & failure=0) => y<= 20
 edon
```

```
node FB
 flow Panne_RLB : [0,2]
 state  etat:[0,2], z:clock
 event  action_B, Cmd
 priority action_B> time
 trans  Panne_RLB=2 & etat=0
  |- action_B -> etat:=1, z:=0;
  z=20 & etat=1 |- Cmd -> z:=0;
 init   etat:=0, z:=0;
 assert etat=1 => z<=20;
```

On utilise alors l'urgence de UPPAAL et on vérifie encore les propriétés $P_i$ sur le système.

Dans ce dernier chapitre, nous avons illustré l'utilisation possible de Timed AltaRica et du prototype Tarc. Ce prototype est encore en cours d'implantation puisque tous les opérateurs ne sont pas encore introduits et que les étiquettes (au sens large, i.e. nom, contraintes et remises à jour) dans la mise à plat ne sont pas encore intégralement récupérées.

# Chapitre 8

# Conclusion et perspectives

L'objectif de ce travail était de proposer un langage hiérarchique de modélisation hybride issu du formalisme AltaRica ainsi que des moyens de vérifier de tels systèmes.

**Synthèse.** Pour cela, nous avons procédé en deux étapes :

**Modélisation.** Nous avons étendu les modèles AltaRica de façon à obtenir des modèles aussi expressifs que les modèles temporisés et hybrides. Les langages Timed AltaRica resp. Hybrid AltaRica obtenus ont été formellement étudiés :
- la bisimulation entre deux modèles AltaRica temps réel est formellement définie,
- une réécriture syntaxique d'un modèle général en un modèle sans sous composant est proposée ce qui ouvre également la voie à une implantation de la hiérarchie.

Nous nous sommes également basés sur des travaux connexes afin d'ajouter des fonctionnalités temps réel aux langages. Pour chaque opérateur, nous avons défini les encodages syntaxiques ce qui donne un moyen de les implanter. Nous avons également prouvé que la bisimulation entre deux modèles Timed AltaRica est conservée si l'on applique des opérateurs d'urgence ou de priorité temporelle.

Enfin, nous avons reliés nos modèles avec les formalismes connus des automates temporisés et automates hybrides. Ainsi, il est possible de passer automatiquement d'un formalisme à l'autre. Un étude complète a été menée pour savoir dans quel classe d'automates appartient un composant AltaRica à partir de sa syntaxe et d'en déduire les analyses décidables qui peuvent lui être appliquées.
- Un composant seul peut être localement dans une classe d'indécidabilité alors que le système complet appartient à une classe décidable.
- Le langage temporisé reconnu par un modèle Timed AltaRica peut être calculé à partir de l'automate temporisé généré.

Il est donc possible de modéliser des systèmes temps réel hiérarchiques avec les langages Timed AltaRica et Hybrid AltaRica. Nous avons illustré cette capacité en modélisant des systèmes réels.

**Vérification.** La deuxième étape de notre recherche consistait à expérimenter les possibilités offertes par les définitions théoriques des langages obtenus. Nous avons implanter partiellement un prototype qui permet de vérifier des modèles temporisés hiérarchiques. Ce prototype n'est pas un model-checker mais un traducteur de systèmes Timed AltaRica en automates temporisés UPPAAL.

**Perspectives.** L'implantation de la partie théorique est encore en cours puisque, actuellement, seules la translation AltaRica temporisé vers automate temporisé et la mise à plat d'un système sont réalisées. Un prochaine étape sera d'incorporer l'urgence et les priorités temporelles en utilisant la librairie de polyèdres POLKA [Jea02] afin de réduire les contraintes.

# Table des figures

1.1 Atelier AltaRica [PR99a] . . . . . 12

2.1 Schéma d'un système d'extraction de pétrole . . . . . 19
2.2 Représentation d'une variable entière sous forme d'automate . . . . . 22
2.3 Exemple d'un automate à variables . . . . . 22
2.4 Représentation de l'automate à contraintes fini $\mathcal{P}aquet$ . . . . . 24
2.5 $(0, 2i, \varepsilon, 0) \in T$ . . . . . 26
2.6 $(0, 2i, e, 1) \in T$ . . . . . 26
2.7 $\mathcal{A}$ sans priorité . . . . . 26
2.8 $\mathcal{A}$ avec priorité $e < e'$ . . . . . 26
2.9 Diagramme commutatif sémantique/priorité . . . . . 27
2.10 Syntaxe d'un composant AltaRica . . . . . 29
2.11 Interface des composants . . . . . 30
2.12 Syntaxe d'un nœudAltaRica . . . . . 31
2.13 Modélisation de l'arbitre . . . . . 33
2.14 Modélisation du jeu de Marienbad . . . . . 33
2.15 Modèle du jeu de Marienbad mis à plat . . . . . 35
2.16 Plan de l'atelier AltaRica . . . . . 36

3.1 Exemple d'un automate hybride . . . . . 40
3.2 Exemple d'un automate temporisé . . . . . 43
3.3 Une exécution de l'automate hybride thermostat . . . . . 50

4.1 Syntaxe d'un composant Timed AltaRica . . . . . 56
4.2 Un réservoir en Timed AltaRica . . . . . 57
4.3 $(0, y, \varepsilon, 0, y') \in T$ . . . . . 59
4.4 $(0, y, e, 1, y') \in T$ avec $y \leq 10$ . . . . . 59
4.5 Diagramme bisimulation et homomorphisme de bisimulation . . . . . 61
4.6 Représentation du STTI $[\![Reservoir]\!]$ . . . . . 63
4.7 $((0, 5), 0) \in Pre_t([\![A]\!])$ . . . . . 64
4.8 $((1, 15), 0) \notin Pre_t([\![A]\!])$ . . . . . 64
4.9 $[\![\mathcal{A}]\!]$ sans urgence . . . . . 66
4.10 $[\![\mathcal{A}]\!]$ avec urgence $e > time$ . . . . . 66
4.11 $[\![A \upharpoonright_{\mathcal{U}_g(E)}]\!]$ - transitions $\big(vrai, \epsilon, X \in \mathbb{R} \wedge Y \in \mathbb{R}\big)$ . . . . . 70
4.12 $[\![A \upharpoonright_{\mathcal{U}_g(E)}]\!]$ - transitions $\big(X \in [10, 20], e, X \in \mathbb{R} \wedge Y \in \mathbb{R}\big)$ . . . . . 71
4.13 Diagramme commutatif sémantique/urgence . . . . . 72
4.14 $[\![A]\!] \upharpoonright_{\mathcal{U}_g(E)}$ . . . . . 73

4.15 Quelques transitions issues de $(0,7,1,10) \in [\![Ex_priorite_temporelle]\!]$ sans priorité . 75
4.16 Les mêmes transitions issues de $(0,7,1,10) \in [\![Ex_priorite_temporelle]\!]$ avec priorité 75
4.17 Syntaxe d'un nœud Timed AltaRica . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
4.18 Système de gestion du carburant . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
4.19 L'Exemple 8 en Timed AltaRica . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
4.20 Automate temporisé du composant temporisé *Reservoir* . . . . . . . . . . . . . . . . 91

5.1 Syntaxe d'un composant hybride . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
5.2 Arbitre tenant compte du temps . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
5.3 Sémantique du composant hybride *Arbitre_B* . . . . . . . . . . . . . . . . . . . . . 101
5.4 Sémantique de $N_1$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103
5.5 Syntaxe d'un nœud hybride . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103
5.6 Sémantique du nœud hybride *Noeud* . . . . . . . . . . . . . . . . . . . . . . . . . . 105
5.7 Résultats d'accessibilité des modèles Hybrid AltaRica . . . . . . . . . . . . . . . . 111

6.1 Résumé des résultats connus . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
6.2 Fonctionnement de Mec V . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
6.3 Partie de l'architecture de Mec V . . . . . . . . . . . . . . . . . . . . . . . . . . 118
6.4 Fonctionnement de Tarc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 121
6.5 Architecture de Tarc . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 122
6.6 Abstraction du nœud *Reservoir* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123
6.7 Automate intermédiaire de *Reservoir* . . . . . . . . . . . . . . . . . . . . . . . . . 124
6.8 Traduction du nœud temporisé *Reservoir* . . . . . . . . . . . . . . . . . . . . . . . 125
6.9 Automate temporisé de *Pompe_controleur* . . . . . . . . . . . . . . . . . . . . . . . 125
6.10 BDD représentant la relation de transition du vecteur < $\varepsilon$,R.joueurA,L1.$\varepsilon$,L2.$\varepsilon$,L3.$\varepsilon$, L4.joue> . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126

7.1 Schéma du système . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127
7.2 Automate temporisé $C_{LR}$ en UPPAAL . . . . . . . . . . . . . . . . . . . . . . . . . 130
7.3 Automate temporisé $C_{LRB}$ en UPPAAL . . . . . . . . . . . . . . . . . . . . . . . . 130
7.4 Automate temporisé $F_R$ en UPPAAL . . . . . . . . . . . . . . . . . . . . . . . . . . 130
7.5 Automate temporisé $F_L$ en UPPAAL . . . . . . . . . . . . . . . . . . . . . . . . . . 130
7.6 Automate temporisé $F_B$ en UPPAAL . . . . . . . . . . . . . . . . . . . . . . . . . . 131
7.7 Automate temporisé local de *main* en UPPAAL . . . . . . . . . . . . . . . . . . . . . 131
7.8 Automate temporisé du système mis à plat . . . . . . . . . . . . . . . . . . . . . . 131

# Annexe A

# Compléments sur AltaRica

Nous rappelons ici la transformation syntaxique d'un nœud AltaRica en un composant AltaRica. Cette construction nous sera utile pour justifier l'implantation de la mise à plat dans le prototype Tarc.

