Reconnaissance de comportements complexes par traitement en ligne de flux d'événements

Ariane Piel

Université Paris 13

Thèse

pour obtenir le grade de

Docteur de l'Université Paris 13

Discipline : Informatique

présentée et soutenue publiquement par

Ariane PIEL

le 27 octobre 2014

à l'Office National d'Études et de Recherches Aérospatiales

Reconnaissance de comportements complexes par traitement en ligne de flux d'évènements

(Online Event Flow Processing for Complex Behaviour Recognition)

Jury

Présidente :

Laure Petrucci	Professeur, *Université Paris 13*

Rapporteurs :

Serge Haddad	Professeur, *École Normale Supérieure de Cachan*
Audine Subias	Maître de conférences, HDR, *Institut National des Sciences Appliquées de Toulouse*

Examinateurs :

Philippe Bidaud	Professeur, *Université Paris 6 – Pierre et Marie Curie*
Christine Choppy	Professeur, *Université Paris 13*
Romain Kervarc	Docteur ingénieur, *Onera*

Invités :

Jean Bourrely	Docteur ingénieur, *Onera*
Patrice Carle	Docteur ingénieur, *Onera*

Directrice de thèse :	Christine Choppy
Encadrants Onera :	Romain Kervarc, avec Jean Bourrely et Patrice Carle
Laboratoires :	Office National d'Études et de Recherches Aérospatiales
	Laboratoire d'Informatique de Paris Nord

N° d'ordre :

Remerciements

Nombreux sont ceux qui ont contribué à l'aboutissement de cette thèse, et je souhaite à présent les remercier très sincèrement.

Cette thèse s'est déroulée à l'Onera au sein du Département de Conception et évaluation des Performances des Systèmes (DCPS). Je tiens à remercier Thérèse Donath, directrice du département, de m'avoir accueillie et de m'avoir permis de réaliser ces travaux dans un environnement stimulant au sein de l'Onera. Je souhaite vivement remercier l'Unité de Techniques pour la Conception et la Simulation de systèmes (TCS) dans laquelle j'ai été chaleureusement intégrée.

Je remercie Christine Choppy, Professeur à l'Université Paris 13, d'avoir dirigé ma thèse durant ces trois années avec tant d'implication et de disponibilité. Son sens de l'organisation et sa ténacité nous a permis d'entreprendre de nombreux projets. Je garde un excellent souvenir d'un workshop organisé à Singapour où nous avons pu allier les plaisirs de la recherche et ceux des voyages.

Je suis très reconnaissante envers Romain Kervarc, ingénieur-chercheur à l'Onera, d'avoir coencadré cette thèse et de m'avoir toujours soutenue tant sur les idées que sur la forme où nous nous retrouvions entre logiciens. Je remercie aussi Romain de m'avoir offert de nombreuses opportunités pour rencontrer des chercheurs à travers le monde, et de m'avoir accordé sa confiance, laissant ainsi libre cours à la poursuite de mes idées.

J'exprime toute ma gratitude à Patrice Carle, chef de l'unité TCS, pour m'avoir aidée sur tous les aspects de ce travail, avec des remarques toujours pertinentes. L'expérience et le recul sur le sujet qu'a apportés Patrice allaient de pair avec un enthousiasme débordant, ce qui a su avoir raison de mes doutes et de mon regard parfois trop critique. Sa certitude dans la qualité et l'intérêt de notre projet a été d'un grand soutien.

Je tiens à remercier tout particulièrement Jean Bourrely, ancien directeur adjoint du DCPS, pour sa clairvoyance et son sens pratique dans tous les moments de questionnement, et pour son aide incommensurable dans l'implémentation de CRL. Les nombreuses heures de travail en commun m'ont donné l'énergie et la stimulation nécessaires à l'aboutissement de cette thèse et comptent dans les instants qui resteront parmi les meilleurs de ces trois ans.

J'exprime toute ma gratitude à Serge Haddad, Professeur à l'École Normale Supérieure de Cachan, et à Audine Subias, Maître de Conférences habilité à l'Institut National des Sciences Appliquées de Toulouse, pour m'avoir honorée d'être les rapporteurs de cette thèse. Leurs remarques ont été précieuses pour l'amélioration de ce travail et ont ouvert quantité de nouvelles perspectives.

Je souhaite étendre cette gratitude à Laure Petrucci, Professeur à l'Université Paris 13, et à Philippe Bidaud, Professeur à l'Université Paris 6, pour m'avoir suivie durant ces trois années puis avoir accepté de participer au jury de cette thèse.

Mes remerciements vont également à Thibault Lang qui m'a généreusement aidée dans le développement des cas d'application traités dans cette thèse, et à Nicolas Huynh pour sa contribution à leur mise en place. J'adresse aussi mes profonds remerciements à Stéphanie Prudhomme pour l'atmosphère chaleureuse qu'elle sait entretenir au sein de l'unité TCS, et pour ses conseils avisés.

Je dois beaucoup à mes collègues et à la bonne ambiance qu'ils font régner au sein du département. J'ai eu la chance d'être particulièrement bien entourée ce qui est inestimable dans le travail de longue haleine que représente un doctorat, travail qui peut être emprunt de frustrations et de solitude. Je remercie tous ceux qui passaient de temps en temps dans mon bureau et j'adresse une pensée particulière à Mathieu et son Apple-attitude, Pierre avec son tweed et son bronzage uniques, Rata et sa bonne humeur, Damien et sa décontraction sudiste, Arthur lorsqu'il n'est pas en vacances, Evrard lorsqu'il est réveillé, Sarah avec sa force surhumaine et ses conseils esthétiques, Pawit et ses bons plats thaïs, Yohan avec son gâteau au chocolat et son fromage blanc à Saulx-les-Chartreux, Joseph avec le petit compartiment de sa machine à pain et son thé, Dalal et son rire, et tous les autres (même ceux du DTIM).

Je souhaite remercier plus particulièrement Antoine notamment pour nos tours de ring et nos échanges autour de bons thés (exempts ou non de grand-mères) qui m'ont permis de surmonter l'étape douloureuse que peut être la rédaction ; Christelle, pour son amitié et son soutien, avec nos nombreuses et longues discussions qui m'ont portée au travers des moments les plus difficiles, aussi bien professionnels que personnels ; et Loïc, qui dès son arrivée en thèse, a su m'offrir une oreille attentive et des conseils éclairés tout en entretenant ma motivation dès qu'elle était trop fragile.

Je ne pourrais oublier de remercier également mes amis Lucille, Ambre, Mathieu, Lise, Estelle, Laetitia, Julie, Anaïs, Charlotte, Yelena, Alice et les Glénanais qui m'ont encouragée jusqu'au jour de ma soutenance.

Enfin, mais non des moindres, je remercie mes frères, David et Thomas, ainsi que ma mère, Sandra et Coline pour avoir toujours cru en moi et avoir fait preuve d'un encouragement sans faille.

Table des matières

Introduction

Dans de nombreux domaines, notamment l'aérospatial, le médical, la finance ou le nucléaire, des quantités très importantes de données sont produites par des systèmes pouvant être réels ou simulés. Pour la manipulation de ces masses énormes de données, des outils d'aide à l'analyse sont nécessaires. Par exemple, l'industrie aérospatiale a recours de façon systématique à la simulation afin de pouvoir étudier l'ensemble des caractéristiques de systèmes de grande envergure ; et il est nécessaire de pouvoir exploiter les données produites.

Par ailleurs, au vu de l'automatisation croissante des tâches, les systèmes mis en cause sont de plus en plus critiques et de plus en plus complexes. Ils mettent en interaction hommes, machines et environnements, rendant ainsi les risques humains et matériels de très grande envergure. Ceci rend nécessaire l'emploi de méthodes formelles pour s'assurer de la correction des outils d'aide à l'analyse utilisés.

C'est dans ce contexte que s'inscrit la problématique de la reconnaissance formelle de comportements dans le cadre de problèmes complexes, domaine du Complex Event Processing (CEP). Il s'agit de développer des outils fiables d'aide à l'analyse de flux d'évènements permettant de reconnaître des activités pouvant être aussi bien normales qu'anormales dans des flux complexes d'évènements.

Parmi les formalisations de description et de reconnaissance de comportements, on peut citer les suivantes :

— L'Event Calculus (EC), dont les travaux récents sont menés principalement par A. Artikis, est fondé sur la programmation logique. Des séries de prédicats, qui peuvent être dérivés par apprentissage [ASPP12], définissent les comportements à reconnaître. Initialement, leur reconnaissance ne pouvait se faire en ligne, mais une solution est proposée dans [ASP12]. De plus, un raisonnement probabiliste peut être introduit dans l'EC [SPVA11].

— Les chroniques de Dousson et al. [DLM07] sont des ensembles de formules décrivant des associations d'évènements observables et sont représentées par des graphes de contraintes.

— Les chroniques de [Ber09, CCK11] développées à l'Onera sont un langage temporel bien distinct des chroniques de Dousson et permettant la description formelle de comportements puis leur reconnaissance en ligne, et ce avec historisation des évènements (c'est-à-dire avec la possibilité de remonter précisément à la source des reconnaissances). Elles sont définies par induction à partir d'opérateurs exprimant la séquence, la conjonction, la disjonction et l'absence. Un modèle de reconnaissance est proposé en réseaux de Petri colorés, où un

réseau calculant les ensembles de reconnaissances peut être construit pour chaque chronique que l'on souhaite reconnaître.

Les applications de ces travaux sont très variées et touchent notamment à la médecine (monitoring cardiaque par exemple), la gestion d'alarmes, la sécurité informatique, l'évaluation de la satisfaction de passagers dans des transports publics, la surveillance vidéo, etc.

Cette thèse consiste à formaliser et étendre un langage de description de comportements, le langage des chroniques de [CCK11], tout en développant des modèles d'implémentation du processus de reconnaissance pour ensuite traiter des applications variées.

La démarche consiste dans un premier temps à formaliser et étendre le langage des chroniques défini dans [CCK11] afin de répondre à un besoin d'expressivité plus important. On commence par définir inductivement la syntaxe du langage, puis, afin de lui donner un sens, on explicite la notion de reconnaissance de chronique par une fonction. Cette dernière est définie par induction pour chaque chronique : à un flux d'évènements, elle associe l'ensemble des reconnaissances de la chronique dans ce flux. Dans [CCK11], le formalisme de représentation des reconnaissances de chroniques est ensembliste (chaque reconnaissance est représentée par un ensemble d'évènements), mais cela ne permet pas de distinguer certaines reconnaissances car il n'est pas possible de savoir à partir de ces ensembles à quelle partie de la reconnaissance de la chronique correspond chaque évènement de l'ensemble de reconnaissance. Pour cela, on modifie ce formalisme en remplaçant la représentation ensembliste par une représentation *arborescente* des reconnaissances qui permet de conserver plus d'informations et de distinguer toutes les reconnaissances possibles.

Après cette formalisation, on peut étendre l'expressivité du langage avec de nouvelles *constructions temporelles* et introduire la manipulation d'*attributs d'évènements*. Davantage de comportements peuvent ainsi être exprimés et une plus grande variété d'applications peut être traitée. Les constructions temporelles choisies sont de deux types : d'une part les constructions contraignant temporellement une chronique par rapport à une autre, et d'autre part celles contraignant une chronique par rapport à un délai. Ces constructions découlent classiquement des relations d'Allen [All81, All83] qui décrivent tous les agencements possibles d'intervalles et sont compatibles avec un modèle de temps continu adapté à des applications réelles. Parallèlement, la notion d'attribut d'évènement est formalisée de manière à pouvoir manipuler des données liées aux évènements du flux, puis, plus généralement, des données liées aux chroniques elles-mêmes. Ceci permet d'introduire plusieurs niveaux de complexité et de considérer des chroniques comme des évènements intermédiaires. La nécessité de pouvoir manipuler de telles données apparaît dès lors que l'on essaie de traiter des exemples d'application d'une légère complexité. En effet, il est primordial de pouvoir alors exprimer des corrélations entre des évènements différents, par exemple, de pouvoir spécifier qu'ils proviennent d'une même source. Pour cela, des chroniques exprimant des contraintes sur les attributs sont définies. De plus, afin de pouvoir considérer plusieurs niveaux d'évènements-chroniques, une fonction permettant de construire des attributs de chroniques est définie. L'ensemble de ces travaux est présenté dans le Chapitre 2.

On obtient alors un langage des chroniques expressif formalisé. Afin de pouvoir utiliser ce formalisme pour effectuer la reconnaissance de comportements, il faut définir des modèles de ce processus qui permettent l'utilisation du langage dans des applications quelconques. On doit pouvoir montrer que l'exécution de ces modèles respecte la sémantique des chroniques. Pour ce faire, on choisit de

développer deux modèles d'implémentations différentes.

Un premier modèle, dans la continuité de celui présenté dans [CCK11], permet de reconnaître les chroniques initiales de [CCK11] à l'aide de réseaux de Petri colorés, un langage de spécification formelle adapté à la reconnaissance de comportements. Pour compléter la construction formelle du modèle de [CCK11], on définit par induction cinq places centrales formant une structure clé de chaque réseau, permettant ensuite de composer ensemble les réseaux et donc de définir une construction formelle par induction des réseaux de reconnaissance. Le marquage des réseaux de Petri colorés construits pour chaque chronique évolue au fur et à mesure du flux d'évènements. Pour répondre au problème de la vérification de la correction du système vis-à-vis de la sémantique, on démontre que ce marquage correspond exactement à la définition formelle de [PBC$^+$]. Cette formalisation et cette preuve sont présentées dans [CCKP12a] et développés dans le Chapitre 3.

Cependant, l'intégralité des transitions des réseaux de Petri de ce modèle sont toujours tirables. Le modèle de reconnaissance est ainsi non déterministe dans le sens où il est nécessaire de tirer à la main les transitions dans un certain ordre, en suivant une stratégie de tirage bien définie, pour obtenir le résultat souhaité, c'est-à-dire l'ensemble de reconnaissance défini par la sémantique. On souhaite donc modifier ce modèle pour obtenir un marquage final unique, tout en conservant la double contrainte d'avoir d'une part une construction modulaire des réseaux calquée sur le langage, et d'autre part de conserver de la concurrence dans l'exécution des réseaux. On procède en deux temps. Tout d'abord, contrairement au premier modèle proposé dont les jetons contiennent des listes d'instances de reconnaissance, on passe à un modèle de réseau multi-jetons : un jeton pour chaque instance de reconnaissance afin d'initier l'implémentation d'un contrôle du réseau via des gardes sur les transitions. Dans un second temps, une structure de contrôle du flux d'évènements est implémentée pour achever la déterminisation du modèle tout en préservant la structure modulaire. La concurrence présente dans l'exécution des réseaux et l'unicité du marquage final après le traitement de chaque évènement du flux sont ensuite mis en avant en développant le graphe des marquages accessibles à l'aide du logiciel CPN Tools. Ces travaux sont exposés dans [CCKP13a] et développés dans le Chapitre 4.

Ce premier modèle de reconnaissance de chronique est cependant limité aux chroniques initiales de [CCK11] et ne permet donc pas de reconnaître des constructions temporelles ni de manipuler des attributs. Un second modèle de reconnaissance sur l'ensemble du langage étendu des chroniques est donc développé en C++ permettant ainsi une application directe. Ses algorithmes sont fidèlement calqués sur la sémantique du langage, justifiant ainsi le fonctionnement du modèle, CRL, qui est déposé sous la licence GNU LGPL.

La bibliothèque CRL est utilisée dans des applications du domaine des drones. On considère d'abord un système de drone (pilote, drone, Air Traffic Control (ATC)). On souhaite évaluer les procédures d'urgence liées à des pannes de liaisons entre les différents acteurs du systèmes. Ces procédures sont en cours de développement et l'on souhaite mettre en avant les éventuelles failles corrigibles, et proposer un système d'alarmes pour les failles humaines dont il est impossible de s'affranchir. On modélise le système et ses procédures par un diagramme UML implémenté en C++ puis on le soumet à des pannes de liaisons éventuellement multiples pour reconnaître les situations incohérentes pouvant donner lieu à un danger. CRL et cette application sont présentées dans [CCKP12b] et [CCKP13b], et développés dans le Chapitre 5.

Reconnaissance de comportements complexes par traitement en ligne de flux d'évènements

On réalise également une seconde application dans le domaine des drones. On considère un drone partant de l'aéroport d'Ajaccio pour une mission d'observation d'un feu en dehors de l'espace aérien contrôlé. Il doit passer successivement dans des zones de l'espace aérien contrôlé puis en sortir, et à chacune de ces actions sont associées des procédures de sécurité (points de passage, clearances, ...). L'objectif est de proposer un moyen de surveillance du drone assurant le respect des procédures. Pour ce faire, on écrit plusieurs niveaux de chroniques, où interviennent des constructions temporelles et des contraintes sur des attributs d'évènements complexes. On utilise CRL pour reconnaître ces chroniques dans des flux d'évènements tests comprenant un fichier de positions du drone produit par le battlelab BLADE de l'Onera [CBP10] ainsi qu'un fichier d'évènements élémentaires (clearances, changement de fréquence radio, ...).

Chapitre 1

Systèmes de traitement formel d'évènements complexes

Sommaire

On va s'intéresser dans ce chapitre à exposer les principales méthodes de traitement formel d'évènements complexes – Complex Event Processing (CEP) et plus généralement de l'Information Flow Processing (IFP) [1]. Ces systèmes se présentent en deux parties principales : il y a

1. La distinction entre CEP et IFP est réalisée dans [CM12], le CEP est présenté comme étant une partie de l'IFP qui contient également d'autres domaines comme celui des bases de données actives.

d'une part le moyen employé pour la description formelle des comportements à reconnaître, et d'autre part le processus de reconnaissance de ces comportements dans un flux d'évènements. Pour une introduction globale au domaine du traitement d'évènements complexes, nous renvoyons aux livres [Luc02, EN10]. D'autre part, [LSA+11] présente un glossaire des termes employés dans le domaine.

Nous allons commencer par dégager dans la Section 1.1 les principales problématiques de l'IFP. Dans les sections suivantes, nous présentons ensuite successivement quatre méthodes de reconnaissance d'évènements complexes qui sont représentatives de notre problématique : l'Event Calculus (EC) dans la Section 1.2, le langage ETALIS dans la Section 1.3, le langage des chroniques de Dousson *et al.* dans la Section 1.4, et le langage des chroniques Onera dans la Section 1.5. Dans une dernière section (1.6), nous présentons succinctement une sélection d'autres méthodes autour de l'IFP et donnons un aperçu des domaines d'application.

1.1 Principaux enjeux

On s'attache dans cette section à mettre en avant les principaux enjeux permettant de distinguer les différentes approches formelles de l'IFP.

Expressivité Il est nécessaire que le langage utilisé pour la description des comportements à reconnaître soit suffisamment expressif pour exprimer toutes les nuances désirées selon l'application envisagée. Cependant, il est clair qu'une grande expressivité ira inévitablement de pair avec une plus grande complexité du processus de reconnaissance. Il s'agit donc de trouver l'équilibre adéquat. La section 3.8 de [CM12] présente les principaux opérateurs et constructions que l'on rencontre dans la littérature. On retrouve notamment la conjonction, la disjonction, la séquence, la négation ou l'absence, l'itération, l'expression de contraintes temporelles, la manipulation de paramètres... La notion de négation ou d'absence est particulièrement épineuse : il est nécessaire de délimiter l'intervalle de temps précis sur lequel porte une négation ou une absence pour pouvoir déterminer si une telle construction est reconnue. Par ailleurs, la question d'avoir une syntaxe ouverte ou fermée se pose. Il s'agit d'autoriser, ou non, la possibilité d'écrire des formules n'ayant pas de sens.

Praticité d'écriture & lisibilité Dans le contexte du dialogue avec des experts pour la réalisation d'une application, la praticité d'écriture du langage est importante. On s'intéresse à la concision et à la lisibilité du langage qui faciliteront la discussion avec les spécialistes du domaine et leur utilisation du système de reconnaissance. Il s'agit là encore de trouver un équilibre entre lisibilité et expressivité : une lisibilité excessive peut conduire à une simplification extrême du langage et donc à une expressivité réduite.

Efficacité L'efficacité du processus de reconnaissance est cruciale. En effet, l'analyse des données se veut souvent être réalisée en ligne. Ceci implique la nécessité d'un temps de calcul réduit permettant de traiter les évènements du flux suffisamment rapidement par rapport à leur fréquence d'arrivée. Naturellement, la rapidité du processus est directement liée à la promptitude de la réponse au problème étudié qui peut être capitale par exemple s'il s'agit de produire une alarme lorsqu'une situation critique se produit.

Multiplicité On peut s'intéresser à établir *toutes* les reconnaissances d'un comportement donné, et non uniquement l'information que le comportement a eu lieu au moins une fois. La multiplicité des reconnaissances est nécessaire dans la perspective de recherche de comportements dangereux où il peut être essentiel de savoir qu'un comportement s'est produit plusieurs fois. Par exemple, dans certains domaines d'application comme la supervision de réseaux de télécommunications [Dou02], certaines pannes ne sont identifiables qu'à travers la multiplicité d'un comportement donné qui n'est en revanche pas significatif individuellement. En revanche, la multiplicité peut aller à l'encontre de l'efficacité du processus qui doit reconnaître toutes les occurrences et n'a donc pas le droit à l'oubli de reconnaissances intermédiaires. Pour pouvoir s'adapter à toutes les situations, on peut définir différents degrés d'exhaustivité des reconnaissances. Par exemple, [CM12] pose les contextes suivants qui établissent différentes gestions de la multiplicité des reconnaissances :

— dans le contexte « récent », seule l'occurrence la plus récente d'un événement qui débute une reconnaissance est considéré (ainsi, pour il y a toujours un évènement initiateur unique) ;
— dans le contexte « chronique », les plus anciennes occurrences sont utilisées en premier, et sont supprimées dès leur utilisation (ainsi, les évènements initiateur et terminal sont uniques) ;
— dans le contexte « continu », chaque évènement initiateur est considéré ;
— dans le contexte « cumulatif », les occurrences d'évènements sont accumulées, puis supprimées à chaque reconnaissance (ainsi, un évènement ne peut pas participer à plusieurs reconnaissances).