**Définition 35 (Sémantique symbolique [Poi00])** *Si* $\mathcal{N} = \langle V_F, E, <, \mathcal{N}_0, \cdots, \mathcal{N}_n, (\tilde{V}, <_{\tilde{V}})\rangle$ *est un nœud, on note* $\mathcal{C}_\mathcal{N} = \langle V_S, V_F, E, A, M\rangle$ *le composant sans priorité construit de la manière suivante :*

1. $\forall i = 0 \cdots n$
   (a) *si* $\mathcal{N}_i$ *est un composant, alors on défini* $\mathcal{N}'_i = \mathcal{N}_i \upharpoonright <^i$ *(la priorité est intégrée selon le procédé de résolution syntaxique donné page 27) ;*
   (b) *si* $\mathcal{N}_i$ *est un nœud, alors on définit* $\mathcal{N}'_i = \mathcal{C}_{\mathcal{N}i}$ *;*
   (c) *on note* $\mathcal{N}'_i = \langle V'_{S_i} \cup C'_{S_i}, V'_{F_i} \cup C'_{F_i}, E'_i, A'_i, M'_i, \emptyset\rangle$ *;*
2. $V_S = V'_{S_0} \cup \cdots \cup V'_{S_n}$ *et* $C_S = C'_{S_0} \cup \cdots \cup C'_{S_n}$ *;*
3. $A = (\exists_{i=1..n}(V'_{F_i} \cup C'_{F_i})). \bigwedge_{i=0..n} A'_i$ *; alors par définition pour* $((s,\nu),(f,\mu)) \in \mathcal{D}^{V_S} \times \mathbb{R}^{C_S} \times \mathcal{D}^{V_F} \times \mathbb{R}^{C_F}$, $((s,\nu),(f,\mu)) \in [\![A]\!] \iff \exists\forall i \in [1..n], \exists \eta_i \in \mathcal{D}^{V'_{F_i}} \times \mathbb{R}^{C'_{F_i}}$ *tel que* $(q,(f,\mu),\eta_1,\cdots,\eta_n)) \in [\![\bigwedge_{i=0..n} A'_i]\!]$ *avec* $\forall i \in [1..n], ((q,(f,\mu),\eta_1,\cdots,\eta_n)))[\![A_i]\!] \iff (q_i,\eta_i) \in [\![A'_i]\!]$.
4. *l'ensemble des macro-transitions* $M \subseteq (\mathbb{F} \times \mathcal{B}(C_T)) \times E \times (\mathbb{E}(V_T)^{V_T} \times \mathcal{A}(C_T))$ *est défini par* $M = (M' \upharpoonright <_{\tilde{V}}) \upharpoonright <^0$*, où* $<^0$ *est la relation de priorité définie dans la Définition 7, où* $M'$ *est donné par* $((g,\gamma),e,(a,R)) \in M'$ *si, et seulement si :*
   - $\forall i = 0 \cdots n, \exists((g_i,\gamma_i),e_i,(a_i,R_i)) \in M'_i, g = (\exists_{i=1..n}V_{F_i}).g_0 \wedge \cdots \wedge g_n$*, et* $\gamma = (\exists_{i=1..n}C_{F_i}) .\gamma_0 \wedge \cdots \wedge \gamma_n$*,*
   - $\forall x \in V_S, x \in V'_{S_i} \implies a(x) = a_i(x)$ *et* $\forall c \in C_S, c \in C'_{S_i} \implies R(c) = R_i(c)$*,*
   - $e = (e_0, e_1, \cdots, e_n) \in \tilde{V}$*.*

# Annexe B

# Exemple d'une réécriture

On donne le nœud temporisé *Pompe* de l'Exemple 24 page 80 réécrit en un composant temporisé équivalent et obtenu par l'altatool *a-semantics* [Poi03].

```
node Pompe
 flow
  R1.R : bool;
  R1.Y : clock;
  R2.R : bool;
  R2.Y : clock;
 state
  res : [0,2];
  z : clock;
  t : clock;
  R1.s : [0,2];
  R1.x : clock;
  R2.s : [0,2];
  R2.x : clock;
 event
  /* 1 */ '<vide,R1.\epsilon,R2.\epsilon>' > time,
  /* 2 */ '<\epsilon,R1.arret,R2.\epsilon>',
  /* 3 */ '<\epsilon,R1.fuite,R2.\epsilon>',
  /* 4 */ '<\epsilon,R1.\epsilon,R2.\epsilon>',
  /* 5 */ '<marche,R1.marche,R2.\epsilon>' > time,
  /* 6 */ '<ok,R1.\epsilon,R2.\epsilon>',
  /* 7 */ '<chgt,R1.\epsilon,R2.\epsilon>',
  /* 8 */ '<\epsilon,R1.\epsilon,R2.arret>',
  /* 9 */ '<marche,R1.\epsilon,R2.marche>' > time,
  /* 10 */ '<\epsilon,R1.\epsilon,R2.fuite>';
 assert
  ((~(((res) = (0)))) | (((((R1.R) = (true))) & (((R2.R) = (false))))));
  ((~(((res) > (0)))) | (((((R1.R) = (false))) & (((R2.R) = (true))))));
  ((~(((res) = (0)))) | (((((R1.Y) <= (100))) & (((z) <= (10))))));
  ((~(((R1.s) = (1)))) | (((R1.x) <= (100))));
  ((R1.x) = (R1.Y));
```

```
  ((~(((R2.s) = (1)))) | (((R2.x) <= (100))));
  ((R2.x) = (R2.Y));
 trans
     ((((R1.s) = (0))) & (((R1.R) = (true))))
   |- '<marche,R1.marche,R2.\epsilon>' ->
   R1.x:=0,R1.s:=1;

     ((((R2.s) = (0))) & (((R2.R) = (true))))
   |- '<marche,R1.\epsilon,R2.marche>' ->
   R2.x:=0,R2.s:=1;

     true
   |- '<\epsilon,R1.\epsilon,R2.\epsilon>' ->;

     ((((((res) < (2))) & (((((R1.Y) != (t))) | (((R1.Y) >= (100))))))) & ((
(((R2.Y) != (t))) | (((R2.Y) >= (100))))))
   |- '<vide,R1.\epsilon,R2.\epsilon>' ->
   res:=2;

     ((((((((res) = (0))) & (((R1.Y) < (90))))) & (((t) = (R1.Y))))) & (((z)
 = (10))))
   |- '<ok,R1.\epsilon,R2.\epsilon>' ->
   z:=0;

     ((res) = (0))
   |- '<chgt,R1.\epsilon,R2.\epsilon>' ->
   res:=1;

     ((((R1.s) = (1))) & (((R1.x) = (100))))
   |- '<\epsilon,R1.arret,R2.\epsilon>' ->
   R1.s:=2;

     ((R1.s) != (2))
   |- '<\epsilon,R1.fuite,R2.\epsilon>' ->
   R1.x:=2,R1.s:=2;

     ((((R2.s) = (1))) & (((R2.x) = (100))))
   |- '<\epsilon,R1.\epsilon,R2.arret>' ->
   R2.s:=2;

     ((R2.s) != (2))
   |- '<\epsilon,R1.\epsilon,R2.fuite>' ->
   R2.x:=2,R2.s:=2;

 init
     R1.s := 0,
     R2.s := 0,
```

```
    t := 0,
    res := 0,
    R1.x := 0,
    R2.x := 0,
    z := 0;
edon
```

# Annexe C

# La grammaire Timed AltaRica

**La métasyntaxe BNF**
Dans cette partie, nous définissons intégralement la syntaxe concrète du langage Timed AltaRica. L'ensemble de la grammaire est exprimé au format BNF (acronyme de Backus Naur Form ou Backus Normal Form). John Backus et Peter Naur ont introduit cette notation pour décrire la syntaxe du langage de programmation ALGOL 60( [?]) et depuis, l'utilisation de cette métasyntaxe formelle s'est généralisée pour exprimer des grammaires context-free.

Les méta-symboles de la BNF sont :

| Symbole | signification |
|---|---|
| ::= | "est défini par" |
| \| | "ou" |
| < > | non-terminal |

La grammaire AltaRica est définie dans [Vin03a] et nous donnons ici les ajouts temporisés qui apparaîtront en **gras**. Toute entité suivie de $_{opt}$ désigne une construction optionnelle. Les mots clés du langage apparaîtront en `type machine à écrire`.

— *Modèle général*

< *Program* > *: :=*
< *définition* > `;`$_{opt}$< *Program* >
| $\epsilon$

< *définition* > *: :=*
< *définition-de-domaine* >
| < *définition-de-constante* >
| < *définition-de-noeud-type* >

— *Définition des domaines*

*définition-de-domaine : :=*
`domain` < *identificateur* > `=` < *spécification-de-domaine* >

*spécification-de-domaine : :=*
*domaine-pré-defini*
| *domaine-intervalle*
| *domaine-énumeré*

*domaine-pré-défini : :=*
`bool`
| `identificateur`
| ***clock***

*domaine-intervalle : :=*
`[` < *expression-constante-entière* > `,` < *expression-constante-entière* > `]`

< *expression-constante-entière* > *: :=*
< *expression* >

*domaine-énuméré : :=*
{ *liste-d-identificateurs* }

— *Définition des constantes*

*définition-de-constante : :=*
`const` < *identificateur* > `=` < *expression-constante* >

< *expression-constante* > *: :=*
< *expression* >

— *Définition des modèles de composants*

< *définition-de-noeud-type* > *: :=*
`node` < *identificateur* > < *corps-de-noeud* > `edon`

< *corps-de-noeud* > *: :=*
< *liste-de-champs* >

< *liste-de-champs* > *: :=*
< *champ-de-nœud* > $;_{opt}$ < *liste-de-champs* >

< *champ-de-nœud* > *: :=*
< *définition-de-variables* >
| < *déclaration-d-événements* >
| < *définition-de-transitions* >
| < *assertions* >

| < *définition-de-sous-nœuds* >
| < *définition-de-vecteurs-de-diffusion* >
| < *définition-d-état-initial* >
| < ***définition-des-invariants*** >

— Définition des variables

< *définition-de-variables* > : :=
`state` < *liste-de-variables-définies* >
| `flow` < *liste-de-variables-définies* >

< *liste-de-variables-définies* > : :=
< *variables-définies* > `;` < *liste-de-variables-définies* >
< *variables-définies* >

< *variables-définies* > : :=
< *liste-d-identificateurs* > `:` < *spécification-de-domaine* > < *attributs* >$_{opt}$

— Déclaration des événements

< *déclaration-d-événements* > : :=
`event` < *liste-d-événements-attr* >

< *liste-d-événements-attr* > : :=
< *liste-de-comparaisons-d-événements* > < *attributs* >$_{opt}$ `;` < *liste-d-événements-attr* >
< *liste-de-comparaisons-d-événements* >

< *liste-de-comparaisons-d-événements* > : :=
< *comparaison-d-événements* > `,` < *liste-de-comparaisons-d-événements* >
| < *comparaison-d-événements* >

< *comparaison-d-événements* > : :=
< *comparaison-d-événements* > **<** < *atome-de-comparaison-d-événements* >
| < *comparaison-d-événements* > **>** < *atome-de-comparaison-d-événements* >
| < *atome-de-comparaison-d-événements* >

< *atome-de-comparaison-d-événements* > : :=
{ < *liste-de-comparaisons-d-événements* > }
| < *identificateur* >

— Définition des transitions

< *définition-de-transitions* > : :=
`trans` < *liste-de-transitions* >

< *liste-de-transitions* > : :=
< *transition* > `;` < *liste-de-transitions* >
| < *transition* > `;`$_{opt}$

< *transition* > : :=
< *expression* > < *liste-de-successeurs* >

< *liste-de-successeurs* > : :=
< *successeur* > < *liste-de-successeurs* >
| < *successeur* >

< *successeur* > : :=
`|-` < *liste-d-identificateurs* > `->` < *liste-d-affectations* >$_{opt}$

< *liste-d-affectations* > : :=
< *affectation* > `,` < *liste-d-affectations* >
| < *affectation* >

< *affectation* > : :=
< *identificateur* > `:=` < *expression* >

— *Assertions*

< *assertions* > : :=
`assert` < *liste-d-assertions* >

< *liste-d-assertions* > : :=
< *expression* > `;` < *liste-d-assertions* >
| < *expression* >

— ***Définition des invariants temporels***[1]

< ***définition-des-invariants*** > : :=
**tinvariant** < ***liste-d-invariants*** >

< ***liste-d-invariants*** > : :=
< *expression* > `;` < ***liste-d-invariants*** >
| < *expression* >

— *Définition de sous-noeuds*

[1] La syntaxe acceptée pour les invariants est moins générale que celle ci-dessous, elle est checkée ultérieurement puisqu'elle nécessite la vérification de type.