Historisation Il s'agit de pouvoir spécifier, pour chaque reconnaissance d'un comportement, quels sont les évènements du flux qui ont mené à cette reconnaissance. L'historisation permet alors de distinguer les différentes reconnaissances et donc d'y réagir plus finement. Il s'agit d'une forme de traçabilité qui apporte également la possibilité d'un retour d'expérience sur les causes du comportement détecté.

Traitement des évènements Plusieurs caractéristiques peuvent être requises par le système sur le flux d'évènements à traiter. Un flux unique totalement et strictement ordonné et dont les évènements arrivent dans l'ordre sans retard constitue un flux idéal, et l'algorithme de reconnaissance associé en sera a priori simplifié. Cependant les domaines d'applications peuvent exiger d'avoir la possibilité de considérer notamment :

— un ordre uniquement partiel entre les évènements à analyser ;
— des sources distribuées d'évènements – il faut alors définir correctement l'ordre entre les évènements provenant de deux flux différents ;
— un ordre non strict entre les évènements pour prendre en compte l'arrivée simultanée d'évènements ;
— des évènements arrivant en retard ou étant corrigés a posteriori – il s'agit alors de pouvoir malgré tout les traiter correctement. La gestion d'événements révisés est utile par exemple dans les cas d'application où un événement a été envoyé puis reçu, mais ensuite révisé suite à l'échec d'une transaction.

Centralisation ou distribution Le flux d'évènements à analyser peut provenir d'une source centralisée ou distribuée. Dans le cas d'un système distribué se pose la question de l'ordonnancement des évènements provenant des différentes parties du système. Par exemple, l'hypothèse d'une horloge synchronisée peut être prise.

Modèle de temps Le modèle de temps adopté est en général linéaire. Il peut être discret, avec une granularité adaptable, ou bien continu. Dans certains systèmes comme les Systèmes de Gestion de Flux de Données – Data Stream Management Systems (DSMSs), aucun modèle de temps particulier n'est utilisé et seul l'ordre entre les évènements est considéré mais cela a alors une incidence sur l'expressivité du langage de description de comportements utilisé.

Actions Il peut être possible de déclencher des actions à différents instants dans le processus de reconnaissance. Parmi les actions, on peut considérer entre autres la production d'un évènement associé à la reconnaissance d'un comportement et son intégration au flux. Se pose alors la question de l'ordre de traitement de ces nouveaux évènements par rapport aux autres évènements du flux. On peut également imaginer des actions d'une complexité plus importante : par exemple, ajouter dynamiquement un nouveau comportement à reconnaître.

Méthodes d'écriture L'écriture, dans le langage formel utilisé, des comportements à reconnaître présente une double difficulté qui dépend de l'expressivité et de la lisibilité du langage. D'une part, il faut exactement identifier toutes les situations à reconnaître pour répondre au problème étudié. D'autre part, il faut correctement transcrire ces comportements dans le langage utilisé. Des méthodes plus ou moins automatisées et principalement fondées sur des statistiques peuvent être proposées aux experts chargés d'écrire les comportements à reconnaître.

Gestion d'incertitudes Lors de l'analyse d'activités réelles, de nombreuses indéterminations apparaissent naturellement : incertitudes sur les comportements à reconnaître, incertitudes sur la date d'occurrence ou même sur l'occurrence elle-même d'évènements... Des mécanismes de gestion d'incertitudes peuvent donc être établis au niveau du langage de description et/ou au niveau de l'algorithme de reconnaissance, selon les indéterminations considérées.

1.2 Event Calculus

Le calcul d'évènements, ou EC, est une formalisation permettant la représentation et le raisonnement sur des évènements et des actions et pouvant s'exécuter sous la forme d'un programme logique. Il s'intéresse à établir la valeur de propositions logiques dans le temps.

L'EC est introduit par Kowalski et Sergot dans [KS86] et tire son nom du calcul de situation (Situation Calculus) dont il se distingue par le fait qu'il traite d'évènements locaux plutôt que d'états globaux. Le but est de s'affranchir du problème du cadre dans un souci d'efficacité. L'EC se veut constituer une analyse formelle des concepts mis en jeu, à savoir les évènements et les actions. Il peut être exprimé à partir de clauses de Horn [2] auxquelles est ajoutée la négation par l'échec [3],

2. On rappelle qu'une clause de Horn est une formule du calcul propositionnel de la forme $(p_1 \wedge \ldots \wedge p_n) \Rightarrow q$ où $n \in \mathbb{N}$, $p_1, \ldots, p_n$ sont des littéraux positifs et q est un littéral quelconque.

3. La négation par l'échec se fonde sur le principe que la base de connaissances est complète et que donc, si tous

se plaçant ainsi dans le cadre de la programmation logique.

Les principes initiaux majeurs de l'EC sont les suivants :

— les évènements peuvent être traités dans un ordre quelconque non nécessairement lié à leur ordre d'occurrence ce qui fait que le passé et le futur sont considérés symétriquement ;
— en particulier, des évènements peuvent être concurrents et ne sont pas forcément considérés comme ponctuels ;
— les mises à jour sont toujours additives dans le sens où elles ne peuvent pas retirer d'informations passées ;
— ce n'est pas tant la date des évènements qui est importante que leur ordre relatif.

Il existe de nombreux dialectes de l'EC fondés sur des axiomatiques proches et permettant de traiter diverses spécificités telles que la gestion d'actions différées ou de changements continus entre états. De telles axiomatiques sont recensées dans [MS99] et on peut également citer [PKB07, PB08].

Nous nous intéressons plus particulièrement à l'EC élaboré par Artikis *et al.* pour la reconnaissance de comportements. Ce dialecte, implémenté en programmation logique, est dédié à la reconnaissance de comportements complexes de haut niveau à partir d'évènements de bas niveau. Les comportements composites à reconnaître sont définis à l'aide de prédicats présentés Tableau 1.1 et permettant l'expression de contraintes temporelles sur un modèle de temps linéaire. Ces prédicats sont définis par des axiomes dont certains pouvant être indépendants du domaine d'application. Ce formalisme est expressif. Il permet d'exprimer des contraintes complexes aussi bien temporelles qu'atemporelles et de décrire des situations dans lesquelles un certain comportement ne doit pas avoir lieu dans un certain laps de temps [AP09]. Cette dernière caractéristique est liée au mécanisme de l'absence dans le formalisme des chroniques.

Selon les demandes de l'utilisateur, un comportement de haut niveau à reconnaître peut être défini comme un évènement ponctuel simple ou complexe à l'aide du prédicat `happensAt` ou comme un *fluent* – *i.e.* une propriété non ponctuelle pouvant prendre différentes valeurs dans le temps – à l'aide de `initially`, `initiatedAt`, `holdsFor`... Pour les activités non ponctuelles, il s'agit de calculer les intervalles maximaux durant lesquels un *fluent* possède une certaine valeur. Pour ce faire, tous les instants de début d'intervalle puis chaque instant de fin correspondant sont évalués. En effet, intuitivement, un principe d'inertie (exprimant qu'une valeur n'a pas changé tant qu'elle n'a pas été modifiée) est respecté dans le sens où l'on considère qu'un *fluent* vérifie une certaine valeur si celle-ci a été fixée par un évènement et que, depuis, aucun autre évènement ne l'a modifiée.

La question de l'écriture des comportements à reconnaître est étudiée dans [ASP10b, AEFF12]. En effet, l'écriture par des experts des activités à reconnaître est fastidieuse et fortement susceptible de donner lieu à des erreurs, donc il est intéressant de développer un procédé automatique pour engendrer des définitions à partir de données temporelles. Une telle méthode d'apprentissage fondée sur une combinaison d'abductions et d'inductions est utilisée dans [ASP10b, AEFF12] pour inférer les comportements à reconnaître.

Une problématique majeure de la reconnaissance de comportements est la possibilité ou non d'exécuter le processus de reconnaissance en temps réel. L'algorithme de base de ce dialecte d'EC fonctionne avec une méthode d'interrogation du système : le raisonnement ne se fait pas au fur

les moyens pour montrer une propriété échouent, c'est que sa négation est vérifiée (hypothèse du monde clos).

Tableau 1.1 – Principaux prédicats de l'EC

Prédicat	**Correspondance intuitive**
`happensAt`(E, T)	Occurrence de l'évènement E à l'instant T.
`initially`$(F = V)$	Le *fluent* F vaut V à l'instant 0.
`holdsAt`$(F = V, T)$	Le *fluent* F vaut V à l'instant T.
`holdsFor`$(F = V, I)$	I est la liste des intervalles maximaux où F vaut V.
`initiatedAt`$(F = V, T)$	À l'instant T, un intervalle de temps où F vaut V débute.
`terminatedAt`$(F = V, T)$	À l'instant T, un intervalle de temps où F vaut V s'achève.
`union_all`(L, I)	I est la liste des intervalles maximaux correspondant à l'union des ensembles d'intervalles de la liste L.
`intersect_all`(L, I)	I est la liste des intervalles maximaux correspondant à l'intersection mutuelle des ensembles d'intervalles de la liste L.
`relative_complement_all`(I', L, I)	I est la liste des intervalles maximaux correspondant à la différence ensembliste entre la liste d'intervalles I' et chaque ensemble d'intervalles de la liste L.

et à mesure mais à la demande de reconnaissance d'une activité de haut niveau [APPS10]. Pour réaliser une analyse en ligne, il est nécessaire de constamment interroger le programme ; et sans système de mémoire-cache, il faut à chaque fois recommencer les calculs à zéro. De plus, l'un des principes de base de l'EC, à savoir le fait que la reconnaissance de comportements ne doit pas être affectée par l'ordre dans lequel arrivent les évènements à analyser, ne contribue pas à diminuer la complexité de calcul. Dans [CM96], Chittaro *et al.* présentent une version de l'EC dénommée Cached Event Calculus (CEC) et dont l'implémentation inclut la gestion de mémoire-cache, réduisant ainsi significativement la complexité du processus. Cependant, comme le CEC n'a pas de système d'oubli ni de péremption et qu'il accepte des évènements datés plus tôt que d'autres évènements ayant déjà été traités, les temps de reconnaissance augmentent au fur et à mesure de l'arrivée des évènements de bas niveau à traiter et, après un certain temps, peuvent finir par ne plus respecter les temps de

calcul minimaux nécessaires pour une reconnaissance en ligne correcte.

Pour résoudre ces problèmes, Artikis *et al.* proposent dans [ASP12] une implémentation efficace de leur processus de reconnaissance, nommée Event Calculus for Run-Time reasoning (RTEC) [4] et réalisée sous YAProlog [5]. Le programme fonctionne toujours par interrogations successives du système, et comme pour le CEC, un dispositif de mémoire-cache est mis en place pour stocker les intervalles maximaux calculés par les prédicats `holdsFor` pour chaque *fluent* F. Ceci permet de n'avoir pas à recalculer ces intervalles à chaque itération. La problématique du calcul vient ensuite du fait que des évènements peuvent ou bien arriver en retard, ou bien être corrigés a posteriori. L'algorithme est élaboré de façon à pouvoir traiter correctement ce genre de situation. Deux paramètres du programme sont à fixer :

— d'une part, le pas $Q_{i+1} - Q_i$ entre deux interrogations Q_i et Q_{i+1} du système ;
— d'autre part, la taille WM (Working Memory) de la fenêtre des évènements à considérer.

À chaque itération Q_i les évènements datés dans l'intervalle $]Q_i - WM, Q_i]$ sont analysés. Ainsi, dans le choix des paramètres, le signe de la différence $WM - (Q_{i+1} - Q_i)$ est significatif :

— si $WM < Q_{i+1} - Q_i$, alors les évènements de la fenêtre $]Q_i, Q_{i+1} - WM]$ ne seront jamais analysés, et si un évènement arrive en retard ou est corrigé, il ne sera pas considéré ;
— si $WM = Q_{i+1} - Q_i$, alors exactement tous les évènements arrivés à temps seront analysés ;
— si $WM > Q_{i+1} - Q_i$, alors le système peut traiter des évènements arrivant ou bien avec un certain délai ou bien corrigés dans un certain délai. L'algorithme fonctionne alors comme suit. Les parties des intervalles maximaux calculés auparavant et s'intersectant avec la fenêtre $]Q_i - WM, Q_i]$ sont tronqués. Ensuite, les évènements de la fenêtre $]Q_i - WM, Q_i]$ sont analysés pour recalculer des intervalles maximaux sur cette fenêtre, éventuellement certains identiques à ceux ayant été tronqués. Enfin, les morceaux sont « recollés » afin de recréer des intervalles maximaux globaux.

L'algorithme plus détaillé de RTEC est présenté dans [ASP12], avec une analyse des performances du système sur des données réelles et sur des données synthétisées.

Dans [AMPP12, PAMP12], une architecture de système pour la reconnaissance de comportements, Event Processing for Intelligent Ressource Management (EP-IRM), est proposée. Elle peut être dotée de nombreux composants pouvant facilement être ajoutés ou supprimés. Certaines applications sont indépendantes du domaine et peuvent être utilisées quelle que soit l'étude, comme par exemple l'affichage de cartes MAP. Le système est également composé d'un moteur détectant les évènements de bas niveau qui sont ensuite envoyés au moteur de reconnaissance de comportements complexes.

Par ailleurs, plusieurs approches probabilistes de l'EC sont développées. Tout d'abord, Artikis *et al.* étendent dans [SPVA11] le formalisme de l'EC au raisonnement probabiliste à l'aide de Réseaux Logiques de Markov [DL09] qui combinent l'expressivité de la logique du premier ordre avec la sémantique formelle probabiliste des réseaux de Markov. D'autre part, dans [SAFP14], Artikis *et al.* proposent une adaptation de l'EC au cadre de la programmation logique probabiliste en utilisant ProbLog [KDDR+11]. Ceci permet de répondre au problème de détections incorrectes des

4. RTEC est disponible en open-source sur le web : `http://users.iit.demokritos.gr/~a.artikis/RTEC-3examples.zip`.
5. `http://www.dcc.fc.up.pt/~vsc/Yap/`

évènements de bas niveau en travaillant sur un flux d'évènements où chaque évènement est doté d'un indice de confiance. En revanche, la reconnaissance en ligne n'est pas encore réalisable avec ce formalisme, et il ne permet pas de traiter le cas des définitions imprécises des comportements à reconnaitre. Une autre approche pour la gestion d'incertitude est présentée dans [AWG+13]. Elle est orthogonale à celle de [SAFP14] dans le sens où elles peuvent être combinées. Le principe consiste à utiliser la variété des sources d'évènements pour déterminer leur véracité. Un système d'« auto-adaptation » est implémenté en utilisant le processus de reconnaissance de comportements lui-même. En effet, des définitions de comportements complexes sont écrites pour identifier les domaines d'incertitude et y réagir : lorsque l'incertitude est significative, le système peut ignorer les évènements d'un certain laps de temps ou bien, momentanément, ne pas considérer certaines sources d'évènements. [AWS+14] répond à la même problématique en ayant en plus recours à du *crowdsourcing* pour trancher lorsque les désaccords entre les sources sont significatifs.

1.3 Le langage ETALIS

Le langage de description de comportements Event-driven Transaction Logic Inference System (ETALIS) [6] [ARFS12, Ani11] est un langage de CEP muni d'une syntaxe et d'une sémantique formelles permettant simultanément de raisonner sur des connaissances temporelles (concernant des évènements) et sur des connaissances stables ou en évolution (règles, faits, ontologies, données encyclopédiques...), et ce à l'aide d'un système réalisant l'analyse de comportements *en ligne*.

ETALIS est un langage de programmation logique. Sa syntaxe est définie par des règles dont les principales constructions sont présentées dans le Tableau 1.2. Le modèle de temps adopté est linéaire et dense mais dénombrable (l'ensemble des rationnels $\mathbb{Q}$) ; les évènements de base du flux à analyser peuvent être aussi bien des évènements instantanés que des évènements ayant une durée, ils sont datés par des intervalles de temps $[T_1, T_2]$ où éventuellement $T_1 = T_2$. Le langage présente une expressivité forte :

- l'ensemble des 13 relations d'Allen peut être décrit,
- des contraintes peuvent être exprimées sur des propriétés d'évènements,
- une notion d'absence est développée, mais limitée au cadre de la séquence,
- une distinction précise est faite entre la conjonction en série et celle en parallèle,
- il est possible de réaliser des définitions récursives de comportements ce qui permet, par exemple, d'accumuler à l'aide d'une fonction une valeur sur une suite d'évènements.

Une première sémantique déclarative formelle est fournie [AFR+10, ARFS12] ; les interprétations de motifs (*patterns*) d'évènements (*i.e.* des comportements à reconnaître) est définie par induction à la manière de la théorie des modèles. Il s'agit d'ensembles de reconnaissances où une reconnaissance est représentée par un couple $\langle q_1, q_2 \rangle$, avec $q_1, q_2 \in \mathbb{Q}$, délimitant l'intervalle de temps nécessaire et suffisant à la reconnaissance. Les informations relatives aux évènements ayant donné lieu à la reconnaissance ne sont pas conservées, il n'y a donc pas de possibilité d'historisation.

Le système de reconnaissance ETALIS est implémenté en Prolog. Pour ce faire, une seconde sémantique, opérationnelle, est définie à l'aide de règles de programmation logique. Les compor-

6. ETALIS est disponible en open-source sur le web : `http://code.google.com/p/etalis/`.

TABLEAU 1.2 – Principales constructions du langage ETALIS [AFR+10]

Constructions	**Correspondance intuitive**
p WHERE t	Le comportement p est reconnu et le terme t prend la valeur vraie.
q	Pour tout $q \in \mathbb{Q}$, correspond à l'instant absolu q.
$(p).q$	Le comportement p est reconnu et dure exactement q, avec $q \in \mathbb{Q}$.
p_1 SEQ p_2	Le comportement p_1 est suivi, strictement (dans le temps), du comportement p_2.
p_1 AND p_2	Les comportement p_1 et p_2 sont reconnus, sans aucune contrainte temporelle.
p_1 PAR p_2	Les comportement p_1 et p_2 sont reconnus en parallèle, *i.e.* ils se chevauchent dans le temps.
p_1 OR p_2	L'un des deux comportements est reconnu.
p_1 EQUALS p_2	Les deux comportements sont reconnus sur exactement le même intervalle de temps.
p_1 MEETS p_2	Le dernier instant de reconnaissance de p_1 est exactement le premier instant de reconnaissance de p_2.
p_1 DURING p_2	Le comportement p_1 est reconnu pendant le comportement p_2.
p_1 STARTS p_2	L'intervalle de reconnaissance de p_1 est un segment initial de l'intervalle de reconnaissance de p_2.
p_1 FINISHES p_2	L'intervalle de reconnaissance de p_1 est un segment final de l'intervalle de reconnaissance de p_2.
NOT$(p_1).[p_2, p_3]$	Les comportements p_2 et p_3 sont reconnus dans cet ordre, sans occurrence de p_1 strictement entre les deux dans le temps.

tements complexes recherchés sont décomposés en des évènements intermédiaires appelés *buts* (goals). ETALIS compile les comportements complexes pour fournir un ensemble de règles permettant l'Event-Driven Backward Chaining (EDBC) – chaînage arrière piloté par les données. C'est le chaînage arrière piloté par les données qui rend possible un processus de reconnaissance en ligne. Deux types de règles résultent de la compilation [AFSS09] :

— D'une part, des règles créant les buts à reconnaître pour avancer dans la reconnaissance du comportement complexe. Les buts créés représentent l'occurrence d'un évènement et l'attente d'un autre évènement pour reconnaître un évènement (complexe). Elles sont de la forme $\texttt{goal}(b^{[-,-]}, a^{[T_1,T_2]}, ie_1^{[-,-]})$ qui exprime qu'à la reconnaissance d'un évènement (éventuellement complexe) a sur l'intervalle $[T_1, T_2]$, le système est en attente d'un évènement b pour reconnaître l'évènement ie_1.

— D'autre part, des règles créant des évènements intermédiaires ou des patterns d'évènements. Celles-ci vérifient dans la base de données si un certain but existe déjà, et, s'il existe effectivement, déclenchent l'évènement qui a été reconnu par le but. Par exemple, si le but $\texttt{goal}(b^{[T_3,T_4]}, a^{[T_1,T_2]}, ie_1^{[-,-]})$ figure dans la base de données, alors l'évènement $ie_1^{[T_1,T_4]}$ est déclenché et ensuite propagé s'il s'agit d'un évènement intermédiaire ou utilisé pour déclencher une action s'il s'agit de l'un des comportements complexes recherchés. Les règles de ce type permettent également de supprimer de la base de données les buts qui ne sont plus nécessaires car obsolètes.

En d'autres termes, chaque but correspond à un sous-comportement restreint à deux évènements (dans les exemples précédents il s'agit de a et b), et les buts sont chaînés pour aboutir à la reconnaissance du comportement complexe recherché. La structure de la décomposition d'un comportement complexe en buts correspond donc essentiellement à celle d'un arbre binaire.

Il n'y a pas de démonstration d'équivalence entre les deux sémantiques. Les auteurs s'assurent en revanche que la sémantique opérationnelle est belle et bien déclarative [ARFS12], et que donc le comportement du système est à la fois prédictible et reproductible.

En ce qui concerne la multiplicité des reconnaissances, ETALIS permet l'implémentation des politiques de consommation d'évènements suivantes (introduites dans la Section 1.1) : contexte récent, contexte chronique et contexte libre (*i.e.* sans restriction, multiplicité totale). Mais il faut noter que l'on perd l'aspect déclaratif si l'on utilise une autre politique que celle dite libre, et l'ordre d'évaluation des règles devient alors significatif.

Une analyse des performances d'ETALIS est présentée dans [AFR+10] sur un cas d'étude analysant l'efficacité d'un système de livraison de fleurs (*Fast Flower Delivery Use Case* [EN10]).

Par ailleurs, la nature logique de l'approche rend possible un raisonnement déductif en temps réel sur une base de connaissances fixe de données structurées de l'environnement. Celle-ci permet de définir une sémantique de l'information utilisable par un système.