< *définition-de-sous-nœuds* > : :=
    `sub` < *liste-de-sous-nœuds* >

< *liste-de-sous-nœuds* > : :=
    < *sous-nœuds* > **;** < *liste-de-sous-nœuds* >
    | < *sous-nœuds* >

< *sous-nœuds* > : :=
    < *liste-d-identificateurs* > **:** < *identificateur* >

— *Définition de vecteurs de diffusion*

< *définition-de-vecteurs-de-diffusion* > : :=
    `sync` < *liste-de-vecteurs-de-diffusion* >

< *liste-de-vecteurs-de-diffusion* > : :=
    < *vecteur-de-diffusion* > **;** < *liste-de-vecteurs-de-diffusion* >
    | < *vecteur-de-diffusion* >

< *vecteur-de-diffusion* > : :=
    **<** < *liste-de-diffusions* > **>** < *contrainte-de-diffusion* >$_{opt}$ < *politique-de-diffusion* >$_{opt}$

< *liste-de-diffusions* > : :=
    < *diffusion* > **,** < *liste-de-diffusions* >
    | < *diffusion* >

< *diffusion* > : :=
    < *chemin* > **?**$_{opt}$

< *contrainte-de-diffusion* > : :=
    **<** < *entier* >
    | **<=** < *entier* >
    | **>** < *entier* >
    | **>=** < *entier* >

< *politique-de-diffusion* > : :=
    `max`
    | `min`

— *Définition des états initiaux*

< *définition-d-état-initial* > : :=
    `init` < *liste-d-affectations-de-chemins* >

*< liste-d-affectations-de-chemins > : :=*
    *< affectation-de-chemin >* `;` *< liste-d-affectations-de-chemins >*
    | *< affectation-de-chemin >*`,` *< liste-d-affectations-de-chemins >*
    | *< affectation-de-chemin >*

*< affectation-de-chemin > : :=*
    *< chemin >* `:=` *< expression >*

— *Directives externes*

*< directives-externes > : :=*
    `extern` ...

— *Expressions*

*< expression > : :=*
    *< expr-conditionnelle >*

*< expr-conditionnelle > : :=*
    `if` *< expression >* `then` *< expression >* `else` *< expression >*
    | *< expr-case >*
    | *< expr-disjonction >*

*< expr-case > : :=*
    `case` { *< expr-liste-de-cas >* }

*< expr-liste-de-cas > : :=*
    *< expr-un-cas >* `,` *< expr-liste-de-cas >*
    | `else` *< expression >*
    | *< expr-un-cas >*

*< expr-un-cas > : :=*
    *< expression >* `:` *< expression >*

*< expr-disjonction > : :=*
    *< expr-disjonction >* `|` *< expr-conjonction >*
    | *< expr-conjonction >*

*< expr-conjonction > : :=*
    *< expr-conjonction >* `&` *< expr-comparaison-logique >*
    | *< expr-comparaison-logique >*

*< expr-comparaison-logique > : :=*

< *expr-comparaison-logique* > **=** < *expr-comparaison-arithmétique* >
| < *expr-comparaison-logique* > **!=** < *expr-comparaison-arithmétique* >
| < *expr-comparaison-logique* > **=>** < *expr-comparaison-arithmétique* >
| < *expr-comparaison-arithmétique* >

< *expr-comparaison-arithmétique* > *: :=*
< *expr-comparaison-arithmétique* > **<** < *expr-additive* >
| < *expr-comparaison-arithmétique* > **<=** < *expr-additive* >
| < *expr-comparaison-arithmétique* > **>** < *expr-additive* >
| < *expr-comparaison-arithmétique* > **>=** < *expr-additive* >

< *expr-additive* > *: :=*
< *expr-additive* > **+** < *expr-multiplicative* >
| < *expr-additive* > **-** < *expr-multiplicative* >
| < *expr-multiplicative* >

< *expr-multiplicative* > *: :=*
< *expr-multiplicative* > ***** < *expr-unaire* >
| < *expr-multiplicative* > **/** < *expr-unaire* >
| < *expr-unaire* >

< *expr-unaire* > *: :=*
**-** < *expr-unaire* >
| **~** < *expr-unaire* >
| < *expr-atomique* >

< *expr-atomique* > *: :=*
**(** < *expression* > **)**
| < *chemin* >
| < *entier* >
| **true**
| **false**

— *Attributs*

< *attributs* > *: :=*
**:** < *liste-d-identificateurs* >

— *Entiers*[2]

< *entier* > *: :=*
*[0-9][0-9]**

[2] les entiers sont reconnus par des expressions régulières

— *Identificateur*

< *identificateur* > : :=
    `[a-zA-Z_][0-9a-zA-Z_]*`
    `'[^'.][^'.]*'`

< *chemin* > : :=
    < *identificateur* > `.` < *chemin* >
    | < *identificateur* >

< *liste-d-identificateurs* > : :=
    < *identificateur* > **,** < *liste-d-identificateurs* >
    | < *identificateur* >

# Annexe D

# La grammaire AltaRica hybride

Nous présentons la grammaire concrète d'AltaRica hybride, nous reprenons la grammaire de Timed AltaRica donnée en Annexe **??** nous donnons ici les ajout des variables continues qui apparaîtront en caractères **gras**.

— *Modèle général*

*< Program > : :=*
    *< définition >* `;`$_{opt}$*< Program >*
    | $\epsilon$

*< définition > : :=*
    *< définition-de-domaine >*
    | *< définition-de-constante >*
    | *< définition-de-noeud-type >*

— *Définition des domaines*

*définition-de-domaine : :=*
    `domain` *< identificateur >* `=` *< spécification-de-domaine >*

*spécification-de-domaine : :=*
    *domaine-pré-defini*
    | *domaine-intervalle*
    | *domaine-énumeré*

*domaine-pré-défini : :=*
    `bool`
    | `identificateur`
    | `clock`
    | ***analog***

*domaine-intervalle : :=*
`[` < *expression-constante-entière* > `,` < *expression-constante-entière* > `]`

< *expression-constante-entière* > *: :=*
< *expression* >

*domaine-énuméré : :=*
{ *liste-d-identificateurs* }

— *Définition des constantes*

*définition-de-constante : :=*
`const` < *identificateur* > `=` < *expression-constante* >

< *expression-constante* > *: :=*
< *expression* >

— *Définition des modèles de composants*

< *définition-de-noeud-type* > *: :=*
`node` < *identificateur* > < *corps-de-noeud* > `edon`

< *corps-de-noeud* > *: :=*
< *liste-de-champs* >

< *liste-de-champs* > *: :=*
< *champ-de-nœud* > `;`$_{opt}$ < *liste-de-champs* >

< *champ-de-nœud* > *: :=*
< *définition-de-variables* >
| < *déclaration-d-événements* >
| < *définition-de-transitions* >
| < *assertions* >
| < *définition-de-sous-nœuds* >
| < *définition-de-vecteurs-de-diffusion* >
| < *définition-d-état-initial* >
| < *définition-des-invariants* >

— *Définition des variables*

< *définition-de-variables* > *: :=*
`state` < *liste-de-variables-définies* >
| `flow` < *liste-de-variables-définies* >

< *liste-de-variables-définies* > : :=
    < *variables-définies* > ; < *liste-de-variables-définies* >
    < *variables-définies* >

< *variables-définies* > : :=
    < *liste-d-identificateurs* > : < *spécification-de-domaine* > < *attributs* >$_{opt}$

— *Déclaration des événements*

< *déclaration-d-événements* > : :=
    `event` < *liste-d-événements-attr* >

< *liste-d-événements-attr* > : :=
    < *liste-de-comparaisons-d-événements* > < *attributs* >$_{opt}$ ; < *liste-d-événements-attr* >
    < *liste-de-comparaisons-d-événements* >

< *liste-de-comparaisons-d-événements* > : :=
    < *comparaison-d-événements* > , < *liste-de-comparaisons-d-événements* >
    | < *comparaison-d-événements* >

< *comparaison-d-événements* > : :=
    < *comparaison-d-événements* > `<` < *atome-de-comparaison-d-événements* >
    | < *comparaison-d-événements* > `>` < *atome-de-comparaison-d-événements* >
    | < *atome-de-comparaison-d-événements* >

< *atome-de-comparaison-d-événements* > : :=
    { < *liste-de-comparaisons-d-événements* > }
    | < *identificateur* >

— *Définition des transitions*

< *définition-de-transitions* > : :=
    `trans` < *liste-de-transitions* >

< *liste-de-transitions* > : :=
    < *transition* > ; < *liste-de-transitions* >
    | < *transition* > ;$_{opt}$

< *transition* > : :=
    < *expression* > < *liste-de-successeurs* >

< *liste-de-successeurs* > : :=
    < *successeur* > < *liste-de-successeurs* >
    | < *successeur* >

< *successeur* > *: :=*
    `|-` < *liste-d-identificateurs* > `->` < *liste-d-affectations* >$_{opt}$

< *liste-d-affectations* > *: :=*
    < *affectation* > `,` < *liste-d-affectations* >
    | < *affectation* >

< *affectation* > *: :=*
    < *identificateur* > `:=` < *expression* >

— *Assertions*

< *assertions* > *: :=*
    `assert` < *liste-d-assertions* >

< *liste-d-assertions* > *: :=*
    < *expression* > `;` < *liste-d-assertions* >
    | < *expression* >

— *Définition des invariants temporels et* ***des conditions de flot***

< *définition-des-invariants* > *: :=*
    `tinvariant` < *liste-d-invariants* >

< *liste-d-invariants* > *: :=*
    < *expression* > `;` < *liste-d-invariants* >
    | < *expression* >
    | < ***expression-derivee*** > `;` < *liste-d-invariants* >
    | < ***expression-derivee*** >

— *Définition de sous-noeuds*

< *définition-de-sous-nœuds* > *: :=*
    `sub` < *liste-de-sous-nœuds* >

< *liste-de-sous-nœuds* > *: :=*
    < *sous-nœuds* > `;` < *liste-de-sous-nœuds* >
    | < *sous-nœuds* >

< *sous-nœuds* > *: :=*
    < *liste-d-identificateurs* > `:` < *identificateur* >

*— Définition de vecteurs de diffusion*

< *définition-de-vecteurs-de-diffusion* > : :=
    `sync` < *liste-de-vecteurs-de-diffusion* >

< *liste-de-vecteurs-de-diffusion* > : :=
    < *vecteur-de-diffusion* > `;` < *liste-de-vecteurs-de-diffusion* >
    | < *vecteur-de-diffusion* >

< *vecteur-de-diffusion* > : :=
    `<` < *liste-de-diffusions* > `>` < *contrainte-de-diffusion* >$_{opt}$ < *politique-de-diffusion* >$_{opt}$

< *liste-de-diffusions* > : :=
    < *diffusion* > `,` < *liste-de-diffusions* >
    | < *diffusion* >

< *diffusion* > : :=
    < *chemin* > `?`$_{opt}$

< *contrainte-de-diffusion* > : :=
    `<` < *entier* >
    | `<=` < *entier* >
    | `>` < *entier* >
    | `>=` < *entier* >

< *politique-de-diffusion* > : :=
    `max`
    | `min`

*— Définition des états initiaux*

< *définition-d-état-initial* > : :=
    `init` < *liste-d-affectations-de-chemins* >

< *liste-d-affectations-de-chemins* > : :=
    < *affectation-de-chemin* > `;` < *liste-d-affectations-de-chemins* >
    | < *affectation-de-chemin* >`,` < *liste-d-affectations-de-chemins* >
    | < *affectation-de-chemin* >

< *affectation-de-chemin* > : :=
    < *chemin* > `:=` < *expression* >

*— Directives externes*

< *directives-externes* > : :=
extern . . .

— *Expressions*

< *expression* > : :=
< *expr-conditionnelle* >

< *expr-conditionnelle* > : :=
if < *expression* > then < *expression* > else < *expression* >
| < *expr-case* >
| < *expr-disjonction* >

< *expr-case* > : :=
case { < *expr-liste-de-cas* > }

< *expr-liste-de-cas* > : :=
< *expr-un-cas* > , < *expr-liste-de-cas* >
| else < *expression* >
| < *expr-un-cas* >

< *expr-un-cas* > : :=
< *expression* > : < *expression* >

< *expr-disjonction* > : :=
< *expr-disjonction* > | < *expr-conjonction* >
| < *expr-conjonction* >

< *expr-conjonction* > : :=
< *expr-conjonction* > & < *expr-comparaison-logique* >
| < *expr-comparaison-logique* >

< *expr-comparaison-logique* > : :=
< *expr-comparaison-logique* > = < *expr-comparaison-arithmétique* >
| < *expr-comparaison-logique* > != < *expr-comparaison-arithmétique* >
| < *expr-comparaison-logique* > => < *expr-comparaison-arithmétique* >
| < *expr-comparaison-arithmétique* >

< *expr-comparaison-arithmétique* > : :=
< *expr-comparaison-arithmétique* > < < *expr-additive* >
| < *expr-comparaison-arithmétique* > <= < *expr-additive* >
| < *expr-comparaison-arithmétique* > > < *expr-additive* >
| < *expr-comparaison-arithmétique* > >= < *expr-additive* >

< *expr-additive* > : :=

< *expr-additive* > `+` < *expr-multiplicative* >
| < *expr-additive* > `-` < *expr-multiplicative* >
| < *expr-multiplicative* >

< *expr-multiplicative* > *: :=*
< *expr-multiplicative* > `*` < *expr-unaire* >
| < *expr-multiplicative* > `/` < *expr-unaire* >
| < *expr-unaire* >

< *expr-unaire* > *: :=*
`-` < *expr-unaire* >
| `~` < *expr-unaire* >
| < *expr-atomique* >

< *expr-atomique* > *: :=*
`(` < *expression* > `)`
| < *chemin* >
| < *entier* >
| `true`
| `false`

— ***Expressions des dérivées***

< ***expression-derivee*** > *: :=*
< ***expression*** > `=>` < ***info-derivee*** > `;` < ***expression-derivee*** >
| < ***expression*** > `=>` < ***info-derivee*** >

< ***info-derivee*** > *: :=*
< ***entier*** > `<=` < ***expression-lineaire*** > `&` < ***expression-lineaire*** > `<=` < ***entier*** >
| < ***expression-lineaire*** > `<=` < ***entier*** >
| < ***expression-lineaire*** > `=` < ***entier*** >

< ***expression-lineaire*** > *: :=*
< ***terme-lineaire*** >
| < ***terme-lineaire*** > `+` < ***expression-lineaire*** >
| < ***expression-lineaire*** > `-` < ***terme-lineaire*** >

< ***terme-lineaire*** > *: :=*
< ***entier*** >
| < ***entier*** > {`d`< ***identificateur*** >}
| {`d`< ***identificateur*** > }

— *Attributs*

< *attributs* > *: :=*

: < *liste-d-identificateurs* >

— *Entiers*

< *entier* > : :=
    `[0-9][0-9]*`

— *Identificateur*

< *identificateur* > : :=
    `[a-zA-Z_][0-9a-zA-Z_]*`
    `'[^'.][^'.]*'`

< *chemin* > : :=
    < *identificateur* > **.** < *chemin* >
    | < *identificateur* >

< *liste-d-identificateurs* > : :=
    < *identificateur* > **,** < *liste-d-identificateurs* >
    | < *identificateur* >

# Bibliographie


[ABC94] A. Arnold, D. Begay, and P. Crubille. *Construction and analysis of transition systems with MEC*. World Scientific, 1994.