ETALIS permet la gestion d'évènements arrivant en retard dans un flux ordonné par rapport au temps [FAR11] grâce à l'ajout de deux types de règles :

— Des règles de type $\texttt{goal\_out}(a^{[-,-]}, b^{[T_3,T_4]}, ie_1^{[-,-]})$ exprimant que l'évènement b a été reçu et que a est en attente, mais *avant* b, pour réaliser la reconnaissance de ie_1.

— Des règles de la forme $\texttt{if goal\_out}(\dots)\ \texttt{and}\ T_2 < T_3$, alors si un évènement a arrive avec effectivement $T_2 < T_3$, l'événement $ie_1^{[T_1,T_4]}$ est déclenché.

Un tel algorithme a la particularité de ne pas gérer les évènements en retard au détriment de l'efficacité du processus de reconnaissance pour les évènements arrivant à temps : la reconnaissance est toujours quasi-immédiate si tous les évènements composant un comportement complexe sont arrivés à temps. En revanche, le système nécessite la mise en place d'une procédure de libération de mémoire assurant la suppression des règles de type `goal_out` après un certain laps de temps. [FAR11] précise que, pour « des raisons pratiques » (sûrement une question d'efficacité de reconnaissance par surcharge de règles), cette fonctionnalité n'a pas été implémentée pour la politique de consommation d'évènements dite libre ; on perd donc la multiplicité des reconnaissances. Pour mener les cas de retard d'un évènements interdit dans une absence, la gestion d'arrivée d'évènements en retard doit être couplée avec la gestion d'évènements révisés, présentée brièvement ci-dessous. En effet, si l'on considère une négation $\text{NOT}(c).[a, b]$ et qu'une reconnaissance est invalidée par l'arrivée d'un c en retard, il faut alors *réviser* la reconnaissance erronée.

ETALIS permet également la gestion d'évènements *révisés* [ARFS11]. Comme évoqué en 1.1, ceci peut s'avérer utile dans le cas d'un échec d'une procédure amenant ainsi la correction d'un évènement déjà envoyé. Des sortes de marqueurs sont ajoutés, permettant d'indiquer quels évènements (éventuellement intermédiaires) ont donné lieu à quels buts. Ceci permet d'identifier les instances d'évènements à réviser et de propager correctement la révision dans tout le système. Des règles `rev` sont ajoutées pour supprimer les buts (`goal`$(\dots)$) insérés mais révisés.

1.4 Le langage des chroniques de Dousson *et al.*

Le langage des chroniques a été développé pour décrire formellement une signature évènementielle et offre un cadre pour le traitement d'évènements complexes – CEP. Le terme de « chronique » est un terme générique englobant plusieurs systèmes formels voués à la description et à la reconnaissance de comportements. On décrit, dans cette sous-section, un premier formalisme de la notion de chronique.

Un langage des chroniques a été introduit notamment dans [Gha96] et développé principalement par Dousson *et al.* [DGG93, Dou02, DLM07]. Il permet de décrire formellement des agencements d'évènements. Une chronique est en quelque sorte un ordre partiel d'évènements observables dans un certain contexte. Le langage est doté d'un processus de reconnaissance en ligne efficace permettant d'analyser un flux d'évènements datés mais dont les évènements n'arrivent pas nécessairement dans leur ordre d'occurrence pour y reconnaître des situations et éventuellement déclencher des actions ou produire un évènement à une date définie relativement aux dates des évènements ayant donné lieu à la reconnaissance.

On manipule des prédicats exprimant le fait qu'un attribut fixé ait une certaine valeur sur un intervalle de temps donné. Un *motif d'évènement* correspond à un changement de valeur d'un attribut, et un *évènement* est alors une instance datée ponctuelle de motif d'évènement [DGG93]. Le modèle de temps adopté est linéaire et discret, la résolution adoptée étant supposée suffisante pour le contexte étudié. Une distinction est opérée entre la date d'occurrence et la date de réception des évènements observés. Une borne sur le délai de réception d'un évènement est fournie par l'utilisateur pour chaque motif d'évènement. Ceci permet, comme dans l'Event Calculus, de considérer des

évènements arrivant avec un retard borné. Par ailleurs, de même que pour l'Event Calculus, on suppose la complétude du flux d'évènement, c'est-à-dire qu'entre deux activations d'une propriété il y a nécessairement un évènement de désactivation de cette propriété (par rapport aux dates d'occurrences).

Un *modèle de chronique* à reconnaître [DGG93] est composé :

— d'un ensemble de motifs d'évènements exprimés par les prédicats du Tableau 1.3,

— de contraintes temporelles sur ces motifs d'évènements reliées par une conjonction implicite (pour des raisons de complexité, il n'est pas possible d'exprimer de disjonction),

— d'un contexte général permettant d'exprimer des contraintes contextuelles,

— d'éventuelles actions à réaliser à la reconnaissance (notamment la production d'évènements).

Dans [Dou02], le prédicat `occurs` présenté dans le Tableau 1.3 est ajouté pour répondre à la question du comptage d'évènements qui est un problème typique du domaine de la gestion d'alarmes où certaines pannes ne sont identifiables que par comptage. `occurs` permet de faciliter l'écriture de ce genre de chroniques et d'optimiser le processus de reconnaissance associé. En effet, sinon, il faut écrire beaucoup de chroniques pour exprimer la même situation, or la complexité du processus de reconnaissance dépend du nombre de chroniques à étudier.

Tableau 1.3 – Principaux prédicats et notations des chroniques [Dou96, Dou02]

Prédicat	**Correspondance intuitive**
`hold`$(P : v, (t_1, t_2))$	Le nom d'attribut P a la valeur v sur l'intervalle $[t_1, t_2]$, sans implication sur l'instant où il a pris cette valeur.
`event`$(P : (v_1, v_2), t)$	L'attribut P passe de la valeur v_1 à la valeur v_2 à l'instant t.
`noevent`$(P, (t, t'))$	P ne change pas de valeur sur l'intervalle $[t, t'[$.
`occurs`$(n_1, n_2, a, (t_1, t_2))$	L'évènement a a lieu exactement N fois, avec $n_1 \leq N \leq n_2$, dans l'intervalle de temps $[t_1, t_2[$. (permet l'unification du langage)
Notation	
`?`	« `?` » permet d'indiquer une valeur quelconque.

Le processus de reconnaissance de chroniques est illustré par la Figure 1.4. Dans un premier temps, il s'agit de compiler hors ligne les modèles de chroniques fournis par l'utilisateur afin de transcrire les contraintes temporelles de chaque modèle en un graphe de contraintes minimal (la relation d'ordre pour laquelle ce graphe est minimal est définie dans [DD99]). Ce pré-traitement permet notamment de mettre en avant les incohérences éventuelles des modèles de chroniques (l'algorithme employé pour la compilation étant incrémental, il désigne même un sous-ensemble de contraintes incohérentes). L'algorithme de reconnaissance est fondé sur ces graphes.

Le système doit ensuite être initialisé avec les états initiaux du monde considéré, datés à $-\infty$, puis la reconnaissance de chroniques peut être lancée. Des instances partielles, *i.e.* des reconnaissances encore non complètes, sont manipulées. Elles sont chacune en attente d'évènements manquants pour compléter la reconnaissance. Ces évènements manquants doivent arriver dans un certain intervalle de temps, la *fenêtre d'admissibilité*, calculée grâce au graphe de contraintes et

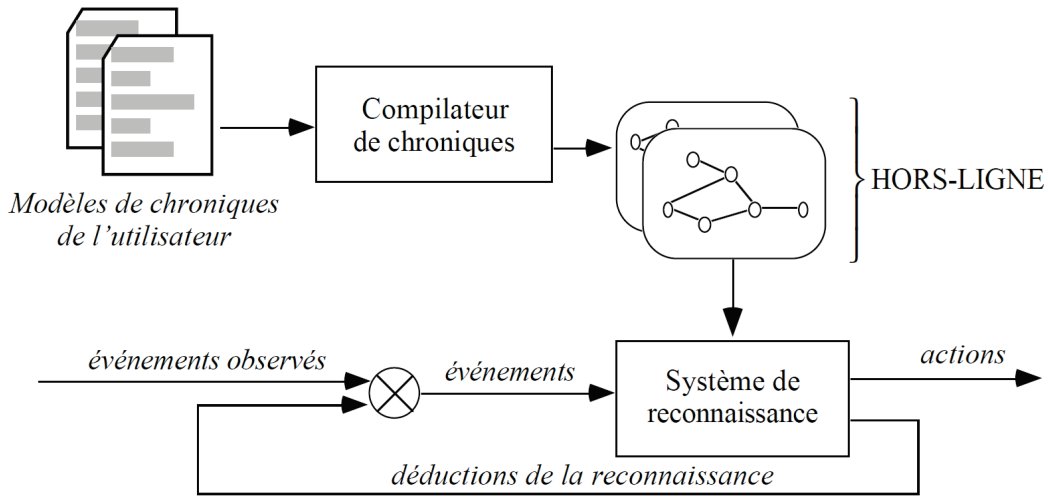


FIGURE 1.4 – Le système de reconnaissance de chroniques [Dou94]

ce afin de vérifier les contraintes temporelles spécifiées dans la chronique correspondante. Si un évènement attendu n'arrive pas avant la fin de sa fenêtre d'admissibilité ou si l'une des conditions générales de la chronique n'est plus vérifiée, l'instance partielle est alors abandonnée et supprimée. Au contraire, lorsqu'une instance partielle s'apprête à être complétée par un évènement, elle est d'abord dupliquée afin de garantir la reconnaissance de *toutes* les situations : l'instance partielle complétée peut ainsi être de nouveau complétée par une autre occurrence d'un évènement similaire. La duplication d'instances est la principale source de complexité du processus de reconnaissance. Afin d'optimiser la manipulation des instances (partielles) de reconnaissance, celles-ci sont stockées dans un arbre. Au fur et à mesure de l'arrivée des évènements pertinents les contraintes temporelles exprimées par les fenêtres d'admissibilité du graphe de contraintes sont mises à jour et les modifications sont propagées dans le reste du graphe. Ceci permet de traiter directement l'arrivée de tout évènement. Une propagation des modifications est également effectuée lorsque le temps avance : le système calcule le prochain instant critique et une mise à jour est effectuée lorsque le temps courant atteint cet instant [DGG93]. Le processus de reconnaissance est donc exhaustif, et le plus efficace pour diminuer sa complexité est de limiter la durée de validité d'une instance de chronique [AD01].

Le système peut également prendre en entrée, en plus des évènements datés, l'assertion `AssertNoMore`(e, I) où e est un évènement et I est un intervalle étendu (*i.e.* une union disjointe d'intervalles) [DLM07]. Cette assertion indique qu'aucun évènement e n'aura lieu dans I. À la réception d'un tel message, les fenêtres d'admissibilité sont mises à jour en leur appliquant la différence ensembliste avec I et les modifications sont propagées dans le graphe. Cette assertion est introduite dans le but d'être utilisée à des fins d'optimisation, ce qui sera détaillé par la suite.

Pour le prédicat $\texttt{occurs}(n_1, n_2, a, (t_1, t_2))$, un compteur est associé au prédicat et tant que l'instant t_2 n'est pas atteint, il n'est pas possible de déterminer si le prédicat est vérifié. À l'arrivée d'un évènement a à la date d, les instants t_1 et t_2 n'ont pas encore nécessairement de valeur mais sont contraints à des intervalles $[I^-_{t_1}, I^+_{t_1}]$ et $[I^-_{t_2}, I^+_{t_2}]$. On étudie alors tous les cas d'ordonnancement de d, $I^-_{t_1}$, $I^+_{t_1}$, $I^-_{t_2}$, et $I^+_{t_2}$.

Ce système de reconnaissance, appelé Chronicle Recognition System (CRS), est construit autour du gestionnaire de graphes temporels IxTeT [GL94]. Il permet ainsi de reconnaître *en ligne* et efficacement *toutes* les occurrences des chroniques spécifiées. De plus, ce système est prédictif. En effet, l'utilisateur peut savoir à tout moment quand une reconnaissance est attendue et quels évènements sont requis pour compléter chaque instance partielle. Une définition plus formelle de la notion de chronique et du système de reconnaissance associé est présentée dans [DG94].

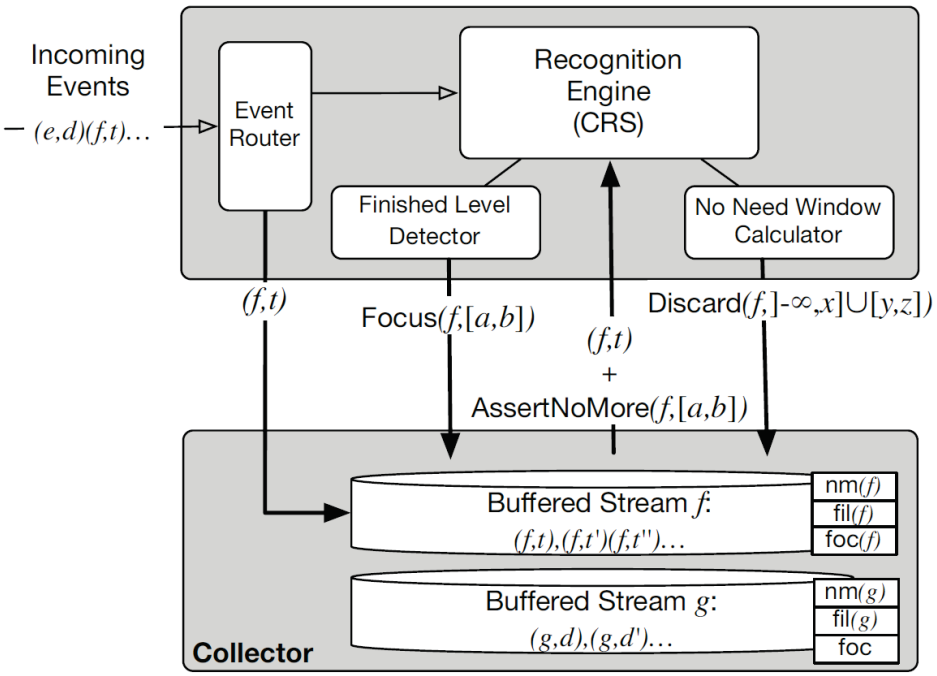


Figure 1.5 – Architecture du système de reconnaissance avec focalisation temporelle [DLM07]

Dans [DLM07], Dousson *et al.* proposent une technique d'optimisation de CRS. Il s'agit tout d'abord d'appliquer une méthode de focalisation temporelle qui permet d'optimiser les situations où certains évènements sont beaucoup plus fréquents que d'autres. Afin de limiter le nombre d'instances partielles créées suite à un évènement fréquent en attente d'un évènement rare, on définit un *niveau* pour chaque type d'évènement. Ceci permet d'établir un critère d'intégration au système de reconnaissance : un évènement de niveau $n+1$ n'est pas intégré à une instance donnée

tant que tous les évènements des niveaux 1 à n n'ont pas été intégrés. Lorsqu'un évènement est intégré à une instance, il est envoyé à toutes les autres instances même si la règle n'est pas vérifiée. Ceci permet de s'assurer que tout évènement est traité exactement une fois. La structure du système est présentée Figure 1.5. Un composant de gestion des évènements est introduit. Les évènements qui ne sont pas immédiatement intégrés sont envoyés au *collectionneur* qui les stocke dans des flux tampons (il y a un buffer par type d'évènement). Chaque flux tampon manipule trois fenêtres temporelles : la fenêtre *assert no more* où plus aucun évènement ne sera reçu, la fenêtre de *filtrage* qui contient les occurrences d'évènements ne convenant à aucune instance, et la fenêtre de *focalisation* qui contient les dates d'occurrence auxquelles un évènement est attendu et devrait donc être immédiatement envoyé à CRS. Ce mécanisme fournit exactement les mêmes reconnaissances qu'avec la version initiale de CRS et a uniquement un effet sur les performances. Celles-ci ne sont pas améliorées systématiquement ; cela dépend de la fréquence des évènements et de leur position dans le graphe de contraintes.

La seconde partie de la méthode est d'introduire un principe de chroniques hiérarchiques qui permet de définir séparément des sous-chroniques et de les intégrer dans une chronique plus complexe. La méthode de focalisation temporelle peut ensuite être appliquée aux sous-chroniques elles-mêmes.

Dans le formalisme des chroniques, le processus d'écriture des situations à reconnaître reste une difficulté centrale. Plusieurs réponses y sont apportées. Un système, Frequency Analyser for Chronicle Extraction (FACE) [DD99], permettant à un expert d'analyser des fichiers de logs d'alarmes pour identifier les phénomènes récurrents et ainsi définir des chroniques est développé. A partir d'une grandeur $f_{q_{\min}}$ fournie par l'utilisateur, on définit une notion de chronique fréquente. On construit ensuite par induction sur la taille des chroniques les chroniques fréquentes dans le flux d'évènements étudié, puis les contraintes temporelles associées à l'aide d'un algorithme favorisant certaines contraintes. Partir d'une base de chroniques déjà posée par des experts permet de diminuer considérablement le temps de calcul. Il s'agit ensuite de filtrer l'ensemble des chroniques fréquentes obtenues : pour déterminer s'il est intéressant de chercher à reconnaître à la fois une chronique et l'une de ses sous-chroniques, une notion de dépendance est définie. [FCD04] propose une méthode de pré-traitement des fichiers pour en extraire des sous-fichiers appropriés et ainsi alléger la saturation de mémoire provoquée par FACE.

Une seconde méthode d'aide à l'écriture de chroniques est fondée sur une simulation du système qui permet de faire apparaître des configurations caractéristiques d'évènements. Ceci permet de récolter les séquences d'évènements datés associés et ainsi de former, pour chaque configuration, une liste de séquences positives (*i.e.* liées à la configuration) et une liste de séquences négatives. Une méthode de programmation logique inductive (ILP) [MDR94] peut ensuite être appliquée sur ces deux listes pour en dériver des chroniques [CD00]. Les techniques d'Inductive Logic Programming (ILP) peuvent être également utilisées directement sur des flux d'évènements, combinées avec l'Inductive Constraint Logic (ICL) [DRVL95] qui permet l'expression de contraintes sur le type de chroniques à apprendre, assurant ainsi une caractérisation précise des situations à reconnaître [QCCW01].

Dans [DG94], Dousson *et al.* commencent à introduire une notion d'incertitude autour de la datation des évènements du flux en utilisant des ensemble flous pour exprimer les ensembles de dates possibles pour un évènement.

Subias *et al.* partent du formalisme des chroniques de Dousson *et al.* et l'adaptent aux systèmes distribués. En effet, il est difficile de développer des mécanismes capables de dater des évènements dans un système distribué, et, de plus, des délais de transmission sont à prendre en compte. Il est donc intéressant de subdiviser l'architecture de contrôle en sous-sites de contrôle pouvant se chevaucher par endroits pour faciliter le diagnostic. Pour ce faire, Subias *et al.* définissent dans [BSC02] une notion de sous-chronique comme extension de la notion de chronique. Une sous-chronique est composée d'un ensemble d'évènements E et d'un ensemble de contraintes sur les évènements de E mais aussi sur des évènements extérieurs. Une chronique peut alors se décomposer en sous-chroniques, avec une sous-chronique correspondant à chaque sous-site de contrôle et telles que l'ensemble des évènements E de chaque sous-chronique doit être inclus dans l'ensemble des évènements observables par le sous-site de contrôle concerné. Alors, une chronique est reconnue lorsque toutes les sous-chroniques sont reconnues. Une sous-chronique possède deux types de contraintes : les contraintes locales portant uniquement sur les évènements de E, et les contraintes globales faisant intervenir des évènements extérieurs.

Le cadre des réseaux de Petri p- et t-temporels (réseaux de Petri classiques auxquels deux types de mécanismes, détaillés ci-dessous, ont été ajoutés [Kha97, BD91]) est choisi pour modéliser le processus de reconnaissance distribué car, d'après [BSC05] :

— il offre une visualisation claire de l'état courant de la reconnaissance de la chronique/sous-chronique ;
— il est approprié pour simuler l'évolution d'une chronique à l'aide d'outils, ou pour revenir en arrière ;
— les occurrences multiples sont facilement représentables ;
— ils pourraient permettre de démontrer la correction du modèle de contraintes temporelles.

Les mécanismes t-temporels (contraintes temporelles de type intervalle sur les transitions) sont utilisés pour exprimer les contraintes de fenêtres d'admissibilité de la chronique (contraintes de type $1 \leq \min_p(d(e_1) - d(e_p)) \leq 3$). Les mécanismes p-temporels (contraintes temporelles sur la durée de séjour admissible d'un jeton dans une place) permettent quant à eux l'expression des contraintes de type intervalle (*i.e.* du type $1 \leq d(e_1) - d(e_2) \leq 3$). Chaque type de contrainte est transposé en une brique de réseau de Petri temporel. Un réseau correspondant à une chronique est réalisé en fusionnant les briques de réseau correspondant à chaque contrainte de la chronique. Dans le réseau obtenu, il n'y a pas de situation de conflit structurel (*i.e.* il n'y a pas de place précédant plusieurs transitions) car les vérifications de contraintes doivent être réalisées de manière indépendantes. Chaque jeton du réseau obtenu correspond à une instance (partielle ou non) de la chronique. Les occurrences d'évènements sont représentées par des transitions, et les jetons (instances partielles) sont dupliqués et complétés au tirage de ces transitions pour obtenir *in fine* toutes les reconnaissances complètes [BSC02].

Dans le cas des sous-chroniques, la problématique principale est la vérification des contraintes globales. Pour vérifier celles-ci, la sous-chronique doit recevoir les informations adaptées de l'extérieur. Les transitions et places correspondantes sont donc également ajoutées au réseau. La problématique du délai de transmission entre les sous-sites de contrôle est étudiée dans [BSC04]. Le centre de la question est que ce délai induit une incertitude sur la vérification des contraintes. La méthode employée est la suivante. Le délai Δ de transmission est supposé borné. Les contraintes glo-

bales du système sont réécrites sous forme d'expressions dépendant de mesures locales (comme les contraintes locales) et des bornes du délai Δ. La possibilité, entre 0 et 1, de vérifier la contrainte est ensuite quantifiée : on obtient des ensembles flous de valeurs permettant de vérifier les contraintes. Dans les cas où la valeur n'est ni 0 ni 1 mais dans]0,1[, [BSC04] propose un système de coopération qui peut être lancé pour que la contrainte soit vérifiée par un autre sous-site de contrôle. Pour mieux manipuler ces délais temporels, [BSC05] passe aux réseaux de Petri p- et t-temporels flous.