[AC88] A Arnold and P. Crubille. A linear algorithm to solve fixed point equations on transition systems. In *Information Processing Letters*, volume 29, pages 57–66, 1988.

[ACD93] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1) :2–34, 1993.

[ACM02] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of the ACM*, 49(2) :172–206, 2002.

[AD90] R. Alur and D. Dill. Automata for modelling real-time systems. In *17 International Conference on Automata, Languages, and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. SpringerVerlag, 1990.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science B*, 126 :183–235, 1994.

[ADE+01] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. In *EMSOFT*, pages 14–31, 2001.

[Ade03] M. Adelaide. *Analyse paramétrique des systèmes hybrides*. PhD thesis, Ecole Centrale de Nantes, octobre 2003.

[AGH+00] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in CHARON. In *HSCC*, pages 6–19, 2000.

[AGP+99] K. Altisen, G. Gössler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *IEEE Real-Time Systems Symposium*, pages 154–163, 1999.

[AGPR00] A. Arnold, A. Griffault, G. Point, and A. Rauzy. The AltaRica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40 :109–124, 2000.

[AGS02] K. Altisen, G. Goessler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Journal of Real-Time Systems, special issue on "Control Approaches to Real-Time Computing"*, 23 :55–84, 2002.

[AH89] R. Alur and T. A. Henzinger. A really temporal logic. In *IEEE Symposium on Foundations of Computer Science*, pages 164–169, 1989.

[AHH93] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[AHLP00] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88 :971–984, 2000.

[AHM+98] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. MOCHA : Modularity in model checking. In *CAV 98 : Computer-Aided Verification*, Lecture Notes in Computer Science 1427, pages 521–525. Springer-Verlag, 1998.

[Alt01] K. Altisen. *Application de la synthèse de contrôleur à l'ordonnancement de systèmes temps-réel.* PhD thesis, Institut National Polytechnique de Grenoble, December 2001.

[AN01] A. Arnold and D. Niwiński. *Rudiments of $\mu$-Calculus.* Elsevier, Amsterdam, The Netherlands, 2001.

[And95] Andersen. Partial model checking. In *LICS : IEEE Symposium on Logic in Computer Science*, 1995.

[And98] H.R. Andersen. An Introduction to Binary Decision Diagrams. Technical Report DD2.16, Electronic, Department of Information Technology, Technical University of Denmark, 1998.

[ARA94] Groupe ARALIA. Arbres de défaillances et diagrammes binaires de décision. In *Actes du 1er congrès interdisciplinaire sur la Qualité et la Sûreté de Fonctionnement*, Université Technologique de Compiègne, pages 47–56, 1994.

[ARA96] Groupe ARALIA. Computation of prime implicants of a fault tree within Aralia. *Reliability Engineering and System Safety*, 1996. Special issue on selected papers from ESREL'95.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Masson, 1992.

[Arn94] A. Arnold. *Finite transition systems.* Prentice-Hall, Masson, 1994.

[Arn95] A. Arnold. *An experience with MEC in a real industrial project.* 1995.

[BBD+02] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL implementation secrets. In *Proc. of 7*th *International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.

[BBE01] F. Boniol, G. Bel, and J. Ermont. Modélisation et vérification de systèmes intégrés asynchrones : étude de cas et approche comparative. *9ème Conférence Internationale sur les Systèmes Temps Réel, RTS'2001*, 2001.

[BCK+02] P. Bieber, C. Castel, C. Kehren, C. Seguin, C. Bougnol, and J.P. Heckman. Analyse d'un système hydraulique avion avec AltaRica, 2002.

[BDFP00] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer-Verlag, 2000.

[BE02] F. Boniol and J. Ermont. Modelling and verifying embedded systems sensitive to resource availability. *3rd European Systems Engineering Conference, EuSEC'2002*, 2002.

[BFP00] P. Bouyer, E. Fleury, and A. Petit. Document de synthèse sur les procédures de vérification des systèmes temps réel. Technical report, LSV, ENS de Cachan, 2000. Calife, Sous projet 4.

[BG92] G. Berry and G. Gonthier. The Esterel synchronous programming language : Design, semantics, implementation. *Science of Computer Programming*, 19(2) :87–152, 1992.

[BGK+02] J. Bengtsson, W. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Automated analysis of an audio control protocol using UPPAAL. *Journal of Logic and Algebraic Programming*, 52–53 :163–181, July-August 2002.

[BGM+01] M. Bozga, S. Graf, L. Mounier, I. Ober, J.L. Roux, and D. Vincent. Timed Extensions for SDL. In *Proceedings of SDL FORUM'01 (Copenhagen, Denmark)*, volume 2078 of *LNCS*, pages 223–240. Springer, June 2001.

[BGP96] B. Bérard, P. Gastin, and A. Petit. On the power of non observable actions in timed automata. In C. Puech and R. Reischuk, editors, *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96)*, number 1046 in Lecture Notes in Computer Science, pages 257–268. Springer-Verlag, 1996.

[BGS00] S. Bornot, G. Gler, and J. Sifakis. On the construction of live timed systems. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 109–126, 2000.

[BHH+97] Ruth Breu, Ursula Hinkel, Christoph Hofmann, Cornel Klein, Barbara Paech, Bernhard Rumpe, and Veronika Thurner. Towards a formalization of the unified modeling language. In *Proceedings of ECOOP'97*, volume 1241. Springer Verlag, LNCS, 1997.

[BLL+95] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer–Verlag, October 1995.

[Bou02a] P. Bouyer. *Modèles et algorithmes pour la vérification des systèmes temporisés.* PhD thesis, ENS de Cachan, avril 2002.

[Bou02b] P. Bouyer. Timed automata may cause some troubles. Technical Report LSV-02-9, Lab. Specification and Verification, ENS de Cachan, 2002.

[BR94] S. Brlek and A. Rauzy. Synchronization of constrained transition systems. In *ed. H. Hong, Proc. of the First International Symposium on Parallel Symbolic Computation – PASCO'94*, World Scientific Publishing, pages 54–62, 1994.

[BRJ98] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[Bry86] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8) :677–691, August 1986.

[BS00] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 163(1) :172–202, 2000.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling urgency in timed systems. *Lecture Notes in Computer Science*, 1536 :103–129, 1998.

[CCI88] CCITT. *CCITT Recommendation Z.100 : Specification and Description Language SDL,Blue Book*, volume X.1-X.5. ITU General Secretariat, Geneva, 1988.

[CGL94] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5) :1512–1542, September 1994.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model checking.* MIT Press, 1999.

[CL93] R. Cori and D. Lascar. *Logique mathématique. Cours et exercices. Tomes I.* Masson, 1993.

[CL00] F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855, pages 373–388. Springer, 2000.

[Com01] H. Comon. Notes de cours de dea, 2001. url : http ://www.lsv.ens-cachan.fr/ comon/dea/Sommaire.html.

[CPR02] F. Cassez, C. Pagetti, and O. Roux. A timed extension for altarica. Technical Report R 12002-13, IRCCyN/CNRS, Nantes, 2002.

[CS92] R. Cleaveland and B. Steffen. A linear–time model–checking algorithm for the alternation–free modal mu–calculus. In Kim G. Larsen and Arne Skou, editors, *Proceedings of Computer Aided Verification (CAV '91)*, volume 575, pages 48–58, Berlin, Germany, 1992. Springer.