1.5 Le langage des chroniques Onera

1.5.1 Une première implémentation : CRS/Onera

Ornato et Carle présentent une première notion de chronique dans [OC94a, OC94b]. Il s'agit de répondre à la problématique de l'automatisation de la reconnaissance des intentions d'un agent considéré comme une boîte noire. Contrairement au domaine de la reconnaissance de plans, les auteurs ne font pas d'hypothèse forte sur le sujet observé : il n'est pas nécessaire d'avoir une base de connaissance exhaustive décrivant les différents plans pouvant être suivis par le sujet, et, en particulier, le sujet n'est pas supposé effectuer les plans sans erreur ni n'en suivre qu'un seul à la fois. De plus, les auteurs souhaitent pouvoir exprimer des notions telles que la suspension ou l'abandon d'un but. Dans cette optique, un système de reconnaissance de comportements, Chronicle Recognition System/Onera (CRS/Onera) est implémenté. Il est fondé sur un langage temporel, le langage des chroniques, qui permet la description de comportements complexes à l'aide d'évènements typés pouvant être dotés de propriétés et des opérateurs suivants :

- la séquence, le non-ordre (conjonction) et la disjonction,
- la non-occurrence d'une chronique sur un intervalle de temps délimité par une seconde chronique (notons qu'il n'y a pas de garantie d'ordre de traitement entre les deux chroniques à l'arrivée d'un évènement et qu'il y a donc des formes indéterminées),
- une notion de délai,
- un opérateur de coupure, le cut, permettant de réduire la combinatoire due à l'exhaustivité recherchée dans le processus de reconnaissance : le cut désigne uniquement la première reconnaissance dans le flux,
- opérateur d'indexation d'évènements permettant d'identifier à une unique occurrence plusieurs évènements de même nom dans une chronique (les opérateurs de non occurrence et de disjonction sont cependant opaques pour l'indexation).

Le système de reconnaissance CRS/Onera se voit imposer trois contraintes principales :

1. les reconnaissances doivent être exhaustives (*i.e. toutes* les reconnaissances possibles doivent être détectées) ;
2. il doit y avoir une historisation des évènements (*i.e.* il faut être capable de dire quels évènements sont à l'origine de chaque reconnaissance) ;
3. le processus de reconnaissance doit être suffisamment efficace pour être réalisé en ligne.

Pour répondre à ces contraintes, l'algorithme de CRS/Onera a été conçu sur la base d'automates dupliqués représentant les instances éventuellement partielles des chroniques à reconnaître. Chaque

reconnaissance partielle de chronique est dupliquée et mise en attente des évènements attendus pour la complétion de la reconnaissance. Cette duplication permet à la fois d'assurer l'exhaustivité du processus et également de gérer les constructions de non-occurrence.

CRS/Onera propose également la gestion d'actions et de tests à la reconnaissance. Il est possible d'exprimer des conditions à vérifier sur la reconnaissance : si l'expression précisée est fausse, alors l'instance de chronique est éliminée. D'autre part, un évènement peut être envoyé dans le flux à la suite d'une reconnaissance. Celui-ci peut ensuite être utilisé dans une autre chronique et ainsi définir des chroniques de niveau supérieur. Il y a également la possibilité d'exécuter du code C++ avant que le système n'engendre d'évènement, ou après.

CRS/Onera se compose d'un compilateur engendrant du code C++. L'utilisateur décrit des fichiers de systèmes de chroniques et le compilateur CRS/Onera engendre le code C++ correspondant avec un facteur d'expansion d'environ 50. Celui-ci est ensuite compilé. Chaque chronique est alors une classe dont les méthodes gèrent les évolutions et les duplications d'instances.

1.5.2 Définition d'une sémantique du langage des chroniques

Dans la lignée de CRS/Onera s'inscrivent les travaux de Bertrand *et al.* [Ber09] dont l'objectif est d'établir une sémantique du langage des chroniques de CRS/Onera.

Une sémantique ensembliste est donnée dans [Ber09] puis aboutie dans [CCK11] pour quatre opérateurs de base (la séquence, la conjonction, la disjonction et l'absence). Un ensemble de reconnaissances est défini pour chaque chronique, explicitant formellement ce que cela signifie de reconnaître dans un flux d'évènements donné le comportement décrit par une chronique. Cette sémantique est détaillée dans la sous-section suivante (1.5.3).

Une seconde sémantique opérationnelle est également développée. Une comparaison de différentes modélisations possibles du processus de reconnaissance de chroniques est réalisée dans [BCC07]. Les auteurs se concentrent sur deux principales difficultés : la multiplicité des reconnaissances et l'historisation des évènements donnant lieu aux reconnaissances. Les automates standards à états finis permettent la reconnaissance d'expressions régulières, mais une chronique, de par la multiplicité de la notion de reconnaissance, n'est pas une expression régulière. Un automate standard ne peut donc reconnaître qu'une seule fois une chronique ce qui ne répond pas à la problématique initiale. Si l'on introduit des automates à compteurs, les occurrences multiples d'une chronique peuvent alors être comptabilisées mais il n'est alors pas possible de distinguer les différentes reconnaissances comme il n'y a pas d'historisation des évènements. En revanche, les automates dupliqués, en créant une instance d'automate pour chaque reconnaissance partielle d'une chronique, permettent de reconnaître toutes les occurrences d'une chronique tout en préservant l'information de quels évènements ont donné lieu à chaque reconnaissance. Cependant, cette méthode ne permet pas d'avoir une approche modulaire dans le processus d'écriture de chroniques. Dans cette optique, les réseaux de Petri colorés qui permettent multiplicité et historisation, sont choisis car non seulement ils sont dotés de moyens de construction modulaire, mais encore des outils d'édition, de simulation et d'analyse sont disponibles pour mettre en avant les propriétés des réseaux.

Une sémantique en réseaux de Petri colorés est donc établie [BCC08, BCC09, Ber09, CCK11].

Un réseau est construit pour chaque chronique. Les transition du réseau sont tirées en fonction du flot d'évènement et les reconnaissances (éventuellement partielles) sont produites et/ou complétées en conséquence. La construction de ces réseaux se fait par induction sur la structure du langage à partir de briques de base, mais celle-ci n'est pas formalisée entièrement.

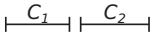
1.5.3 Détail de la sémantique ensembliste du langage des chroniques de CRS/Onera

Dans cette sous-section, nous détaillons précisément la sémantique du langage des chroniques telle que présentée dans [CCK11]. Nous présentons le langage des chroniques puis formalisons le concept d'évènement pour pouvoir ensuite définir la notion de reconnaissance d'une chronique.

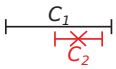
Une chronique décrit un certain agencement d'évènements. Le langage est construit à partir d'évènements simples et des opérateurs suivants, où C_1 et C_2 sont des chroniques :

— la *disjonction* $C_1 \,||\, C_2$ qui correspond à l'occurrence d'au moins l'une des deux chroniques C_1 et C_2.

— la *conjonction* $C_1 \& C_2$ qui correspond à l'occurrence des deux chroniques C_1 et C_2, dans un ordre quelconque, éventuellement entrelacées.



— la *séquence* $C_1\, C_2$ qui correspond à l'occurrence de la chronique C_1 suivie de l'occurrence de la chronique C_2.



— l'*absence* $(C_1) - [C_2]$ qui correspond à l'occurrence de la chronique C_1 sans occurrence de la chronique C_2 pendant l'occurrence de C_1.

Formellement, on définit le langage des chroniques à partir d'un ensemble donné de noms d'évènement comme suit :

Définition 1 (langage des chroniques). Soit $\mathfrak{N}$ un ensemble dénombrable dont les éléments sont des *noms* d'évènement simple.

On définit l'ensemble des *chroniques sur* $\mathfrak{N}$, noté $X(\mathfrak{N})$, par le schéma inductif suivant :

$$\frac{A \in \mathfrak{N}}{A \in X(\mathfrak{N})}\ (\text{nom})$$

$$\frac{C_1 \in X(\mathfrak{N}) \quad C_2 \in X(\mathfrak{N})}{C_1 \,||\, C_2 \in X(\mathfrak{N})}\ (\text{disjonction}) \qquad \frac{C_1 \in X(\mathfrak{N}) \quad C_2 \in X(\mathfrak{N})}{C_1 \& C_2 \in X(\mathfrak{N})}\ (\text{conjonction})$$

$$\frac{C_1 \in X(\mathfrak{N}) \quad C_2 \in X(\mathfrak{N})}{C_1\, C_2 \in X(\mathfrak{N})}\ (\text{séquence}) \qquad \frac{C_1 \in X(\mathfrak{N}) \quad C_2 \in X(\mathfrak{N})}{(C_1) - [C_2] \in X(\mathfrak{N})}\ (\text{absence})$$

Considérons deux exemples illustrant informellement l'expressivité du langage.

Exemple 1. Soit A, B et D des noms d'évènement simple de $\mathfrak{N}$. La chronique $(A \& B) \,||\, D$ correspond à deux évènements de noms respectifs A et B dans un ordre quelconque ou à un évènement de nom D.

Exemple 2. Soit A, B, D et E des noms d'évènement simple de $\mathfrak{N}$. La chronique $(A\,B) - [D \mid\mid E]$ correspond à un évènement de nom A suivi d'un évènement de nom B, sans occurrence ni d'un évènement de nom D ni d'un évènement de nom E entre les occurrences de A et B.

Évènements

Nous souhaitons définir la notion de reconnaissance de chronique afin de munir le langage d'une sémantique. Pour ce faire, il est nécessaire de formaliser au préalable le concept d'évènement et les notions qui lui sont associées.

Définition 2 (évènements). Soit $\mathfrak{N}$ un ensemble dénombrable de *noms* d'évènement. Soit $\mathfrak{E}$ un ensemble dénombrable dont les éléments sont des *évènements*.

Une *fonction de nommage* est une fonction totale $\nu : \mathfrak{E} \mapsto \mathfrak{N}$. Elle associe un nom à chaque évènement.

Le triplet $(\mathfrak{E}, \mathfrak{N}, \nu)$ est alors appelé *espace nommé d'évènements*.

Remarque 1. On distingue ainsi les noms des évènements (qui servent à construire les chroniques) que l'on notera par convention en majuscules $(A, B, C, D, \ldots)$ et les évènements (qui constituent les données observées à analyser) que l'on notera en minuscule $(a, b, c, d, \ldots)$. Pour faciliter la compréhension, on posera en général $\nu(a) = A$, $\nu(b) = B$, ...

Définition 3 (flux d'évènements). Soit $(\mathfrak{E}, \mathfrak{N}, \nu)$ un espace nommé d'évènements et soit $I \subseteq \mathbb{N}$.

Un *flux d'évènements* est une suite $\varphi = (u_i)_{i \in I}$ d'éléments de $\mathfrak{E}$.

On notera son *domaine* $\mathring{\varphi}$. On a ainsi $I = \mathring{\varphi}$. Il s'agit de l'ensemble des indices d'occurrence des évènements.

Par convention, si rien n'est spécifié, on commencera la numérotation à 1 (car cela correspondra à celle engendrée par le modèle de reconnaissance en réseaux de Petri colorés qui sera présenté Chap. 3 et 4).

Si $\varphi = (u_i)_{i \in I}$ est un flux d'évènements et si $J \subseteq I$, on définit la *restriction de φ à J*, notée $\varphi_{|J}$, par $\varphi_{|J} = (u_i)_{i \in J}$.

Pour un flux $\varphi = (u_i)_{i \in I}$, on définit les fonctions $\mathcal{E}_\varphi(i) = u_i$ et $\mathcal{N}_\varphi(i) = \nu(\mathcal{E}_\varphi(i)) = \nu(u_i)$.

$\mathcal{E}_\varphi(i)$ correspond au i^{e} évènement du flux φ.

$\mathcal{N}_\varphi(i)$ correspond au nom du i^{e} évènement du flux φ.

Reconnaissance d'une chronique

Il s'agit maintenant de s'appuyer sur les définitions précédentes pour doter le langage d'une sémantique ensembliste définissant la notion de reconnaissance de comportements. Une reconnaissance d'une chronique est représentée par l'ensemble exact des indices des évènements ayant donné lieu à la reconnaissance. Il est donc nécessaire de commencer par définir les notions suivantes liées aux indices.

Définition 4 (instances). Soit $(\mathfrak{E}, \mathfrak{N}, \nu)$ un espace nommé d'évènements sur lequel est défini un flux d'évènements $\varphi = (u_i)_{i \in I}$.

Une *instance* de φ est un sous-ensemble fini de I.

Un *support de* φ est un sous-intervalle fini de I. (Ainsi, tout support de φ est une instance de φ.)

Le *support d'une instance* r de φ est l'ensemble $[r] = \{i \in I : \min r \leq i \leq \max r\}$.

On notera $]r] = [r] \setminus \{\min r\}$ et $[r[= [r] \setminus \{\max r\}$.

On dit que deux instances r et r' de φ sont *compatibles*, noté $r \bowtie r'$, si $\max r < \min r'$ (c'est-à-dire si r « précède » r').

On définit la relation ternaire « r *est la réunion compatible de* r_1 *et* r_2 » par $r = r_1 \overset{\bowtie}{\cup} r_2$ si, et seulement si, $r_1 \bowtie r_2 \wedge r = r_1 \cup r_2$.

Remarque 2. $\bowtie$ est une relation d'ordre strict sur les instances de I.

Pour chaque chronique, en fonction du flux φ étudié, on définit par induction l'ensemble des reconnaissances associées.

Définition 5 (ensembles de reconnaissances). Soit $(\mathfrak{E}, \mathfrak{N}, \nu)$ un espace nommé d'évènements sur lequel est défini un flux d'évènements $\varphi = (u_i)_{i \in I}$.

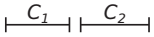
Soit $C \in X(\mathfrak{N})$.

On définit par induction l'*ensemble des reconnaissances de* C *sur le flux* φ, noté $R_C(\varphi)$:

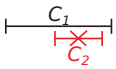
— si $C = A \in \mathfrak{N}$, alors $R_A(\varphi) = \{\{i\} : i \in \overset{\circ}{\varphi} \wedge \mathcal{N}_\varphi(i) = A\}$.
La chronique A est reconnue lorsqu'un évènement de nom A a lieu.

— si $C = C_1 \,||\, C_2$, alors $R_C(\varphi) = R_{C_1}(\varphi) \cup R_{C_2}(\varphi)$.
La chronique $C_1 \,||\, C_2$ est reconnue si la chronique C_1 est reconnue ou si la chronique C_2 est reconnue.

— si $C = C_1 \& C_2$, alors $R_C(\varphi) = \{r_1 \cup r_2 : r_1 \in R_{C_1}(\varphi) \wedge r_2 \in R_{C_2}(\varphi)\}$.
La chronique $C_1 \& C_2$ est reconnue si la chronique C_1 est reconnue et si la chronique C_2 est également reconnue, sans autre contrainte.



— si $C = C_1\, C_2$, alors $R_C(\varphi) = \{r_1 \cup r_2 : r_1 \in R_{C_1}(\varphi) \wedge r_2 \in R_{C_2}(\varphi) \wedge r_1 \bowtie r_2\}$.
La chronique $C_1\, C_2$ est reconnue si la chronique C_1 est reconnue, si la chronique C_2 est reconnue et si la reconnaissance de C_1 précède le début de la reconnaissance de C_2.



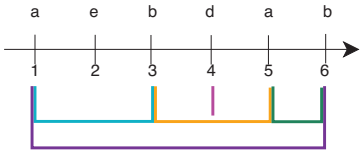
— si $C = (C_1) - [C_2]$, alors $R_C(\varphi) = \{r_1 : r_1 \in R_{C_1}(\varphi) \wedge (\mathfrak{P}_\mathfrak{f}([r_1[) \cap R_{C_2}(\varphi) = \emptyset)\}$ où, pour tout ensemble s, $\mathfrak{P}_\mathfrak{f}(s)$ est l'ensemble des parties finies de s.
La chronique $C = (C_1) - [C_2]$ est reconnue si la chronique C_1 est reconnue et s'il n'y a pas eu de reconnaissance de la chronique C_2 pendant la reconnaissance de la chronique C_1.

Ainsi, pour une chronique C et un flux φ, chaque reconnaissance de C dans φ correspond à un ensemble d'indices (les indices des évènements donnant lieu à la reconnaissance), et $R_C(\varphi)$ est l'ensemble de tous ces ensembles.

Exemple 3. Soit a, b, d et e des évènements de $\mathfrak{E}$ tels que $\nu(a) = A, \nu(b) = B, \nu(d) = D$ et $\nu(e) = E$.

Considérons la chronique $C = (A\&B) \,||\, D$ et le flux $\varphi = (a, e, b, d, a, b)$ avec $\overset{\circ}{\varphi} = [\![1, 6]\!]$.

On a alors $R_C(\varphi) = \{\{4\}, \{1, 3\}, \{1, 6\}, \{3, 5\}, \{5, 6\}\}$.



Exemple 4. Soit a, b, d, e, f et g des évènements de $\mathfrak{E}$ tels que $\nu(a) = A, \nu(b) = B, \nu(d) = D, \nu(e) = E, \nu(f) = F$ et $\nu(g) = G$.

Considérons la chronique $C = (A\,B) - [F]$ et le flux $\varphi = (d, a, e, a, b, g, a, f, b)$ avec $\overset{\circ}{\varphi} = [\![1, 9]\!]$.

On a alors $R_C(\varphi) = \{\{2, 5\}, \{4, 5\}\}$. Notons que $\{2, 9\}$, $\{4, 9\}$ et $\{7, 9\}$, bien qu'étant des reconnaissances de $A\,B$, ne sont pas des reconnaissances de C car $\mathcal{N}_\varphi(8) = F$.

Exemple 5. Soit a, b, d et e des évènements de $\mathfrak{E}$ tels que $\nu(a) = A, \nu(b) = B, \nu(d) = D$ et $\nu(e) = E$.

Considérons la chronique $C = (A\,B) - [D\,E]$ et le flux $\varphi = (a, d, a, e, b)$ avec $\overset{\circ}{\varphi} = [\![1, 5]\!]$.

On a alors $R_C(\varphi) = \{\{3, 5\}\}$. $\{1, 5\}$ n'est pas une reconnaissance de C car $\{2, 4\} \in R_{D\,E}(\varphi)$ et $\{2, 4\} \subset [\![1, 5]\!]$.

Exemple 6. Soit a, b, d, e, f et g des évènements de $\mathfrak{E}$ tels que $\nu(a) = A, \nu(b) = B, \nu(d) = D, \nu(e) = E, \nu(f) = F$ et $\nu(g) = G$.

Considérons la chronique $C = (A\,B) - [(D\,E) - [F\,G]]$ et le flux $\varphi = (a, d, f, g, e, b)$ avec $\overset{\circ}{\varphi} = [\![1, 6]\!]$.

On a alors $R_C(\varphi) = \{\{1, 6\}\}$ car $R_{(D\,E)-[F\,G]}(\varphi) = \{\}$.

Remarque 3. On peut montrer par induction directe sur les chroniques que, pour tout $C \in X(\mathfrak{N})$, $R_C(\{\}) = \{\}$.

1.6 D'autres modes de représentation et de reconnaissance de comportements

Dans les quatre sections précédentes, nous avons détaillé différentes approches du traitement d'évènements complexes. Dans cette section, nous présentons plus succinctement une dernière sélection de systèmes de reconnaissance de comportements moins proches de notre problématique. Nous renvoyons à [CM12] pour une collection plus complète de systèmes d'IFP. Dans [CM12], les problématiques et différentes options envisageables pour l'IFP sont détaillées, puis un grand nombre de systèmes sont présentés, répartis en quatre groupes :

— le domaine des bases de données actives ;
— les systèmes de gestion de flux de données ;
— les systèmes de CEP ;
— les systèmes disponibles dans le commerce.

Nous renvoyons également à d'autres surveys pour une vision plus complète [dCRN13, FTR$^+$10, ASPP12]. [FTR$^+$10] présente le CEP conjointement avec le domaine de l'analyse prédictive à travers une selection d'articles, d'outils commerciaux puis d'outils académiques ou en libre accès. [ASPP12] compare les chroniques de Dousson (Section 1.4), l'EC (Section 1.2) et la logique de Markov qui permet la prise en compte d'incertitudes dans le domaine du CEP [BTF07, dSBAR08, HN03, TD08, KDRR06]. Les trois approches sont comparées sur trois axes : la description des comportements à reconnaître, le raisonnement à réaliser pour effectuer la reconnaissance, et les méthodes d'apprentissage existant pour mettre en œuvre l'écriture des comportements.

MUSE [KM87] Kumar et Mukerjee établissent un système de reconnaissance incrémental appelé MUSE. Le modèle de temps adopté est linéaire et discret, avec une résolution suffisamment fine. Un évènement est un couple (φ, τ) où φ correspond à un nom d'évènement et τ est un ensemble d'instants consécutifs, décrivant ainsi un certain intervalle de temps. Des assertions temporelles peuvent être utilisées : les 13 assertions de Allen ainsi que quatre autres relations permettant d'exprimer des relations entre des évènements encore « incomplets » (*i.e.* dont l'ensemble d'instants consécutifs où l'évènement est vérifié n'est pas encore complet). Ceci permet d'assurer que, pour chaque couple d'évènements donné, une unique des ces dix-sept assertions est toujours vérifiée. Le processus de reconnaissance peut ainsi être effectué au fur et à mesure, à l'aide d'automates à états finis qui résument les différentes règles. Des conjonctions et disjonctions d'assertions peuvent ensuite être spécifiées. Pour finir, une sémantique temporelle sur les évènements permet de définir l'ensemble d'instants α d'une reconnaissance et ainsi de s'adapter à différents cas : on peut avoir tout simplement $\alpha = \tau$, mais on peut aussi définir $\alpha = \min(\tau)$ (sémantique instantanée) ou d'autres sémantiques plus complexes.

SAMOS [GD94b, GD94a] Swiss Active Mechanism based Object-oriented database Systems (SAMOS) est un système de gestion de base de données actives qui offre notamment un langage de description de comportements qui permet de spécifier des évènements complexes à intégrer aux règles de gestion. Les évènements considérés sont ponctuels (pour les évènements complexes, une algèbre d'évènements permet de définir leur instant d'occurrence) et dotés de paramètres. Des opérateurs – disjonction, conjonction, séquence, n occurrences dans un intervalle donné, absence d'occurrence dans un intervalle donné, première occurrence uniquement sur un intervalle donné – permettent de composer les évènements. Le mot clé *same* peut être apposé à un évènement complexe pour préciser des contraintes d'égalité sur les paramètres des évènements mis en jeu. Le système est également doté d'un intervalle de suivi des évènements indiquant une fenêtre dans laquelle reconnaître un comportement donné (reconnaître E dans I). Cet intervalle peut aussi bien être délimité explicitement avec des instants absolus qu'implicitement, et il peut être défini pour réapparaître périodiquement. Pour certaines constructions comme l'absence d'un comportement, il est obligatoire de préciser un tel intervalle. [GD94a] propose un modèle de reconnaissance des évènements complexes ainsi définis à l'aide d'un formalisme proche des réseaux de Petri colorés, car celui-ci permet de faire circuler dans le réseau les informations relatives aux paramètres des évènements complexes ou non. La notion de SAMOS Petri nets (S-PN) est introduite. Il s'agit de réseaux de Petri colorés possédant trois types de places, les places en entrée correspon-

dant aux évènements composant un évènement complexe, les places en sortie correspondant aux évènements complexes, et des places auxiliaires. Les opérateurs sont représentés par des transitions, et le mot clé *same* par une garde éventuelle sur les transitions. Une contrainte « d'absence de conflit » est imposée, c'est-à-dire qu'un jeton ne peut pas activer deux places simultanément. Seules les constructions relatives à la conjonction et à la reconnaissance de n occurrences sur un intervalle sont présentées. Lorsqu'un évènement complexe fait partie de la composition d'un autre évènement complexe, les réseaux correspondants sont combinés. Inversement, lorsqu'un évènement simple participe à plus d'un événement complexe, le jeton correspondant à l'évènement simple est dupliqué ; ainsi, à l'occurrence d'un évènement, seule une place doit être marquée. Ce modèle de reconnaissance est implémenté au sein de SAMOS.

GEM [MSS97, MSS96] Le langage Generalised Event Monitoring (GEM) est un langage déclaratif fondé sur un système de règles et s'attachant à la reconnaissance de comportements dans le cadre de systèmes distribués. La seule hypothèse réalisée est l'existence d'une horloge globale bien synchronisée. Les évènements considérés sont dotés d'attributs, dont par défaut, la classe d'évènement auquel l'évènement appartient, le composant du système dont il est issu et son instant d'occurrence. Des évènements complexes peuvent être construits à l'aide d'opérateurs de conjonction, de délai suivant une reconnaissance, d'absence d'un évènement complexe entre deux autres évènements, de disjonction, et de séquence. Une garde optionnelle peut exprimer des contraintes sur les attributs et sur les instants de début et de fin des reconnaissances, ce qui permet notamment de décrire l'ensemble des relations d'Allen, et un opérateur d'identification permet de se référer à une instance précise d'un évènement dans une règle. Les règles sont construites en quatre parties :

- un identifiant unique de la règle ;
- une fenêtre de détection déterminant la durée pendant laquelle doit être conservé l'historique des évènements liés à la règle ;
- la description de l'évènement complexe à reconnaître ;
- les actions à effectuer si l'évènement complexe est détecté (actions de notification explicite de l'évènement – interne ou externe à la règle –, transfert de certains évènements ayant donné lieu à la reconnaissance – dans une visée de filtrage par exemple –, activation ou désactivation de règles.

Des commandes de contrôle globales similaires aux actions ci-dessus sont également disponibles. Le processus de détection des comportements est fondé sur une structure arborescente associée à chaque comportement à reconnaître et suivant la structure de l'expression du comportement. Chaque nœud possède un type identifiant l'opérateur dont il s'agit, la garde associée, et l'historique des évènements correspondants. À l'arrivée d'un évènement, celui-ci est inséré à sa place dans l'arbre, sans considération temporelle autre, ce qui permet d'autoriser un retard dans l'arrivée des évènements, dans la limite de la fenêtre temporelle de détection définie. La gestion du retard des évènements se fait donc au sein même de l'étape de détection. Au niveau de chaque nœud, l'historique des évènements concernés est conservé, dans le cadre de la fenêtre de détection, ce qui permet de diminuer les évènements à ordonner. Des pointeurs sont également utilisés pour éviter la duplication d'historiques. [MSS96] décrit l'intégration et l'implémentation de GEM en C++.

Rete [For82, Ber02, WBG08, WGB08] Rete [For82] est un algorithme permettant de comparer une grande quantité de motifs (*patterns*) à une grande quantité d'objets. Il s'agit de déterminer l'intégralité des correspondances correctes et l'accent est porté en particulier sur l'efficacité du système. La version originale de Rete ne permet pas de considérations temporelles, et il existe de nombreuses extensions de ce système. [Ber02] introduit dans Rete une sémantique temporelle où tout évènement simple ou complexe est considéré comme ponctuel. Les règles définissant les comportements à reconnaître sont compilées pour obtenir des graphes composés de trois types de nœuds : les nœuds de classe qui filtrent les faits selon les classes, les nœuds de jonction qui combinent les faits, et les nœuds de discrimination qui filtrent les faits selon leurs attributs. Les faits sont alors stockés au niveau de chaque nœud et sont propagés en fonction des règles transcrites. Pour la gestion des contraintes temporelles, une horloge interne discrète est introduite conjointement avec une notion d'évènement qui s'oppose à la notion de fait. Les évènements sont datés et des contraintes temporelles peuvent être spécifiées sur les dates d'occurrence (on rappelle que chaque évènement même complexe est considéré comme ponctuel) avec les prédicats *before* et *after* bornant à un intervalle la différence entre les deux dates d'occurrence des évènements. Les contraintes peuvent être combinées avec des opérateurs de disjonction, de conjonction et de négation. L'accent est mis sur l'importance de la notion de changement d'état qui peut être exprimée à l'aide des évènements introduits. L'introduction de contraintes temporelles permet une gestion de la mémoire à l'intérieur même du système : les évènements rendus obsolètes sont oubliés. Une autre extension de Rete est développée dans [WBG08]. Elle s'oppose à [Ber02] dans la considération des évènements qui ne sont plus ponctuels mais dotés d'un instant de début et d'un instant de fin, ce qui est fondamental pour éviter des erreurs de reconnaissance dans le cas de séquences (problématique mise en avant dans [GA02]). Rete est donc étendu avec une sémantique temporelle d'intervalle qui permet l'expression des treize relations d'Allen étendues de deux manières :

— possibilité de définir la valeur exacte ou une plage de valeurs sur la durée séparant l'instant de début et l'instant de fin de deux intervalles de temps dans le cas des opérateurs non restrictifs (during, finishes, starts, overlaps, before) ;
— définition possible de limites de tolérance au niveau des opérateurs restrictifs (par exemple pour equals).

Il faut noter que ces extensions suppriment le caractère exclusif des opérateurs d'Allen mais permettent de s'adapter aux situations réelles que l'on peut souhaiter exprimer. Un système de calcul de la durée de vie des évènements est implémenté, découlant des contraintes temporelles spécifiées. [WGB08] présente une extension de Rete complémentaire à [WBG08] pour la gestion de fenêtres temporelles de validité autour des évènements. Ces fenêtres temporelles rendent obsolètes les évènements dont l'instant de fin sort de la fenêtre – notons qu'un évènement peut donc commencer avant la fenêtre dans laquelle il est considéré.

Snoop [CM94, CKAK94], SnoopIB [AC06] Le domaine des bases de données actives se consacre notamment à surveiller la séquence d'évènements affectant la base de donnée depuis l'extérieur. Le système reconnaît des comportements et peut y réagir suivant des règles Event-Condition-Action (ECA) spécifiant les comportements à reconnaître et les actions à effectuer

lors de la détection. Snoop [CM94] est un tel système. Il est construit sur un modèle de temps linéaire discret et permet l'analyse d'évènements primitifs ne pouvant avoir d'occurrences simultanées et composés à la fois :

— d'évènements explicites externes fournis au système avec leurs paramètres contenant au moins le type de l'évènement ainsi que sa date ;
— d'évènements temporels pouvant être absolus (c'est-à-dire datés) ou relatifs (*i.e.* placés temporellement par rapport à un autre évènement de Snoop) ;
— d'évènements de la base de donnée correspondant aux opérations de manipulation des données.

Les évènements composites à reconnaître sont définis à partir de ces évènements primitifs et des opérateurs de disjonction, de séquence, de conjonction, n occurrences parmi ..., l'opérateur apériodique $\mathtt{A}(\mathtt{E}_1, \mathtt{E}_2, \mathtt{E}_3)$ ($\mathtt{E}_2$ a lieu entre $\mathtt{E}_1$ et $\mathtt{E}_3$) et l'opérateur périodique $\mathtt{P}(\mathtt{E}_1, [\mathtt{t}], \mathtt{E}_3)$ ($\mathtt{E}_1$ puis $\mathtt{E}_3$ dure exactement $[\mathtt{t}]$. Notons qu'il n'y a pas alors d'expression de négation ou d'absence (la difficulté est évoquée dans la Section 5.4 de [CM94]). Les évènements sont tous considérés comme ponctuels et la notion de modificateur d'évènement est introduite pour définir l'instant d'occurrence d'un évènement initialement non ponctuel. Par défaut, il existe deux modificateurs d'évènements, à savoir *begin_ of* et *end_ of*. Le choix de multiplicité des reconnaissances dans le processus de reconnaissance de ces comportements complexes varie selon le contexte adopté parmi le contexte récent, le contexte chronique, le contexte continu et le contexte cumulatif présentés dans la Section 1.1. Le processus de reconnaissance est fondé sur des graphes associés à chaque évènement complexe à reconnaître et obtenus après compilation des expressions de ces évènements. Les arbres suivent la structure des expressions concernées, et, dans le cas de sous-expressions identiques, les parties associées de l'arbre sont amalgamées. Dans [CM94], un algorithme est détaillé pour le contexte récent qui permet l'utilisation d'un buffer de taille fixe au niveau de chaque nœud des arbres (contrairement aux contextes continu et cumulatif qui demandent beaucoup d'espace mémoire). Une notion d'équivalence d'expressions est définie et peut permettre de réécrire une description de comportement sous une autre forme pour optimiser le processus de reconnaissance par exemple en dévoilant des nouvelles sous-expressions communes. [CKAK94] présente la sémantique de Snoop à l'aide de formules logiques du premier ordre. Cependant, comme pour Rete, la considération d'évènements uniquement ponctuels pose problème dans le cas de composition avec une séquence par exemple car il faut pouvoir également comparer les instants d'initiation (et pas uniquement ceux de terminaison) des reconnaissances (problématique mise en avant dans [GA02]). Pour répondre à ce problème, [AC06] propose SnoopIB, une nouvelle sémantique fondée sur des intervalles et dont la définition formelle est en partie présentée dans [GA02]. Deux nouveaux opérateurs sont introduits, à savoir l'opérateur de non occurrence d'un évènement entre l'instant de terminaison d'un évènement et l'instant d'initiation d'un autre évènement, et l'opérateur PLUS exprimant l'occurrence d'un évènement suivi d'une durée.

Domaines d'application

Au travers de cette sélection de systèmes d'analyse d'évènements complexes, nous avons un aperçu général des différentes approches possibles, dans des domaines variés. La multiplicité de ces systèmes est due au large éventail d'applications possibles de la reconnaissance de comportements. Nous donnons ici un échantillon des différents domaines dans lesquels l'IFP s'est montrée utile :

- supervision et l'analyse de situations dangereuses à l'aide d'un drone pour aider les services de police [Hei01, DKH09a, DKH09b, DGK+00], avec des chroniques ;
- de nombreux domaines d'application en médecine comme le monitoring cardiaque où des méthodes d'apprentissage sont appliquées[CD00, CCQW03, QCC+10, Doj96, Por05, CD97], avec des chroniques ;
- gestion d'alarmes pour la détection d'intrusions informatique [MD03], avec CRS et les chroniques de Dousson *et al.* ;
- diagnostic de web-services [PS09, CGR+07, LGCR+08], avec des chroniques ;
- évaluation de la qualité de transports publics (projet PRONTO) [KVNA11, VL10], avec l'EC ;
- surveillance vidéo (projet CAVIAR) [SA11, ASP10b, ASP10a, AP09], avec l'EC ;
- analyse des médias sociaux [SA11] ;
- aide à la prise de décision dans le cadre de combats aériens [CV98], avec des chroniques ;
- supervision et gestion de réseaux [SEC+10], avec des chroniques ;
- dans l'industrie, supervision d'une turbine à gaz dans une usine pétrochimique [MNG+94] et supervision d'une usine de lait [MCCDB10], avec des chroniques ;
- caractérisation d'activité humaine [CMM12], avec des chroniques.

Après cette introduction et ce survol des systèmes existants, nous allons développer le système de reconnaissance de comportements des Chroniques/Onera afin de se rapprocher d'un système répondant aux enjeux évoqués dans la Section 1.1.

Chapitre 2

Construction d'un cadre théorique pour la reconnaissance de comportements : le langage des chroniques

Sommaire

De nombreuses applications concrètes de la reconnaissance de comportements nécessitent notamment des moyens à la fois de validation et de traçabilité, comme il sera illustré dans le Chapitre 5. Pour cela, il s'agit de fournir un cadre théorique solide pour la reconnaissance de comportements en adoptant une approche purement formelle qui assure une possibilité de vérification et d'analyse du processus de reconnaissance. Dans ce chapitre, nous posons ce cadre théorique en développant un langage de description de comportements, le langage des chroniques introduit dans la Section 1.5 [1], et en formalisant le processus de reconnaissance associé à l'aide d'une sémantique [PBC$^+$] :

— nous développons un formalisme autour de la notion d'évènement et de leurs attributs ;
— nous étendons largement la syntaxe du langage des chroniques de [CCK11] avec des constructions permettant non seulement l'expression de contraintes temporelles variées mais aussi la spécification de contraintes complexes sur des attributs d'évènement ;
— nous introduisons une nouvelle représentation de la notion de reconnaissance de chronique, en passant d'ensembles d'ensembles à des ensembles d'arbres ce qui donne une structure des reconnaissances plus précise et permet ainsi des opérations plus fines sur les reconnaissances ;
— nous formalisons le processus de reconnaissance à travers une sémantique du langage que nous avons étendu ;
— nous rendons possible l'implémentation du processus de reconnaissance avec un modèle de temps continu grâce à une fonction « Look-ahead » qui fournit le prochain instant où interroger le programme.

Dans une première section (2.1), nous posons les définitions générales formalisant le contexte de notre travail d'analyse de comportements complexes, à savoir les notions d'évènement et d'attributs d'évènement. Nous définissons ensuite dans la Section 2.2 la syntaxe étendue du langage des chroniques, permettant notamment la spécification à la fois de contraintes sur des attributs d'évènement et de contraintes temporelles exprimant des exigences sur les délais. La Section 2.3 définit ensuite la sémantique du langage des chroniques, spécifiant ainsi la notion de reconnaissance, et ce après avoir défini une nouvelle représentation arborescente des reconnaissances. Pour nous familiariser avec le langage des chroniques et pour simplifier les démonstrations à venir, nous étudions dans la Section 2.4 les principales propriétés du langage. Dans la visée d'une implémentation du processus de reconnaissance et pour assurer la gestion d'un modèle de temps continu, nous définissons dans la Section 2.5 une fonction dite de « Look-ahead » qui permet par la suite le pilotage des appels au processus de reconnaissance. Le chapitre s'achève avec un tableau récapitulatif informel des propriétés du langage construit dans la Section 2.6.

1. Rappelons que les chroniques étudiées ici se réfèrent à celles introduites par P. Carle dans [CBDO98] qui diffèrent de celles introduites par C. Dousson dans [DGG93].

2.1 Définitions générales préalables

L'objectif de cette section est de formaliser la notion d'évènement qui sera manipulée tout au long du chapitre. Nous reprenons le cadre formel posé dans la Section 1.5 [CCK11] en introduisant la notion d'attribut dans la formalisation du concept d'évènement. En effet, nous souhaitons maintenant considérer des évènements munis d'informations (par exemple un identifiant, une qualité, des coordonnées, une vitesse, ...) sur lesquelles il sera ensuite possible de raisonner en effectuant des comparaisons ou des calculs, et de poser des contraintes. Cette extension primordiale est motivée par de nombreuses applications qui nécessitent de pouvoir raisonner sur des données liées aux évènements du flux que l'on souhaite analyser. Par exemple, supposons que nous surveillons un avion en vol dans le but de s'assurer que la fréquence radio sur laquelle il est réglé correspond bien à celle associée à sa zone de vol. Il faut identifier les évènements relatifs à l'avion au milieu de tous les autres évènements, puis accéder aux données relatives à la fréquence radio et à la position de l'appareil afin d'effectuer des comparaisons entre elles. Dans une autre situation, on peut également être amené à effectuer des calculs, pour évaluer la distance entre deux avions et garantir une distance minimale.

Nous allons donc introduire la notion d'attribut d'évènement. Les évènements observés pourront être dotés d'une ou plusieurs caractéristiques, que nous appellerons *attributs* ou *propriétés*. Nous cherchons à reconnaître des agencements complexes de tels évènements, agencements décrits par des formules de chroniques que nous définirons par la suite. Dans ces chroniques, nous souhaitons exprimer des contraintes sur ces attributs d'évènement. Pour manipuler librement ces propriétés, nous construisons, à partir des attributs d'évènement, des attributs de reconnaissance de chronique qui auront un rôle similaire, à savoir représenter des informations liées au comportement plus global décrit par la chronique. Ainsi, si l'on souhaite écrire des chroniques pour réaliser de l'évitement de collision à partir de mesures radar, il peut être intéressant d'écrire une première chronique calculant la distance entre deux aéronefs à partir des données brutes du radar. Cette chronique correspond alors à la reconnaissance d'une distance avec comme propriétés les identifiants des deux appareils ainsi que la donnée numérique de la distance calculée. La chronique peut ensuite être utilisée au sein d'autres chroniques pour engendrer des alertes par exemple. La chronique, munie de ses nouveaux attributs, peut alors être considérée comme un évènement complexe de plus haut niveau, souvent non ponctuel, et formant une abstraction des évènements du flux. La chronique obtenue peut alors être utilisée pour former une autre chronique, au même titre qu'un simple évènement, et l'on peut disposer de ses attributs.

Pour définir ces notions d'évènement et d'attribut, on considère les trois ensembles suivants :

— $\mathfrak{N}$ un ensemble dénombrable de *noms d'évènement*, contenant un élément τ utilisé pour nommer les instants temporels purs ;
— $\mathfrak{P}$ un ensemble dénombrable de *noms de propriété* ou aussi de *noms d'attribut* contenant un élément particulier $\diamondsuit$ dénommant les propriétés anonymes qui désignent les propriétés qui n'ont pas encore été nommées par l'utilisateur ;
— $\mathcal{V}$ un ensemble de *valeurs* de propriété.

2.1.1 Évènements et leurs attributs

Commençons par définir la notion d'évènement. Contrairement à [CCK11], on souhaite considérer un modèle de temps continu. Les évènements sont donc datés par des réels et nous les identifions maintenant par un couple $(nom, date)$ et non plus par leur indice d'occurrence dans le flux d'évènements étudié.

Définition 6 (évènements). Un *évènement* est une paire $(e,t) \in \mathfrak{N} \times \mathbb{R}$ composée d'un nom d'évènement et d'une date représentée par un réel. Le nom d'évènement $\tau \in \mathfrak{N}$ est réservé pour identifier les évènements temporels purs. Sur l'ensemble des évènements $\mathfrak{E} \subseteq \mathfrak{N} \times \mathbb{R}$, la projection sur la date est la *fonction de datation*, notée $\theta : \mathfrak{E} \to \mathbb{R}$.

Les informations spécifiques associées aux évènements sous la forme d'attributs sont regroupées dans un ensemble d'attributs comme suit :

Définition 7 (attributs, ensembles d'attributs). Un *attribut*, aussi appelé une *propriété*, est une paire $a = (p, v) \in \mathfrak{P} \times \mathcal{V}$ composée d'un nom de propriété et d'une valeur. Sur l'ensemble des attributs, la projection sur le nom de propriété est appelée la *fonction de référence*, notée ρ.

Un *ensemble d'attributs d'évènement* est un ensemble $\mathrm{X} \subseteq \mathfrak{P} \times \mathcal{V}$ vérifiant la *propriété fonctionnelle* suivante, qui exprime que X est le graphe d'une fonction, c'est-à-dire que chaque propriété n'a qu'une seule valeur :

$$\forall p \forall v((p,v) \in \mathrm{X} \Rightarrow \forall w((p,w) \in \mathrm{X} \Rightarrow w = v)) \tag{2.1}$$

Par la suite on considère un *ensemble* $\mathfrak{A}_e(\mathfrak{P}, \mathcal{V})$ *d'ensembles d'attributs d'évènement* sur $\mathfrak{P} \times \mathcal{V}$ stable par union d'ensembles d'attributs d'évènement ayant des noms disjoints, c'est-à-dire vérifiant la contrainte suivante :

$$\forall X_1 \in \mathfrak{A}_e(\mathfrak{P}, \mathcal{V}) \quad \forall X_2 \in \mathfrak{A}_e(\mathfrak{P}, \mathcal{V})$$
$$\{p \in \mathfrak{P} : \exists v \in \mathcal{V}(p,v) \in X_1\} \cap \{p \in \mathfrak{P} : \exists v \in \mathcal{V}(p,v) \in X_2\} = \emptyset \Rightarrow X_1 \cup X_2 \in \mathfrak{A}_e(\mathfrak{P}, \mathcal{V})$$

Cette contrainte de stabilité est nécessaire pour la bonne définition des ensembles de reconnaissance donnée dans la Définition 16 (en effet, $\mathfrak{A}_e(\mathfrak{P}, \mathcal{V})$ est le domaine de définition de la fonction $\mathcal{D}$ de la Définition 11).