[CS01] C. Castel and C. Seguin. Modèles formels pour l'évaluation de la sûreté de fonctionnement des architectures logicielles d'avionique modulaire integrée. *AFADL*, 2001.

[CWA+96] E. Clarke, J. Wing, R. Alur, R. Cleaveland, D. Dill, A. Emerson, S. Garland, S. German, J. Guttag, A. Hall, T. Henzinger, G. Holzmann, C. Jones, R. Kurshan, N. Leveson, K. McMillan, J. Moore, D. Peled, A. Pnueli, J. Rushby, N. Shankar, J. Sifakis, P. Sistla, B. Steffen, P. Wolper, J. Woodcock, and P. Zave. Formal methods : state of the art and future directions. *ACM Computing Surveys*, 28(4) :626–643, 1996.

[CWU98] M. Conrad, M. Weber, and O. Uller. Towards a methodology for the design of hybrid systems in automotive electronics, 1998.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal mu-calculus. *Theoretical Computer Science*, 126(1) :77–96, February 1994.

[Dan00] T. Dang. *Verification and Synthesis of Hybrid Systems*. PhD thesis, Institut National Polytechnique de Grenoble, Laboratoire Verimag, 2000.

[Der74] M. Dertouzos. Control robotics : the procedural control of physical processes. *Information Processing*, 74 :807–813, 1974.

[Der98] Voeten Van Der. System level modelling for hardware/software systems, 1998.

[DGV96] A. Deshpande, Aleks G., and P. Varaiya. SHIFT : A formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems*, pages 113–133, 1996.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407, pages 197–212. Lecture Notes in Computer Science, Springer-Verlag, 1989.

[DM99] L. C. Dipippo and L. MA. A UML package for specifying real-time objects. Technical Report TR99-274, University of Rhode Island, 1999.

[DOTY95] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III : Verification and Control*, volume 1066, pages 208–219, Rutgers University, New Brunswick, NJ, USA, 22–25 October 1995. Springer.

[DP99a] J. Delatour and M. Paludetto. De hood/pno à uml/pno : Une méthode pour les systèmes temps réels basée uml et objets à réseau de petri. In Hermès, editor, *Actes du Congrés Modélisation des Systèmes Réactifs, MSR'99*, 1999.

[DP99b] J. Delatour and M. Paludetto. Uml et les réseaux de petri : Vers une sémantique des modèles dynamiques et une méthodologie de développement des systèmes temps réel. *Revue l'objet*, 1999.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *7th IEEE Real Time Systems Symposium, RTSS'96*, pages 73–81, Washington, DC, USA, 1996. IEEE Computer Society Press.

[EH82] E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 169–180. ACM Press, 1982.

[Eri99] C. Ericson. Fault tree analysis - A History. In *The 17 th International System Safety Conference*, 1999.

[FH00] M. Frappier and H. Habrias. A comparison of the specification methods, 2000.

[FM02] S. Flake and W. Mueller. Specification of real-time properties for UML models. In Ralph H. Sprague, Jr., editor, *Proc. 35th Annual Hawaii International Conference on System Sciences (HICSS-35)*. IEEE Computer Society, 2002.

[FV94] L. Fribourg and M. Veloso-Peixoto. Automates concurrents à contraintes. *Technique et Science Informatiques*, 13(6) :837–866, 1994.

[GK00] A. Göllü and M. Kourjanski. Object-oriented design of automated highway simulations using shift programming language, 2000.

[GLP+98] A. Griffault, S. Lajeunesse, G. Point, A. Rauzy, J.-P. Signoret, and P. Thomas. The AltaRica language. In *Proceedings of the International Conference on Safety and Reliability, ESREL'98*. Balkema Publishers, June 20-24 1998.

[Gri02] A. Griffault. Stratégies gagnantes dans les jeux. Le jeu de Nim *dit de Marienbad*, 2002. url : http ://altarica.labri.fr/Examples/.

[Gri03] A. Griffault. Conception et validation d'un protocole avec le modèle AltaRica. In Jean-Marc Jézéquel, editor, *AFADL : Approches Formelles dans l'Assistance au Développement de Logiciels*, pages 293–307. IRISA, January 2003.

[GSB98] Radu Grosu, Thomas Stauner, and Manfred Broy. A modular visual model for hybrid systems. In *Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'98)*. Springer-Verlag, 1998.

[GV03] A. Griffault and A. Vincent. Vérification de modèles AltaRica. In *MAJECSTIC 2003*, 2003.

[Har87] D. Harel. Statecharts : A visual formalism for complex systems. *Science of Computer Programming*, 8(3) :231–274, June 1987.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 1996. IEEE Computer Society Press.

[HHWT97] T. A. Henzinger, P. H. Ho, and H. Wong-Toi. HYTECH : A model checker for hybrid systems. *Lecture Notes in Computer Science*, 1254 :460– ? ?, 1997.

[HKPV95] T. A. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata ? In *27th Annual Symposium on Theory of Computing (STOC)*, pages 373–382. ACM Press, 1995.

[HLM94] T. Hutinet, S. Lajeunesse, and L. Martin. Atelier fiabex, vers une étude des études sdf en phase de conception. In *Actes du Congrès $\lambda\mu$ 94, ESREL'94*, pages 694–700, 1994.

[HLMR02] M. Heymann, F. Lin, G. Meyer, and S. Resmerita. Analysis of zeno behaviors in hybrid systems. In *the Proceeding of the 41th IEEE Conference on Decision and Control*, pages 2379–2384, 2002.

[HLS99] K. Havelund, K. G. Larsen, and A. Skou. Formal verification of a power controller using the real-time model checker UPPAAL. In *5th International AMAST Workshop on Real-Time and Probabilistic Systems*, 1999.

[HM00] T. A. Henzinger and R. Majumdar. A classification of symbolic transition systems. *Lecture Notes in Computer Science*, 1770 :13–34, 2000.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *7th. Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In *In K. R. Apt, editor, Logics and Models of Concurrent Systems*, volume F-13 of NATO ASI Series, pages 477–498, New York, 1985. Springer-Verlag.

[HS96] Klaus Havelund and Natarajan Shankar. Experiments in Theorem Proving and Model Checking for Protocol Verification. In Marie-Claude Gaudel and Jim Woodcock, editors, *FME'96 : Industrial Benefit and Advances in Formal Methods*, pages 662–681. Springer-Verlag, 1996.

[Jea02] B. Jeannet. *Convex Polyhedra Library*, release 1.1.3c edition, March 2002. Documentation of the “New Polka” library available at http ://www.irisa.fr/prive/Bertrand.Jeannet/newpolka.html.

[JLSE99] K.H. Johansson, J. Lygeros, S. Sastry, and M. Egerstedt. Simulation of zeno hybrid automata. In *IEEE Conference on Decision and Control*, Phoenix, 1999.

[JM95] M. Jourdan and F. Maraninchi. Static timing analysis of real-time systems. In *Workshop on Languages, Compilers and Tools for Real-Time Systems*, pages 79–87, 1995.

[JMO93] M. Jourdan, F. Maraninchi, and A. Olivero. Verifying quantitative real-time properties of synchronous programs. In *Computer Aided Verification*, pages 347–358, 1993.

[Koy92] R. Koymans. (real) time : A philosophical perspective. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[KSF+99] S. Kowalewski, O. Stursberg, M. Fritz, H. Graf, I. Hoffmann, J. Preussig, M. Remelhe, S. Simon, and H. Treseler. A case study in tool-aided analysis of discretely controlled continuous systems : The two tanks problem. In Panos J. Antsaklis, Wolf Kohn, Michael D. Lemmon, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems*, volume 1567 of *Lecture Notes in Computer Science*. Springer, 1999.