Ces évènements, dotés éventuellement de leurs attributs, sont regroupés sous la forme de flux d'évènements que l'on souhaite étudier et analyser avec notre système de reconnaissance de comportements.

Définition 8 (flux d'évènements). Un *flux d'évènements* est défini comme une suite d'évènements $\varphi = (u_i)_{i \in \mathbb{N}} \in \mathfrak{E}^{\mathbb{N}}$ ordonnée par rapport au temps :

$$\forall i \forall j (i < j \Rightarrow \theta(u_i) < \theta(u_j))$$

et dotée d'une *fonction d'extraction d'attributs* $\alpha : \{\varphi(i) : i \in \mathbb{N}\} \to \mathfrak{A}_e(\mathfrak{P}, \mathcal{V})$ qui fournit l'ensemble d'attributs associé à chaque évènement permettant ainsi l'accès aux valeurs des attributs d'évènement simple dans le flux d'évènements.

2.1.2 Opérations sur les attributs

Lors du processus de reconnaissance qui analyse le flux d'évènements pour reconnaître les comportements décrits par des chroniques, des évènements sont rassemblés pour former des reconnaissances, comme ce sera formalisé dans la Section 2.3. Durant ce processus de reconnaissance, des attributs doivent être manipulés et peuvent être modifiés. Il peut être nécessaire de réaliser des opérations sur divers attributs de différents évènements. Si ces opérations sont imbriquées dans un comportement plus complexe à reconnaître, les résultats de ces opérations doivent être stockés sous forme d'attributs associés cette fois-ci aux *reconnaissances* et non plus aux évènements, afin de pouvoir être utilisés a posteriori. Comme évoqué précédemment, les reconnaissances de comportements sont donc elles aussi dotées d'attributs.

Définition 9 (attribut de reconnaissance). Un *ensemble d'attributs de reconnaissance de comportements* est un ensemble $X \subseteq \mathfrak{P} \times \mathfrak{A}_e(\mathfrak{P}, \mathcal{V})$ vérifiant la propriété fonctionnelle (2.1). L'ensemble des ensembles d'attributs de reconnaissance est noté $\mathfrak{A}_r(\mathfrak{P}, \mathcal{V})$.

Comme défini au début de cette section, l'ensemble des noms de propriété contient un nom spécifique, $\diamondsuit$, utilisé comme nom anonyme. Lors de la progression du processus de reconnaissance, des attributs peuvent être calculés et enregistrés sous ce nom, en tant que nouveaux attributs temporaires, avant d'être éventuellement nommés pour être utilisés par la suite, comme il sera détaillé 2.3.2.

Les fonctions suivantes permettent l'expression de telles opérations sur les attributs.

Définition 10 (transformations d'attributs). Une *transformation d'attributs* est une fonction définie sur l'ensemble des ensembles d'attributs de reconnaissance $\mathfrak{A}_r(\mathfrak{P}, \mathcal{V})$ dans l'ensemble des ensembles d'attributs d'évènement $\mathfrak{A}_e(\mathfrak{P}, \mathcal{V})$ qui permet d'engendrer de nouvelles propriétés qui seront anonymes jusqu'à ce qu'elles soient oubliées ou nommées.

Une fonction de transformation d'attributs f doit vérifier la contrainte suivante :

$$\forall X_r \in \mathfrak{A}_r(\mathfrak{P}, \mathcal{V}) \quad \{p \in \mathfrak{P} : \exists v \in \mathcal{V}(p, v) \in f(X_r)\} \\ \cap \{p \in \mathfrak{P} : \exists X_e \in \mathfrak{A}_e(\mathfrak{P}, \mathcal{V}) \quad \exists p_r \in \mathfrak{P}((p_r, X_e) \in X_r \wedge \exists v \in \mathcal{V}(p, v) \in X_e)\} = \emptyset$$

qui exprime que les nouveaux attributs d'évènement créés par la fonction f ont des noms strictement différents de ceux déjà employés dans l'ensemble d'attributs de reconnaissance qui est en argument de f. Cette obligation participe à assurer l'unicité d'utilisation des noms de propriété dans une chronique.

L'*ensemble des fonctions de transformation d'attributs* sur $(\mathfrak{P}, \mathcal{V})$ est noté $\mathfrak{T}(\mathfrak{P}, \mathcal{V})$.

Les transformations d'attributs produisent ainsi de nouvelles données attachées aux reconnaissances. Pour pouvoir les employer par la suite dans des comparaisons ou des calculs, il est nécessaire de les identifier en les nommant. Les nouvelles propriétés qui sont soit issues d'une transformation d'attributs soit directement issues du flux d'évènements (*i.e.* des attributs d'évènement) ont d'abord un nom anonyme $\diamondsuit$ qui leur est donné par la fonction suivante :

Définition 11 (fonction de dénomination anonyme). Une *fonction de dénomination anonyme* est définie sur l'ensemble des ensembles d'attributs d'évènement. Elle crée un ensemble d'attributs de reconnaissance, réduit à un singleton, en nommant $\diamondsuit$ l'ensemble d'attributs d'évènement

initial comme défini ci-dessous :

$$\begin{array}{rccl} \mathcal{D}: & \mathfrak{A}_e(\mathfrak{P},\mathcal{V}) & \to & \mathfrak{A}_r(\mathfrak{P},\mathcal{V}) \\ & X & \mapsto & \begin{cases} \{(\diamondsuit, X)\} \text{ si } X \neq \emptyset \\ \emptyset \text{ si } X = \emptyset \end{cases} \end{array}$$

Les propriétés anonymes $\diamondsuit$ peuvent ensuite être nommées par la fonction suivante :

Définition 12 (fonction de renommage d'attributs). La *fonction de renommage d'attributs* est définie sur l'ensemble des ensembles d'attributs de reconnaissance et sur l'ensemble des noms de propriété. Elle nomme l'attribut anonyme $\diamondsuit$ de l'ensemble d'attributs comme défini ci-dessous [2] :

$$\begin{array}{rccl} \mathcal{R}: & \mathfrak{A}_r(\mathfrak{P},\mathcal{V}) \times \mathfrak{P} & \to & \mathfrak{A}_r(\mathfrak{P},\mathcal{V}) \\ & (X, p') & \mapsto & \{(p', v) : (\diamondsuit, v) \in X\} \\ & & & \cup \{(p, v) \in X : p \neq \diamondsuit\} \end{array}$$

Ces fonctions permettent d'effectuer toutes les opérations nécessaires à la manipulation d'attributs et sont utilisées dans la Section 2.3.2 pour définir le processus de reconnaissance où l'on manipule des attributs qui doivent être nommés et parfois modifiés.

2.2 Définition d'une syntaxe étendue du langage des chroniques : ajout de contraintes sur des attributs d'évènement et de constructions temporelles

Nous pouvons maintenant définir une syntaxe du langage des chroniques étendant largement celle de [CCK11]. De même que dans la Section 1.5, le langage des chroniques est construit par induction à partir d'évènements simples et de divers opérateurs permettant de spécifier des contraintes, temporelles ou non, sur les évènements étudiés. Commençons par détailler les différentes extensions et modifications envisagées.

Expression de contraintes sur des attributs d'évènement Le langage des chroniques peut maintenant être étendu pour permettre de prendre en compte et de raisonner sur les attributs définis dans la Section 2.1.1 à l'aide des fonctions définies dans la Section 2.1.2. Toute chronique est dotée d'un prédicat P qui exprime les contraintes souhaitées sur les propriétés manipulées. Avant de pouvoir valider une reconnaissance, il faut que le prédicat P, évalué sur les valeurs des attributs de la reconnaissance, soit vérifié. Pour la manipulation des attributs, une chronique est également dotée d'une fonction de transformation d'attributs (Définition 10) introduisant de nouveaux attributs.

2. Il est immédiat de montrer que les images de la fonction $\mathcal{R}$ sont bien des éléments de $\mathfrak{A}_r(\mathfrak{P},\mathcal{V})$.

Une construction de nommage Nous ajoutons également une construction de nommage afin de pouvoir spécifier un nom pour nommer les nouvelles propriétés anonymes $\diamondsuit$, comme introduit précédemment dans 2.1. Une telle construction est nécessaire. En effet, un nom unique doit être donné aux nouveaux attributs, ils ne peuvent donc pas être nommés d'après le nom de l'évènement ou de la sous-chronique auquel ils correspondent car plusieurs évènements de même nom ou même plusieurs sous-chroniques peuvent prendre part à une description de comportement, mais il est nécessaire de pouvoir les distinguer. Donc, comme plusieurs évènements d'un même nom peuvent prendre part à la construction d'une chronique, il est nécessaire de pouvoir se référer précisément à l'un d'entre eux afin de savoir de quel évènement il faut récupérer dans le flux les valeurs des attributs pour évaluer le prédicat. Le nom spécifié par la construction de nommage remplit ce rôle mais pour cela il faut assurer qu'un nom donné n'est employé qu'une unique fois.

Notion de contexte Pour assurer que les noms de propriété ne sont effectivement utilisés qu'une seule fois dans une chronique, il est nécessaire de faire apparaître des contraintes sur ces noms au niveau de la construction du langage. Pour ce faire, nous définissons la notion de contexte d'une chronique : il s'agit de l'ensemble des noms de propriété mis en jeu dans la chronique.

Le contexte d'une chronique donnée, construit par induction en parallèle du langage, est intuitivement généralement l'union des contextes des sous-chroniques directes formant la chronique étudiée (c'est-à-dire l'union des noms de propriété des sous-chroniques). Cette gestion du contexte doit être particularisée pour quelques constructions, et nous détaillons les raisons de ces particularités après la définition suivante.

Le contexte est également utilisé pour déterminer quels attributs sont disponibles pour l'évaluation du prédicat P et de la fonction de transformation f associés à la chronique. On devra ainsi distinguer deux types de contextes : un *contexte d'évaluation* pour le prédicat P et la fonction f, et un *contexte résultant* à transmettre dans l'induction pour la construction des contextes de chroniques plus complexes. Le contexte évolue donc en deux temps : une première fois avant la reconnaissance de la chronique et l'évaluation du prédicat, et une seconde fois après. Nous appelons *contexte d'évaluation* le contexte correspondant au domaine possible du prédicat et de la fonction, et *contexte résultant*, le contexte d'évaluation éventuellement modifié, après l'évaluation du prédicat, et qui sert à définir le prochain contexte d'évaluation dans l'induction. Des contraintes qui apparaissent dans la construction du langage pour assurer qu'un nom ne peut être utilisé qu'une seule fois portent sur le contexte d'évaluation. Les raisons précises derrière l'existence de ces deux notions de contexte seront développées après la Définition 13.

Des opérateurs de contraintes sur les délais Afin de modéliser des contraintes temporelles, dix constructions sont ajoutées par rapport à [CCK11]. Ces constructions découlent de la logique d'intervalles d'Allen [All83] évoquée dans le Chapitre 1. Il s'agit de spécifier des contraintes sur des intervalles de reconnaissance de chroniques, c'est-à-dire, pour une reconnaissance donnée, sur l'intervalle de temps nécessaire et suffisant pour établir cette reconnaissance. Une première partie des opérateurs de contraintes temporelles permet d'exprimer toutes les relations temporelles entre deux intervalles de reconnaissance de deux chroniques (opérateurs equals, starts, finishes, meets,

overlaps, et during) à l'instar des treize relations d'Allen [3]. D'autres constructions décrivent trois différentes contraintes sur la durée de reconnaissance d'une chronique (opérateurs lasts δ, at least δ, at most δ) exprimant que le temps de reconnaissance de la chronique doit être respectivement exactement, au moins ou au plus une certaine durée δ. Elles correspondent à une transcription des opérateurs d'Allen equals, during et le symétrique de during. Seules ces trois constructions sont retenues car ce sont les seules ayant un sens dans notre contexte. En effet, si l'on considère par exemple la transposition de la relation starts, la contrainte qu'une certaine durée δ « débute » une reconnaissance n'apporte aucune spécification supplémentaire que celle qui impose que la reconnaissance dure au moins δ. La relation overlaps, quant à elle, lorsqu'elle est transcrite entre une reconnaissance et une durée, serait vérifiée dans tous les cas ; de même que la relation before. Il n'y a donc pas de sens à les introduire. La dernière construction spécifie l'écoulement d'un laps de temps directement après la reconnaissance d'une chronique (opérateur then δ) et correspond à une transposition de l'opérateur d'Allen meets.

Les bornes de l'absence La notion d'absence est une notion cruciale dans le domaine de la reconnaissance de comportements, comme évoqué dans la Section 1.1. Dans la description d'une absence, il est nécessaire, pour pouvoir statuer d'une reconnaissance, de spécifier l'intervalle de temps pendant lequel le comportement non désiré ne doit pas se produire. Dans [CCK11], le langage des chroniques permet la description de l'absence d'un comportement pendant un autre comportement. C'est ce second comportement qui spécifie l'intervalle de temps à observer et pendant lequel le comportement interdit ne doit pas avoir lieu. Se pose alors la question de l'inclusion ou non des bornes dans l'intervalle considéré : le comportement interdit peut-il commencer en même temps ou se terminer au même instant que le comportement recherché ? Dans [CCK11], un seul cas de figure est formulable : le comportement interdit peut commencer en même temps mais doit terminer strictement avant la fin de la reconnaissance recherchée pour l'invalider. Nous introduisons dans la définition suivante (*cf.* [absence]) trois nouvelles notations qui permettent l'expression des trois possibilités. Notons que l'ancienne notation est utilisée parmi les trois, mais, afin de rendre la lecture du langage plus intuitive, elle ne désigne plus les même bornes que dans [CCK11] (*cf.* Définition 16).

Le « cut » Nous ajoutons également plusieurs nouveaux opérateurs. Les deux premiers ajouts correspondent à des séquences dont nous souhaitons limiter la multiplicité des reconnaissances. Nous commençons par ajouter un opérateur que l'on appelle « cut », noté « ! ». Il exprime la reconnaissance consécutive de deux comportements `A` et `B`, comme dans le cadre d'un séquence, mais nous la restreignons à la première occurrence du comportement `B` après chaque comportement `A`. Nous limitons donc le nombre de reconnaissances par rapport à une séquence classique.

3. L'opérateur d'Allen before correspond à une version stricte de la séquence (la séquence est en fait une disjonction de meets et before) qui fait déjà partie du langage de [CCK11] et n'est donc pas ajouté.
Notons également que l'opérateur de conjonction peut alors être exprimé comme une disjonction de tous les opérateurs d'Allen.

Le changement d'état ou « double cut » Nous ajoutons une seconde construction, correspondant à une séquence dont nous limitons la multiplicité des reconnaissances, avec une chronique exprimant le « changement d'état ». Il s'agit de décrire le passage d'un comportement caractérisant un état à un autre comportement caractérisant un autre état. Cette construction est représentée par l'opérateur noté « !! » et correspond, comme nous le verrons plus clairement par la suite, à un cut « dans les deux sens », donc nous l'appelons également « double cut ». Elle répond à une problématique fréquente lors du traitement de cas concrets. Considérons par exemple un drone à terre dont le décollage doit être détecté. Ses coordonnées de position permettent aisément d'identifier si le drone se trouve à terre (`on_ground`) ou bien s'il vole (`above_ground`). Le changement d'état de l'un à l'autre caractérise le décollage de l'appareil. Identifier ce changement d'état correspond à détecter « le dernier `on_ground` avant le premier `above_ground` », ce qui correspond à la sémantique de l'opérateur « !! ». Notons que ce changement d'état peut être exprimé à l'aide des autres opérateurs, mais lorsque l'on traite des applications réelles, il apparaît que l'opérateur de changement d'état est souvent nécessaire. Cette construction est donc ajoutée au langage afin de s'affranchir de définitions fastidieuses de chroniques.

Une construction d'évènement de reconnaissance Lors de l'écriture des descriptions des comportements à reconnaître, il peut être intéressant de décomposer les comportements complexes pour les décrire en plusieurs étapes. Il y a alors deux possibilités pour imbriquer une chronique dans une autre. Soit la chronique est considérée comme un évènement ayant un instant de fin et un instant de début a priori disjoints, soit la chronique est réduite à son instant de reconnaissance. La construction par induction du langage des chroniques implique l'intégration naturelle du premier cas dans la syntaxe. En revanche, si l'on souhaite pouvoir exprimer le second cas, il faut rajouter une construction à apposer à une chronique pour se référer uniquement à son instant de reconnaissance. Pour ce faire, nous introduisons un opérateur, appelé « at » et noté « @ », qui correspond à la détection d'un évènement « abstrait d'une chronique », c'est-à-dire réduit à son instant de reconnaissance et donc nécessairement ponctuel.

Donnons maintenant la syntaxe du langage muni des extensions décrites ci-dessus. Notons que la sémantique du langage est présentée Définition 16.

Définition 13 (chroniques). Soit $\mathfrak{N}$, $\mathfrak{P}$, et $\mathcal{V}$ les ensembles introduits au début de la Section 2.1, p. 43. Soit $\mathfrak{S}$ un ensemble de symboles de prédicats. L'ensemble des *chroniques sur* $(\mathfrak{N}, \mathfrak{P}, \mathcal{V}, \mathfrak{S})$, noté $\mathfrak{X}$, est un ensemble de triplets (C, P, f), où :

- C est une *formule de chronique*, comme définie dans la définition inductive de $\mathfrak{X}$ qui suit ;
- $P \in \mathfrak{S}$ est un symbole de prédicat ;
- $f \in \mathfrak{T}(\mathfrak{P}, \mathcal{V})$ est une transformation d'attributs.

$\mathfrak{X}$ est défini inductivement avec deux notions de *contextes*, qui sont des fonctions de $\mathfrak{X}$ dans $\mathfrak{P}$:

- un *contexte d'évaluation*, noté $\mathcal{C}_e(\cdot)$;
- un *contexte résultant*, noté $\mathcal{C}_r(\cdot)$.

Pour tous $C_1 \in \mathfrak{X}$ et $C_2 \in \mathfrak{X}$, on a :

[évènement simple] Si $A \in \mathfrak{N}$, alors $(\mathbf{A}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(A, P, f) = \{\Diamond\}$, et $\mathcal{C}_r(A, P, f) = \mathcal{C}_e(A, P, f)$;

[séquence] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1\ C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1\ C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1\ C_2, P, f) = \mathcal{C}_e(C_1\ C_2, P, f)$;

[conjonction] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1 \& C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \& C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, $\mathcal{C}_r(C_1 \& C_2, P, f) = \mathcal{C}_e(C_1 \& C_2, P, f)$;

[disjonction] $(\mathbf{C_1 \mid\mid C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \mid\mid C_2, P, f) = \mathcal{C}_r(C_1) \cap \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 \mid\mid C_2, P, f) = \mathcal{C}_e(C_1 \mid\mid C_2, P, f)$;

[absence] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{(C_1) - [C_2]}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e((C_1) - [C_2], P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r((C_1) - [C_2], P, f) = \mathcal{C}_r(C_1)$;
De même pour $\mathbf{(C_1) -]C_2]}$, $\mathbf{(C_1) - [C_2[}$ [4] et $\mathbf{(C_1) -]C_2[}$;

[meets] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1\ meets\ C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ meets } C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 \text{ meets } C_2, P, f) = \mathcal{C}_e(C_1 \text{ meets } C_2, P, f)$;

[overlaps] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1\ overlaps\ C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ overlaps } C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 \text{ overlaps } C_2, P, f) = \mathcal{C}_e(C_1 \text{ overlaps } C_2, P, f)$;

[starts] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1\ starts\ C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ starts } C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 \text{ starts } C_2, P, f) = \mathcal{C}_e(C_1 \text{ starts } C_2, P, f)$;

[during] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1\ during\ C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ during } C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 \text{ during } C_2, P, f = \mathcal{C}_e(C_1 \text{ during } C_2, P, f)$;

[finishes] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1\ finishes\ C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ finishes } C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 \text{ finishes } C_2, P, f) = \mathcal{C}_e(C_1 \text{ finishes } C_2, P, f)$;

[equals] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1\ equals\ C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ equals } C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 \text{ equals } C_2, P, f) = \mathcal{C}_e(C_1 \text{ equals } C_2, P, f)$;

[lasts δ] Si $\delta \in \mathbb{R}_*^+$, alors $(\mathbf{C_1\ lasts\ \delta}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ lasts } \delta, P, f) = \mathcal{C}_r(C_1)$, et $\mathcal{C}_r(C_1 \text{ lasts } \delta, P, f) = \mathcal{C}_e(C_1 \text{ lasts } \delta, P, f)$;

[at least δ] Si $\delta \in \mathbb{R}_*^+$, alors $(\mathbf{C_1\ at\ least\ \delta}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ at least } \delta, P, f) = \mathcal{C}_r(C_1)$, et $\mathcal{C}_r(C_1 \text{ at least } \delta, P, f) = \mathcal{C}_e(C_1 \text{ at least } \delta, P, f)$;

[at most δ] Si $\delta \in \mathbb{R}_*^+$, alors $(\mathbf{C_1\ at\ most\ \delta}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ at most } \delta, P, f) = \mathcal{C}_r(C_1)$, et $\mathcal{C}_r(C_1 \text{ at most } \delta, P, f) = \mathcal{C}_e(C_1 \text{ at most } \delta, P, f)$;

[then δ] Si $\delta \in \mathbb{R}_*^+$, alors $(\mathbf{C_1\ then\ \delta}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \text{ then } \delta, P, f) = \mathcal{C}_r(C_1)$, et $\mathcal{C}_r(C_1 \text{ then } \delta, P, f) = \mathcal{C}_e(C_1 \text{ then } \delta, P, f)$;

[nommage de propriété] Si $x \in \mathfrak{P} \setminus \{\Diamond\}$, alors $(\mathbf{C_1 \to x}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 \to x, P, f) = \mathcal{C}_r(C_1)$, $\mathcal{C}_r(C_1 \to x, P, f) = \{x, \Diamond\}$;

[cut] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\Diamond\}$, alors $(\mathbf{C_1 ! C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1 ! C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1 ! C_2, P, f) = \mathcal{C}_e(C_1 ! C_2, P, f)$;

4. C'est cette construction qui correspond à celle de l'absence dans [CCK11] présentée dans la Section 1.5.3 et notée $(C_1) - [C_2]$.

[changement d'état] Si $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\diamondsuit\}$, alors $(\mathbf{C_1!!C_2}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(C_1!!C_2, P, f) = \mathcal{C}_r(C_1) \cup \mathcal{C}_r(C_2)$, et $\mathcal{C}_r(C_1!!C_2, P, f) = \mathcal{C}_e(C_1!!C_2, P, f)$;

[évènement de reconnaissance] $(\mathbf{@C_1}, P, f) \in \mathfrak{X}$,
$\mathcal{C}_e(@C_1, P, f) = \mathcal{C}_r(C_1)$, et $\mathcal{C}_r(@C_1, P, f) = \mathcal{C}_r(C_1)$.

Remarque 4. La grammaire du langage des chroniques, sans considération de contexte, s'exprime comme suit sous la forme de Backus-Naur, avec $A \in \mathfrak{N}$, $\delta \in \mathbb{R}_*^+$ et $x \in \mathfrak{P} \setminus \{\diamondsuit\}$:
$C ::= A \mid C\ C \mid C\&C \mid C \mid\mid C \mid (C) - [C] \mid C$ meets $C \mid C$ overlaps $C \mid C$ starts $C \mid C$ during C
$\mid C$ finishes $C \mid C$ equals $C \mid C$ lasts $\delta \mid C$ at least $\delta \mid C$ at most $\delta \mid C$ then $\delta \mid C{\rightarrow}x$
$\mid C!C \mid C!!C \mid @C$

Éclaircissements sur la double notion de contexte La contrainte récurrente $\mathcal{C}_e(C_1) \cap \mathcal{C}_e(C_2) = \{\diamondsuit\}$ assure que tout nom de propriété ne peut être utilisé qu'une seule fois dans une chronique, ce qui est nécessaire pour pouvoir identifier correctement les propriétés. En effet, comme évoqué précédemment, un nom d'attribut se réfère à un évènement ou une chronique spécifique, et un même nom de propriété ne peut donc pas être donné à plusieurs structures au sein d'une même chronique.

Quant à la construction des contextes, la définition générique intuitive évoquée précédemment est celle de la séquence et elle est partagée par la plupart des opérateurs. Le contexte d'évaluation est simplement l'union des contextes résultant des deux sous-chroniques, regroupant ainsi tous les noms de propriété mis en jeu dans l'ensemble de la chronique. Le contexte résultant est identique au contexte d'évaluation.

Cependant, ceci ne peut pas être appliqué à l'ensemble des opérateurs, et c'est là qu'apparaît la nécessité de définir deux contextes. Présentons les trois exceptions.

Le cas de la disjonction Tout d'abord, notons que la disjonction est la seule chronique construite à partir de deux sous-chroniques mais à laquelle n'est pas imposée la contrainte générique d'avoir l'intersection des contextes d'évaluation réduite au singleton $\{\diamondsuit\}$. Au contraire, nous allons nous intéresser aux noms de propriété qui sont employés dans les deux branches de la disjonction. Cette particularité provient du fait que, dans une disjonction, seule l'une des deux sous-chroniques peut être reconnue. Pour qu'un nom d'attribut puisse avoir un sens dans une disjonction, c'est-à-dire pour qu'il soit toujours possible de lui attribuer une valeur en tout cas de figure, un nom d'attribut dans une disjonction doit se référer à un évènement dans chacune des deux branches. En effet, sinon, on ne peut pas assurer que, quelle que soit la branche reconnue, tout nom d'attribut se réfère effectivement à un évènement du flux ce qui est nécessaire pour évaluer le prédicat et la fonction de transformation. Considérons par exemple la chronique $(A{\rightarrow}x\ B{\rightarrow}y) \mid\mid (D{\rightarrow}z\ A{\rightarrow}x)$ qui est une disjonction reconnaissant soit la séquence de deux évènements A suivi de B soit la séquence de D suivi de A. Toute reconnaissance de cette chronique mettra en jeu un évènement de nom de propriété x, mais selon la branche de la disjonction reconnue, elle mettra en jeu un évènement de nom de propriété soit y soit z. Au niveau de la disjonction, on peut donc se référer à x pour lequel des valeurs seront toujours disponibles, mais on ne peut pas se référer à y ou z

comme cela dépend de la branche reconnue. Pour cette raison, le contexte d'évaluation associé à une disjonction est l'intersection des contextes de ses sous-chroniques.

Le cas de l'absence Le cas de l'absence est similaire à celui de la disjonction. Nous souhaitons permettre l'expression de la contrainte suivante, sur les attributs des *deux* sous-chroniques d'une absence : « $(C_1) - [C_2]$ vérifiant le prédicat P » correspond au comportement décrit par « C_1 reconnue sans qu'il n'y ait aucune occurrence de C_2 *vérifiant le prédicat* P durant la reconnaissance de C_1, où P peut porter sur les attributs à la fois de C_1 et de C_2 ». Par exemple, le comportement suivant peut alors être décrit : « Un avion d'ID n a modifié sa fréquence radio à $f \neq 118.075$ (qui est en fait la fréquence recherchée) à l'instant t au plus 5 min après le décollage, et, après t mais avant cette échéance de 5 min, la fréquence radio n'a pas été corrigée par l'avion n ». Pour reconnaître ce comportement, les attributs des *deux* sous-chroniques C_1 et C_2 doivent être comparés pour identifier le même avion n et pour avoir accès à l'instant t. Notons que cet exemple sera développé dans l'application présentée dans la Section 5.3. Pour permettre au prédicat d'avoir accès aux attributs des deux sous-chroniques, le contexte d'évaluation de la chronique doit être l'union des deux contextes résultants des deux sous-chroniques, comme c'est le cas généralement. Cependant, la chronique $(C_1) - [C_2]$ est reconnue s'il n'y a aucune reconnaissance de C_2 pendant la reconnaissance de C_1, donc les deux contextes ne doivent pas être passés dans l'induction à une chronique de plus haut niveau. En effet, comme dans le cas de la disjonction, il n'y a pas nécessairement de valeurs pour les attributs de C_2 car il n'y a pas nécessairement de reconnaissance de C_2. C'est pour cela que l'on introduit la notion de contexte résultant, qui permet de conserver dans le contexte d'évaluation l'union des deux contextes des sous-chroniques, mais de ne passer dans l'induction uniquement le contexte de C_1 en réduisant le contexte résultant à celui-ci.

Le cas du nommage de propriété La dernière exception à la définition générique des contextes est le cas du nommage de propriété. Le rôle de cette construction de nommer les attributs de la chronique. Afin que ce nom puisse être utilisé par la suite, ce nom doit donc être ajouté au contexte résultant de la chronique. Par ailleurs, il a été décidé que, lorsque les propriétés d'une chronique sont ainsi nommées, on ne conserve que les propriétés créées par la fonction de transformation f (qui sont stockées sous le nom anonyme $\diamondsuit$ avant d'être éventuellement nommées) et les anciennes propriétés sont « oubliées ». Ceci se traduit par le fait que le contexte résultant est donc réduit à l'ensemble $\{\diamondsuit, x\}$ où x est le nom de propriété choisi. Notons que si l'on souhaite conserver l'ensemble des propriétés présentes à l'intérieur de la chronique nommée, cela est possible à travers de la fonction f. Le choix d'oublier par défaut ces propriétés a été fait car cela permet d'abstraire la sous-chronique en une sorte d'évènement de plus haut niveau muni d'attributs plus complexes tout en se défaisant d'informations superflues.

2.3 Définition de la sémantique du langage à travers la notion de reconnaissance de chronique

Dans la section précédente, nous avons établi la syntaxe de notre langage des chroniques, en introduisant de nombreuses nouvelles constructions. Dans cette section, nous allons maintenant définir la sémantique de ce langage en définissant la notion de reconnaissance d'une chronique. Dans un premier temps (Section 2.3.1) nous étudierons le modèle de représentation des reconnaissances dans le formalisme. Dans [CCK11], une reconnaissance est représentée par un ensemble. Nous montrerons qu'une représentation arborescente est plus appropriée. Nous définirons dans un second temps (Section 2.3.2) la sémantique du langage des chroniques fondée sur ce formalisme de reconnaissance arborescente.

2.3.1 Passage à une représentation arborescente des reconnaissances

Il s'agit d'étudier ici le formalisme employé pour représenter une reconnaissance de chronique. Rappelons que dans [CCK11], une reconnaissance d'une chronique est un ensemble contenant les indices d'occurrence des évènements ayant donné lieu à la reconnaissance (*cf.* Définition 1.5.3). Nous souhaitons mettre en avant un problème de multiplicité des reconnaissances lié à la structure ensembliste utilisée.

Commençons par étudier l'exemple suivant qui illustre ce problème.

Considérons la chronique $C = (A\ B)\&A$ sur le flux $\varphi = ((a,1),(a,2),(b,3))$.

Avec le formalisme de [CCK11] dans lequel une reconnaissance est un ensemble d'évènements, nous obtenons trois reconnaissances de la chronique C sur le flux φ :

$$R_C(\varphi) = \{\{1,2,3\},\{1,3\},\{2,3\}\}$$

Remarquons que, dans la reconnaissance $\{1,2,3\}$, on ne peut pas identifier quel évènement a participe à la reconnaissance de la séquence $A\ B$ et quel évènement a correspond à l'évènement simple A. Ceci est dû à l'impossibilité de distinguer les ensembles $\{1,3,2\}$ et $\{2,3,1\}$, et donc à distinguer les deux a. Or, comme introduit dans la Section 1.1, l'historisation des évènements, à savoir l'identification exacte de quel évènement a participé à quel morceau de la reconnaissance, est une problématique omniprésente. D'autre part, l'introduction d'attributs rend primordial l'appariement exact des évènements à la chronique pour que les valeurs correctes des propriétés soient considérées. Nous voudrions donc pouvoir différencier les ensembles $\{1,3,2\}$ et $\{2,3,1\}$, ce qui donnerait donc quatre reconnaissances pour la chronique C sur le flux φ. Ce problème a donc également une incidence sur la combinatoire du système.

Pour résoudre cette question, nous proposons de manipuler des reconnaissances sous forme d'arbres binaires plutôt que sous forme de simples ensembles. Davantage d'informations peuvent ainsi être conservées : l'arbre d'une reconnaissance est calqué sur la structure arborescente de la chronique associée et l'on a donc la possibilité de connaître la correspondance exacte entre les noms d'évènements simples de la chronique et les évènements du flux prenant part à la reconnaissance.

Pour l'exemple précédent, cela permet de différencier les appariements des a du flux et ainsi

obtenir quatre reconnaissances. Avec la représentation d'arbre binaire définie par la suite (Définition 14), l'ensemble de reconnaissances est alors :

$$R_C(\varphi) = \{\langle\langle 1,3\rangle,2\rangle, \langle\langle 2,3\rangle,1\rangle, \langle\langle 1,3\rangle,1\rangle, \langle\langle 2,3\rangle,2\rangle\}$$

Par ailleurs, pour favoriser la lecture, nous adoptons un autre système de représentation des évènements, en indiquant les évènements et leur date d'occurrence en lieu et place de leur indice d'occurrence dans le flux, comme annoncé au début de la Section 2.1.1. Nous serons également amenés à ajouter un paramètre d à l'ensemble de reconnaissance, indiquant l'instant jusqu'auquel les évènements du flux ont été pris en compte. Ce paramètre est nécessaire pour l'écriture des ensemble de reconnaissances de certains opérateurs exprimant des contraintes temporelles. Avec ces nouvelles notations, l'ensemble de reconnaissances de l'exemple précédent sera noté :

$$\begin{aligned} R_C(\varphi,3) = \{&\langle\langle(a,1),(b,3)\rangle,(a,2)\rangle, \langle\langle(a,2),(b,3)\rangle,(a,1)\rangle,\\ &\langle\langle(a,1),(b,3)\rangle,(a,1)\rangle, \langle\langle(a,2),(b,3)\rangle,(a,2)\rangle\} \end{aligned}$$

Définissons maintenant formellement la représentation arborescente utilisée. Une reconnaissance r est l'arbre binaire des évènements (e,t) ayant donné lieu à la reconnaissance. La structure de l'arbre de reconnaissance reflète celle de la chronique associée, les feuilles de celui-ci correspondant aux évènements donnant lieu à la reconnaissance.

Les informations qui sont pertinentes sont la structure de l'arbre ainsi que les étiquettes des feuilles [5]. Pour identifier sans ambiguïté les appariements d'évènements avec les sous-chroniques de la chronique étudiée, il n'est pas nécessaire de nommer d'autres nœuds que les feuilles. Nous utilisons donc le formalisme suivant.

Définition 14 (arbres de reconnaissance). L'ensemble $\mathcal{A}(\mathfrak{E})$ des *arbres de reconnaissance sur l'ensemble d'évènements* $\mathfrak{E}$ se définit par induction comme suit, où X est un ensemble d'attributs :

- si $(e,t) \in \mathfrak{E}$, alors $((e,t),X) \in \mathcal{A}(\mathfrak{E})$;
- si $r_1 \in \mathcal{A}(\mathfrak{E})$ et $r_2 \in \mathcal{A}(\mathfrak{E})$, alors $(\langle r_1,r_2\rangle,X) \in \mathcal{A}(\mathfrak{E})$;
- si $r \in \mathcal{A}(\mathfrak{E})$, alors $(\langle r\rangle,X) \in \mathcal{A}(\mathfrak{E})$, $(\langle r,\perp\rangle,X) \in \mathcal{A}(\mathfrak{E})$ et $(\langle \perp,r\rangle,X) \in \mathcal{A}(\mathfrak{E})$.

Nous définissons aussi l'ensemble $\mathcal{F}(r)$ des *feuilles d'un arbre de reconnaissance* r par induction :

- si $r = ((e,t),X)$ avec $(e,t) \in \mathfrak{E}$, alors $\mathcal{F}(r) = \{(e,t)\}$;
- si $r = (\langle r_1,r_2\rangle,X)$ avec $r_1 \in \mathcal{A}(\mathfrak{E})$ et $r_2 \in \mathcal{A}(\mathfrak{E})$, alors $\mathcal{F}(r) = \mathcal{F}(r_1) \cup \mathcal{F}(r_2)$;
- si $r \in \{(\langle r_1,\perp\rangle,X),(\langle \perp,r_1\rangle,X),(\langle r_1\rangle,X)\}$ avec $r_1 \in \mathcal{A}(\mathfrak{E})$, alors $\mathcal{F}(r) = \mathcal{F}(r_1)$.

Notons que, dans la définition précédente, nous avons introduit une notation pour distinguer deux types de paires : les feuilles qui sont des paires composées d'un nom d'évènement et d'une date, notées entre parenthèses (), et les ramifications des arbres, notées entre chevrons $\langle\rangle$.

Avant de définir la sémantique de notre langage à travers la notion de reconnaissance, nous définissons au préalable deux fonctions $T_{\min}$ et $T_{\max}$ retournant respectivement, en fonction d'une reconnaissance r, le premier et le dernier instants auxquels a lieu un évènement participant à r :

5. L'ensemble des feuilles d'un arbre de reconnaissance correspond à la reconnaissance dans le formalisme de [CCK11].

Définition 15 (temps min **et** max**).**

$$T_{\min} : \mathcal{A}(\mathfrak{E}) \to \mathbb{R}, r \mapsto \min\{t : (e,t) \in \mathcal{F}(r)\}$$
$$T_{\max} : \mathcal{A}(\mathfrak{E}) \to \mathbb{R}, r \mapsto \max\{t : (e,t) \in \mathcal{F}(r)\}$$

Ainsi, pour toute reconnaissance r, l'intervalle $[T_{\min}(r), T_{\max}(r)]$ correspond à l'intervalle de temps nécessaire et suffisant pour établir la reconnaissance r. Ces deux fonctions permettront de poser des contraintes temporelles entre les intervalles d'occurrence de chroniques.

2.3.2 Formalisation de la notion de reconnaissance de chronique

Nous pouvons maintenant poser la sémantique de notre langage des chroniques en définissant la notion de reconnaissance d'une chronique tout en intégrant la nouvelle représentation arborescente des reconnaissances. La définition de la sémantique est délicate pour deux raisons principales :

— d'une part, le processus de reconnaissance doit permettre d'*évaluer des prédicats* sur des attributs d'évènement provenant de différentes parties de la chronique ;
— d'autre part, les chroniques doivent être reconnues *en ligne*. Avec ces deux contraintes à l'esprit, nous définissons pour chaque chronique un ensemble de reconnaissances qui correspond à toutes les reconnaissances de la chronique représentées sous forme d'arbres et munies de leurs attributs de reconnaissance associés.

Comme l'on souhaite que le processus de reconnaissance puisse être effectué *au fur et à mesure* tout en étant *exhaustif*, la construction des ensembles de reconnaissances est progressive, ce qui s'exprime par une définition inductive dépendant de l'instant d jusqu'auquel les évènements du flux sont considérés. La constitution de ces ensembles ne nécessite donc pas de connaître par avance l'intégralité du flux d'évènements.

Nous définissons donc, par induction pour chaque chronique, l'ensemble de ses reconnaissances sur un flux d'évènements et jusqu'à une date donnés. Les évènements de cet ensemble sont des couples (r, X) où r est un arbre de reconnaissance et X est l'ensemble d'attributs de reconnaissance associé. Chaque définition peut se décomposer en trois parties :

— l'expression des contraintes temporelles liées à l'opérateur ;
— la vérification du prédicat ;
— la définition de l'ensemble d'attributs de reconnaissance.

Définition 16 (ensembles de reconnaissances). Soit $C \in \mathfrak{X}$ une chronique. Considérons, pour tout prédicat P, un $(\mathfrak{P} \times \mathcal{V})$-modèle $\mathfrak{M}$ dans lequel il existe une interprétation $\hat{P}$ de P.

L'*ensemble des reconnaissances de C sur le flux d'évènements φ jusqu'à la date d* est noté $R_C(\varphi, d)$ et est un sous-ensemble de $\mathcal{A}(\mathfrak{E})$.

L'*ensemble d'attributs associé à une reconnaissance r* est noté X_r. Nous utilisons les notations suivantes : $X_r^{\diamondsuit} = \{(\diamondsuit, v) \in X_r : v \in \mathfrak{A}_e(\mathfrak{P}, \mathcal{V})\}$, qui est un singleton, et $X_r^* = X_r \setminus X_r^{\diamondsuit}$.

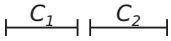
Les ensembles de reconnaissances et les ensembles d'attributs associés aux reconnaissances sont définis par induction comme suit :

— Un ***évènement simple*** **A** vérifiant le prédicat P est reconnu si un évènement de nom A a lieu dans le flux avant l'instant d et si ses attributs vérifient le prédicat P. Les attributs de reconnaissance sont réduits à une unique propriété anonyme contenant les éventuels attributs créés par la fonction f et les attributs de l'évènement du flux.

Si $A \in \mathfrak{N}$, alors :
$R_{(A,P,f)}(\varphi, d) = \{((e,t), X_{(e,t)}) : e = A \wedge \exists i\ \varphi(i) = (e,t) \wedge t \leq d \wedge \hat{P}[\mathcal{D} \circ \alpha((e,t))]$
$\wedge X_{(e,t)} = \{\ (\diamondsuit, X) : \exists X_e \in \mathfrak{A}_e(\mathfrak{P}, \mathcal{V})\ (\ X = X_e \cup \alpha((e,t)) \wedge f \circ \mathcal{D} \circ \alpha((e,t)) = \{(\diamondsuit, X_e)\}\ )\ \}\ \}$
En d'autres termes, les reconnaissances d'un évènement simple suivent les règles suivantes :
— le nom de l'évènement est correct ;
— la date de l'évènement est inférieure à l'horizon ($t \leq d$) ;
— le prédicat, s'il existe, est vérifié sur les attributs de l'évènement ($\hat{P}[\mathcal{D} \circ \alpha((e,t))]$).
Et en supplément, l'ensemble d'attributs associé à la reconnaissance est constitué de la manière suivante. Il contient :
— les attributs de l'évènement dans le flux ($\alpha((e,t))$) ;
— les attributs créés par la fonction f ($f \circ \mathcal{D} \circ \alpha((e,t))$).

— Une ***disjonction*** $\mathbf{C_1} \mid\mid \mathbf{C_2}$ vérifiant un prédicat P est reconnue lorsque l'une des deux sous-chroniques est reconnue et vérifie P. La structure de l'arbre indique quelle branche de la chronique a été reconnue. Les attributs de reconnaissance sont réduits aux attributs dont les noms sont utilisés à la fois dans C_1 et C_2, comme détaillé dans la Section 2.2 (p.51), avec les attributs anonymes créés par la fonction f. On rappelle que la fonction ρ permet d'accéder au nom d'une propriété.
$R_{(C_1||C_2,P,f)}(\varphi, d) = \{(\langle r, \perp\rangle, X_{\langle r,\perp\rangle}) : r \in R_{C_1}(\varphi, d) \wedge \hat{P}[X^*_r]$
$\wedge X_{\langle r,\perp\rangle} = \{x \in X^*_r : \rho(x) \in \mathcal{C}_e(C_1||C_2)\} \cup \mathcal{D} \circ f[X^*_r]\}$
$\cup \{(\langle \perp, r\rangle, X_{\langle \perp,r\rangle}) : r \in R_{C_2}(\varphi, d) \wedge \hat{P}[X^*_r]$
$\wedge X_{\langle \perp,r\rangle} = \{x \in X^*_r : \rho(x) \in \mathcal{C}_e(C_1\ ||\ C_2)\} \cup \mathcal{D} \circ f[X^*_r]\}$

— Une ***séquence*** $\mathbf{C_1 C_2}$ vérifiant P est reconnue lorsque C_2 est reconnue *après* avoir reconnu C_1 et que les deux reconnaissances vérifient P. Les attributs de reconnaissance sont ceux des reconnaissances de C_1 et de C_2 avec les attributs anonymes créés par la fonction f.