[Lab98] A. Labroue. Conditions de vivacité dans les automates temporisés. Technical Report LSV-98-7, Lab. Specification and Verification, ENS de Cachan, 1998.

[Lap85] J. C. Laprie. Dependable computing and fault tolerance : concepts and terminology. In *Fault Tolerant Computing Symposium 15*, pages 2–11, Ann Arbor, June 1985. IEEE Computer Society.

[Lar99] F. Laroussinie. A user guide to Cmc v1.3, 1999.

[LBB+01] K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible : Efficient cost-optimal reachability for priced timed automata. *Lecture Notes in Computer Science*, 2102 :493+, 2001.

[LHM98] M.D. Lemmon, K.X. He, and I. Markovsky. A tutorial introduction to supervisory hybrid systems, 1998.

[LJS+01] J. Lygeros, K. H. Johansson, S. N. Simić, J. Zhang, and S. Sastry. Continuity and invariance in hybrid automata. In *Proc. 40th IEEE Conference on Decision and Control*, Orlando, FL, 2001.

[LL73] C. L. Liu and J. W Layland. Scheduling algorithms for multi-programming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20 :40–61, 1973.

[LL90] L. Lamport and N. Lynch. Distributed computing : Models and methods. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science : Volume B : Formal Models and Semantics*, pages 1157–1199. Elsevier, Amsterdam, 1990.

[LL95] F. Laroussinie and K. G. Larsen. Compositional model checking of real time systems. In *Proc. of the66th International Conference on Concurrency Theory (CONCUR'95)*, pages 27–41, Philadelphia, PA, 1995.

[LL98a] F. Laroussinie and K. G. Larsen. CMC : A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic Publishers, 1998.

[LL98b] F. Laroussinie and K.G. Larsen. CMC : A tool for compositional model-checking of real-time systems. In *IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, volume 1066, pages 439–456, Paris, France, November 1998. Kluwer Academic.

[LLW95] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic - and back. In *Proc. 20th Int. Symp. Math. Found. Comp. Sci. (MFCS'95), Prague, Czech Republic, Aug.-Sep. 1995*, volume 969, pages 27–41. Springer, 1995.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, December 1995.

[LPY01] M. Lindahl, P. Pettersson, and W. Yi. Formal Design and Analysis of a Gearbox Controller. *Springer International Journal of Software Tools for Technology Transfer (STTT)*, 3(3) :353–368, 2001.

[McM92] K. L. McMillan. The SMV system. Technical Report CMU-CS-92-131, 1992.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academics Publishers, 1993.

[MR98] F. Maraninchi and Y. Rémond. Mode-automata : About modes and states for reactive systems. *Lecture Notes in Computer Science*, 1381 :185– ? ?, 1998.

[MY96] O. Maler and S. Yovine. Hardware timing verification using kronos. In *7 th Israeli Conference on Computer Systems and Software Engineering*, Herzliya, Israel, June 1996.

[NSY93] X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30(2) :181–202, 1993.

[NY00] P. Niebert and S. Yovine. Computing optimal operation schemes for chemical plants in multi-batch mode. In *HSCC*, pages 338–351, 2000.

[Pag96] F. Pagani. Partial orders and verification of real-time systems. In B. Jonsson and J. Parrow, editors, *Proceedings of the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume LNCS 1135, pages 327–346. Springer Verlag, Uppsala, Sweden, 1996.

[Pag03a] C. Pagetti. The Timed Altarica compiler, 2003. Available at `http ://altarica.labri.fr`.

[Pag03b] C. Pagetti. Une extension temporisée d'AltaRica. In Hermès, editor, *Actes du Congrés Modélisation des Systèmes Réactifs, MSR'03*, Metz, France, 2003. AltaRica.

[PCR03] C. Pagetti, F. Cassez, and O. Roux. Tarc : A timed hierarchical modelling tool. In Hung Dang Van and Zhiming Liu, editors, *Workshop on Formal Aspects of Component Software FACS'03*, volume Technical Report 284, UNU/IIST, Pisa, Italy, september 2003.

[PD96] Fong Pong and Michel Dubois. A survey of verification techniques for cache coherence protocols. *ACM Computing Surveys (CSUR) archive*, 29 :82 – 126, 1996.

[PL00] P. Pettersson and K. G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70 :40–44, February 2000.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, Providence, Rhode Island, October 31–November 2 1977. IEEE, IEEE Computer Society Press.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13 :45–60, 1981.

[Poi00] G. Point. *AltaRica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. PhD thesis, University of Bordeaux I, Janvier 2000.

[Poi03] G. Point. The altatools, 2003. Available at `http ://altarica.labri.fr`.

[PR99a] G. Point and A. Rauzy. AltaRica - constraint automata as a description language. *European Journal on Automation*, 1999. Special issue on the *Modelling of Reactive Systems*.

[PR99b] G. Point and A. Rauzy. AltaRica - langage de modélisation par automates à contraintes. In Hermès, editor, *Actes du Congrés Modélisation des Systèmes Réactifs, MSR'99*, 1999.

[Rau95] A. Rauzy. Toupie = $\mu$-calculus + constraints. In *Proceedings of Computer Aided Design, CAV'95*, volume 939, pages 114–126. Lecture Notes in Computer Science, Springer-Verlag, 1995.

[Rau02] A. Rauzy. Mode automata and their compilation into fault trees. *Reliability Engineering and System Safety*, 78 :1–12, 2002.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall International Editions, New York, NY, 1991.

[Rus94] J. Rushby. Critical system properties : Survey and taxonomy. *Reliability Engineering and System Safety*, 43(2) :189–219, 1994.

[RW89] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of IEEE : Special Issue on Discrete Event Systems*, 77 :81–98, 1989.

[RW92] K. Rudie and W. M. Wonham. Protocol verification using discrete-event systems. In *Proc. 31st IEEE Conf. on Decision and Control*, pages 3770–3777, 1992.

[SBB+99] P. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, and A. Petit. *Vérification de logiciels : Techniques et outils du model-checking*. Vuibert, 1999.

[Sig95] J.-P. Signoret. Moca-rp v9. Technical Report EP/P/SE/MRT-ARF/JPS9634, Elf-Aquitaine, 1995.

[SY96] J. Sifakis and S. Yovine. Compositional specification of timed systems. In *13th Annual Symp. on Theoretical Aspects of Computer Science, STACS'96*, volume 1046, pages 347–359. Lecture Notes in Computer Science, Springer-Verlag, 1996. Invited paper.

[TPS98] C. Tomlin, G. Pappas, and S. Sastry. Conflict resolution for air traffic management : A study in muti-agent hybrid systems, 1998.

[TY98] S. Tripakis and S. Yovine. Verification of the fast reservation protocol with delayed transmission using the tool Kronos. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium, RTAS'98*, pages 165–, 1998.

[Vin03a] A. Vincent. *Conception et réalisation d'un vérificateur de modèles AltaRica*. PhD thesis, University of Bordeaux I, 2003.

[Vin03b] A. Vincent. The MEC V compiler, 2003. Available at `http ://altarica.labri.fr`.

[Wil94] T. Wilke. *Automaten und Logiken zur Beschreibung zeitabhängiger Systeme.* PhD thesis, Inst. f. Informatik u. Prakt. Math., CAU Kiel, 1994.

[You82] C. J. Young. *Real Time Languages : Design and Development.* 116. Ellis Horwood, 1982.

[Yov97] S. Yovine. Kronos : A verification tool for real-time systems. (kronos user's manual release 2.2), 1997.