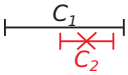


$R_{(C_1\,C_2,P,f)}(\varphi, d) = \{(\langle r_1, r_2\rangle, X_{\langle r_1,r_2\rangle}) : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_2}(\varphi, d)$
$\wedge T_{\max}(r_1) < T_{\min}(r_2) \wedge \hat{P}[X^*_{r_1} \cup X^*_{r_2}]$
$\wedge X_{\langle r_1,r_2\rangle} = X^*_{r_1} \cup X^*_{r_2} \cup \mathcal{D} \circ f[X^*_{r_1} \cup X^*_{r_2}]\}$

— Une ***conjonction*** $\mathbf{C_1 \& C_2}$ vérifiant P est reconnue lorsque à la fois C_1 et C_2 sont reconnues et vérifient P. Les attributs de reconnaissance sont construits comme pour la séquence.
$R_{(C_1\&C_2,P,f)}(\varphi, d) = \{(\langle r_1, r_2\rangle, X_{\langle r_1,r_2\rangle}) : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_2}(\varphi, d)$
$\wedge \hat{P}[X^*_{r_1} \cup X^*_{r_2}]$
$\wedge X_{\langle r_1,r_2\rangle} = X^*_{r_1} \cup X^*_{r_2} \cup \mathcal{D} \circ f[X^*_{r_1} \cup X^*_{r_2}]\}$

— Une ***absence*** $(\mathbf{C_1}) - [\mathbf{C_2}]$ vérifiant un prédicat P est reconnue lorsque C_1 est reconnue sans aucune occurrence de C_2 *vérifiant* P durant la reconnaissance de C_1. *Attention*, dans le cas d'une absence, la signification du prédicat est donc particulière. Les attributs de reconnaissance sont alors réduits à ceux de la reconnaissance de C_1, comme détaillé dans 2.2, complétés des éventuels attributs anonymes créés par la fonction f.

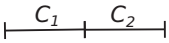
$$R_{((C_1)-[C_2],P,f)}(\varphi,d) = \{(\langle r_1\rangle, X_{\langle r_1\rangle}) : r_1 \in R_{C_1}(\varphi,d) \\ \wedge \forall r_2 \in R_{C_2}(\varphi,d)\ (\ T_{\min}(r_1) > T_{\min}(r_2) \\ \vee T_{\max}(r_1) < T_{\max}(r_2) \\ \vee \neg \hat{P}[X^*_{r_1} \cup X^*_{r_2}]\ ) \\ \wedge X_{\langle r_1\rangle} = X^*_{r_1} \cup \mathcal{D} \circ f[X^*_{r_1}]\}$$



Pour les ensembles de reconnaissances associés aux autres possibilités de bornes de l'absence évoquées en 2.2, seule la contrainte temporelle change :

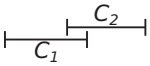
- pour $(C_1)-]C_2[$, elle devient $T_{\min}(r_1) \geq T_{\min}(r_2) \vee T_{\max}(r_1) \leq T_{\max}(r_2)$;
- pour $(C_1) - [C_2[$ [6], elle devient $T_{\min}(r_1) > T_{\min}(r_2) \vee T_{\max}(r_1) \leq T_{\max}(r_2)$;
- pour $(C_1)-]C_2]$, elle devient $T_{\min}(r_1) \geq T_{\min}(r_2) \vee T_{\max}(r_1) < T_{\max}(r_2)$;

— Une chronique **C_1 meets C_2** est reconnue lorsque C_2 est reconnue *exactement après* C_1 et P est vérifié. Les attributs de reconnaissance sont construits comme pour la séquence.



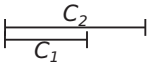
$R_{(C_1 \text{ meets } C_2,P,f)}(\varphi,d)$ *est défini comme la séquence mais avec la contrainte temporelle* $T_{\max}(r_1) = T_{\min}(r_2)$.

— Une chronique **C_1 overlaps C_2** est reconnue lorsque à la fois C_1 et C_2 sont reconnues avec des reconnaissances *se chevauchant temporellement* et P est vérifié. Les attributs de reconnaissance sont construits comme pour la séquence.



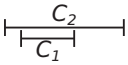
$R_{(C_1 \text{ overlaps } C_2,P,f)}(\varphi,d)$ *est défini comme la séquence mais avec la contrainte temporelle* $T_{\min}(r_1) < T_{\min}(r_2) < T_{\max}(r_1) < T_{\max}(r_2)$.

— Une chronique **C_1 starts C_2** est reconnue lorsque à la fois C_1 et C_2 sont reconnues avec la reconnaissance de C_1 *débutant en même temps* que celle de C_2 et *finissant avant*, et P est vérifié.



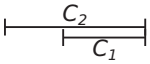
$R_{(C_1 \text{ starts } C_2,P,f)}(\varphi,d)$ *est défini comme la séquence mais avec la contrainte temporelle* $T_{\min}(r_1) = T_{\min}(r_2) \wedge T_{\max}(r_1) < T_{\max}(r_2)$.

— Une chronique **C_1 during C_2** est reconnue lorsque C_1 est reconnue *pendant* une reconnaissance de C_2, et P est vérifié.



$R_{(C_1 \text{ during } C_2,P,f)}(\varphi,d)$ *est défini comme la séquence mais avec la contrainte temporelle* $T_{\min}(r_1) > T_{\min}(r_2) \wedge T_{\max}(r_1) < T_{\max}(r_2)$.

— Une chronique **C_1 finishes C_2** est reconnue lorsque à la fois C_1 et C_2 sont reconnues et que la reconnaissance de C_1 *commence après* celle de C_2 et *se termine au même instant*, et P est vérifié.



$R_{(C_1 \text{ finishes } C_2,P,f)}(\varphi,d)$ *est défini comme la séquence mais avec la contrainte temporelle* $T_{\min}(r_1) > T_{\min}(r_2) \wedge T_{\max}(r_1) = T_{\max}(r_2)$.

— Une chronique **C_1 equals C_2** est reconnue lorsque à la fois C_1 et C_2 sont reconnues sur *exactement le même intervalle de temps*, et P est vérifié.

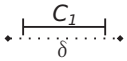
$R_{(C_1 \text{ equals } C_2,P,f)}(\varphi,d)$ *est défini comme la séquence mais avec la contrainte temporelle* $T_{\min}(r_1) = T_{\min}(r_2) \wedge T_{\max}(r_1) = T_{\max}(r_2)$.

6. C'est cette construction qui correspond à celle de l'absence dans [CCK11] présentée dans la Section 1.5.3 et notée $(C_1) - [C_2]$.

— Une chronique $\mathbf{C_1}$ **lasts** δ est reconnue lorsque C_1 est reconnue, P est vérifié, et la taille de l'intervalle de reconnaissance est *exactement* δ.

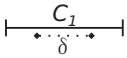
$$R_{(C_1 \text{ lasts } \delta,P,f)}(\varphi,d) = \{(\langle r\rangle, X_{\langle r\rangle}) : r \in R_{C_1}(\varphi,d) \wedge T_{\max}(r) - T_{\min}(r) = \delta \wedge \hat{P}[X_r^*] \wedge X_{\langle r\rangle} = X_r^* \cup \mathcal{D}\circ f[X_r^*]\}$$

— Une chronique $\mathbf{C_1}$ **at most** δ est reconnue lorsque C_1 est reconnue, P est vérifié, et la taille de l'intervalle de reconnaissance est *au plus* δ.



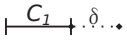
$R_{(C_1 \text{ at most } \delta,P,f)}(\varphi,d)$ *est défini comme la chronique « lasts » mais avec la contrainte temporelle* $T_{\max}(r) - T_{\min}(r) < \delta$.

— Une chronique $\mathbf{C_1}$ **at least** δ est reconnue lorsque C_1 est reconnue, P est vérifié, et la taille de l'intervalle de reconnaissance est *au moins* δ.



$R_{(C_1 \text{ at least } \delta,P,f)}(\varphi,d)$ *est défini comme la chronique « lasts » mais avec la contrainte temporelle* $T_{\max}(r) - T_{\min}(r) > \delta$.

— Une chronique $\mathbf{C_1}$ **then** δ est reconnue lorsque exactement δ unités de temps s'écoulent après une reconnaissance de C_1. L'arbre de reconnaissance conserve l'évènement d'instant temporel pur (τ,t) correspondant à l'instant de la reconnaissance. Ceci permet, entre autre, d'avoir une définition correcte de $T_{\max}$.



$$R_{(C_1 \text{ then } \delta,P,f)}(\varphi,d)=\{(\langle r,(\tau,t)\rangle, X_{\langle r,(\tau,t)\rangle}) : t \le d \wedge r \in R_{C_1}(\varphi,t) \wedge t = T_{\max}(r)+\delta \wedge \hat{P}[X_r^*] \wedge X_{\langle r,(\tau,t)\rangle} = X_r^* \cup \mathcal{D}\circ f[X_r^*]\}$$

— Une chronique de ***nommage de propriété*** $\mathbf{C_1}{\to}x$ est reconnue lorsque C_1 est reconnue. Les attributs de reconnaissance sont les attributs anonymes de la reconnaissance de C_1 mais *renommés* x, avec les nouveaux attributs anonymes créés par la fonction f.

$$R_{(C_1\to x,P,f)}(\varphi,d) = \{(\langle r\rangle, X_{\langle r\rangle}) : r{\in}R_{C_1}(\varphi,d) \wedge \hat{P}[X_r^*] \wedge X_{\langle r\rangle} = \mathcal{R}(X_r^{\diamondsuit},x) \cup \mathcal{D}\circ f[X_r^*]\}$$

— Une chronique de ***cut*** $\mathbf{C_1!C_2}$ est reconnue lorsque C_1 et C_2 sont reconnues successivement et lorsque la reconnaissance de C_2 est la première après celle de C_1 En d'autres termes, il n'y a pas d'autre reconnaissance de C_2 entre les deux reconnaissances sélectionnées.

$$R_{(C_1!C_2,P,f)}(\varphi,d)=\{(\langle r_1,r_2\rangle, X_{\langle r_1,r_2\rangle}) : r_1{\in}R_{C_1}(\varphi,d) \wedge r_2{\in}R_{C_2}(\varphi,d) \wedge T_{\max}(r_1){<}T_{\min}(r_2) \wedge\forall r_2' \in R_{C_2}(\varphi,d)\ (\ T_{\max}(r_1) < T_{\min}(r_2') \Rightarrow (T_{\min}(r_2) < T_{\min}(r_2') \vee (T_{\min}(r_2) = T_{\min}(r_2') \wedge T_{\max}(r_2) \le T_{\max}(r_2'))))\ ) \wedge\hat{P}[X_{r_1}^* \cup X_{r_2}^*] \wedge X_{\langle r_1,r_2\rangle} = X_{r_1}^* \cup X_{r_2}^* \cup \mathcal{D}\circ f[X_{r_1}^* \cup X_{r_2}^*]\}$$

— Une chronique de ***changement d'état*** $\mathbf{C_1!!C_2}$ est reconnue lorsque C_1 et C_2 sont reconnues successivement, la reconnaissance de C_1 étant la dernière avant une reconnaissance de C_2, et la reconnaissance de C_2 étant la première après une reconnaissance de C_1. En d'autres termes, il n'y a aucune autre reconnaissance de C_1 ni de C_2 entre les deux reconnaissances sélectionnées.

$$R_{(C_1!!C_2,P,f)}(\varphi,d)=\{(\langle r_1,r_2\rangle, X_{\langle r_1,r_2\rangle}) : r_1{\in}R_{C_1}(\varphi,d) \wedge r_2{\in}R_{C_2}(\varphi,d) \wedge T_{\max}(r_1){<}T_{\min}(r_2) \wedge\forall r_2' \in R_{C_2}(\varphi,d)\ (\ T_{\max}(r_1) < T_{\min}(r_2') \Rightarrow (T_{\min}(r_2) < T_{\min}(r_2') \vee (T_{\min}(r_2) = T_{\min}(r_2') \wedge T_{\max}(r_2) \le T_{\max}(r_2'))))\ ) \wedge\forall r_1' \in R_{C_1}(\varphi,d)\ (\ T_{\max}(r_1') < T_{\min}(r_2) \Rightarrow (T_{\max}(r_1') \le T_{\max}(r_1))) \wedge\hat{P}[X_{r_1}^* \cup X_{r_2}^*] \wedge X_{\langle r_1,r_2\rangle} = X_{r_1}^* \cup X_{r_2}^* \cup \mathcal{D}\circ f[X_{r_1}^* \cup X_{r_2}^*]\}$$

— Une chronique d'***évènement de reconnaissance*** $@(\mathbf{C_1})$ est reconnue lorsque C_1 est reconnue, mais son intervalle de reconnaissance est réduit à l'instant de reconnaissance.
$R_{(@(C_1),P,f)}(\varphi, d) = \{((e,t), X_{\gamma_r}) : r \in R_{C_1}(\varphi, d) \wedge (e,t) \in \mathcal{F}(r) \wedge t = T_{\max}(r) \wedge \hat{P}[X_r^*] \wedge X_{\gamma_r} = X_r \cup \mathcal{D} \circ f[X_r^*]\}$

Remarque 5. Intuitivement, le *cut* $C_1 \,!\, C_2$ permet de reconnaître la première reconnaissance de C_2 suivant une reconnaissance de C_1, et le *double cut*, ou *changement d'état*, permet de reconnaître la première reconnaissance de C_2 suivant la dernière reconnaissance de C_1. Cependant, il n'y a pas toujours d'unique première ou d'unique dernière reconnaissance. En ce qui concerne C_2, pour les deux opérateurs, les premières reconnaissances sont celles qui commencent et qui s'achèvent aux instants les plus tôt après la reconnaissance de C_1 considérée. Ainsi, sur le flux $\varphi = ((a,1),(b,2),(d,3),(d,4),(e,5),(e,6))$, la chronique $C = A\,!\,(B\ D\ E)$ est reconnue deux fois : $R_C(\varphi, 6) = \{\langle\langle(a,1), \langle\langle(b,2),(d,3)\rangle,(e,5)\rangle\rangle, \langle\langle(a,1), \langle\langle(b,2),(d,4)\rangle,(e,5)\rangle\rangle\}$. Pour le *changement d'état*, les dernières reconnaissances de C_1 considérées sont celles se terminant à l'instant le plus tard, sans considération de leur instant de début. Ainsi, la chronique $C = (A\ B)\,!!\,D$, sur le flux $\varphi = ((a,1),(a,2),(b,3),(d,4),(d,5))$ est reconnue deux fois : $R_C(\varphi, 5) = \{\langle\langle(a,1),(b,3)\rangle,(d,4)\rangle, \langle\langle(a,2),(b,3)\rangle,(d,4)\rangle\}$.

Remarque 6 (NP-complétude). Notons que le problème de la reconnaissance de chroniques dans un flux d'évènements est NP-complet.

En effet, il est immédiat que l'on peut vérifier en temps polynomial qu'un arbre r est une reconnaissance de C dans un flux φ.

De plus, le problème est NP-difficile car le problème SAT peut s'y réduire. Considérons une formule logique P sous forme normale conjonctive. Elle possède n littéraux. Prenons comme évènements des évènements e possédant comme attributs une valeur de vérité pour chaque littéral. Prenons ensuite comme flux d'évènements une suite finie φ d'évènements e couvrant toutes les répartitions de valeurs de vérité possibles pour les n littéraux. Considérons la chronique $C = (E, P)$. Si l'on sait reconnaître C dans φ en temps polynomial, alors on sait également résoudre SAT en temps polynomial car la chronique C est reconnue dans φ si et seulement si la formule P est satisfiable.

Le problème reste NP-complet si l'on ôte la possibilité d'exprimer des contraintes à l'aide de prédicats. Il faut alors encoder la formule logique à l'aide des opérateurs du langage des chroniques, en particulier la conjonction, la disjonction et l'absence.

Nous achevons ici la définition de la sémantique du langage des chroniques, qui donne un sens à la notion de reconnaissance d'une chronique dans un flux d'évènements.

Par la suite, lorsque les valeurs du prédicat P et de la fonction de transformation d'attributs f n'auront pas d'incidence sur le propos, nous nous permettrons de noter C au lieu de (C, P, f) afin d'alléger les notations.

2.4 Propriétés du langage des chroniques

Pour acquérir une meilleure familiarité avec le langage des chroniques, nous proposons d'étudier dans cette section ses principales propriétés.

2.4.1 Définition d'une relation d'équivalence sur les chroniques

Pour étudier les propriétés du langage, nous allons définir une relation d'équivalence sur l'ensemble des chroniques fondée sur l'ensemble des ensembles de feuilles de reconnaissance.

Définition 17 (ensembles des ensembles de feuilles). Pour toute chronique C, tout flux φ et tout réel d, nous définissons l'ensemble des ensembles de feuilles de reconnaissance :

$$\mathfrak{F}_C(\varphi, d) = \{\mathcal{F}(r) : r \in R_C(\varphi, d)\}$$

Notons que cet ensemble correspond à l'ensemble des reconnaissances du formalisme de [CCK11].

Les propriétés du langage des chroniques peuvent être étudiées avec la relation d'équivalence suivante :

Définition 18 (équivalence de chroniques). Soit $C_1 \in \mathfrak{X}$, et $C_2 \in \mathfrak{X}$. On dit que les chroniques C_1 et C_2 sont *équivalentes* si, quel que soit le flux d'évènements φ et quel que soit le réel d, $\mathfrak{F}_{C_1}(\varphi, d) = \mathfrak{F}_{C_2}(\varphi, d)$, c'est-à-dire $\{\mathcal{F}(r) : r \in R_{C_1}(\varphi)\} = \{\mathcal{F}(r) : r \in R_{C_2}(\varphi)\}$. Nous noterons $C_1 \equiv C_2$.

Exemple 7. Notons que, pour deux chroniques C_1 et C_2, les chroniques $C_1 \& C_2$ et $(C_1\, C_2)$ $||$ $(C_2\, C_1)$ ne sont pas équivalentes.

En effet, dans $C_1 \& C_2$, les occurrences de C_1 et de C_2 ne sont pas forcément disjointes et peuvent être entrelacées. Considérons ainsi $C_1 = A\, B$ et $C_2 = B\, D$ où A, B et D sont des noms d'évènement simple, et $\varphi = ((a,1),(b,2),(d,3))$. On a $C_1 \& C_2 = (A\, B) \& (B\, D)$ et $(C_1\, C_2)$ $||$ $(C_2\, C_1) = (A\, B\, B\, D)$ $||$ $(B\, D\, A\, B)$. Alors $\mathfrak{F}_{C_1 \& C_2}(\varphi) = \{\{1,2,3\}\}$ mais $\mathfrak{F}_{(C_1\, C_2)||(C_2\, C_1)}(\varphi) = \{\}$.

On a en revanche équivalence entre $C_1 \& C_2$ et $(C_1\, C_2)$ $||$ $(C_2\, C_1)$ si C_1 et C_2 sont des noms d'évènements simples distincts. Effectivement, si C_1 et C_2 sont des évènements simples, leurs occurrences ne peuvent être entrelacées que si $C_1 = C_2 = A \in \mathfrak{N}$. On note ainsi qu'il n'y a pas équivalence entre les chroniques A et $A \& A$: si on considère le flux $\psi = ((a,1),(a,2))$, alors $\mathfrak{F}_A(\psi) = \{\{1\},\{2\}\}$ mais $\mathfrak{F}_{A \& A}(\psi) = \{\{1\},\{2\},\{1,2\}\}$ car $R_A(\psi, 2) = \{(a,1),(a,2)\}$ et $R_{A \& A}(\psi, 2) = \{\langle (a,1),(a,2)\rangle, \langle (a,2),(a,1)\rangle, \langle (a,1),(a,1)\rangle, \langle (a,2),(a,2)\rangle\}$.

2.4.2 Relations entre ensembles de reconnaissances

Une première propriété exprimant l'incrémentalité du processus de reconnaissance se montre directement par induction. Cette propriété permet de simplifier les démonstrations qui suivent.

Propriété 1. *Une induction directe montre que pour toute chronique C et tous instants d, d' :*

$$d \leq d' \Longrightarrow R_C(\varphi, d) \subseteq R_C(\varphi, d')$$

Avant de démontrer des propriétés sur les chroniques, commençons par faire la remarque suivante qui pourra alléger les démonstrations :

Remarque 7. Une séquence est une conjonction particulière, c'est-à-dire pour toutes chroniques C_1 et C_2, pour tout flux φ et pour tout réel d :

$$R_{C_1\ C_2}(\varphi, d) \subseteq R_{C_1 \& C_2}(\varphi, d)$$

(et donc $\mathfrak{F}_{C_1\ C_2}(\varphi, d) \subseteq \mathfrak{F}_{C_1 \& C_2}(\varphi, d)$). Ceci découle immédiatement de la définition de reconnaissance.

Remarque 8. Comme évoqué dans la Section 2.2, le changement d'état est un cut particulier, et le cut est une séquence particulière. Pour toutes chroniques C_1 et C_2, pour tout flux φ et pour tout réel d :

$$R_{C_1 !! C_2}(\varphi, d) \subseteq R_{C_1 ! C_2}(\varphi, d) \subseteq R_{C_1\ C_2}(\varphi, d)$$

Remarque 9. La conjonction peut s'exprimer comme la disjonction de tous les opérateurs d'Allen :

$C_1 \& C_2 \equiv (C_1 C_2)$ || (C_1 meets C_2) || (C_1 overlaps C_2) || (C_1 starts C_2) || (C_1 during C_2) || (C_1 finishes C_2) || (C_1 equals C_2)

Propriété 2. *Soit $C \in \mathfrak{X}$ une chronique. Soit φ un flux d'évènements. Soit $d \in \mathbb{R}^+$.*
Soit I et J tels que $I \subseteq J$, J soit inclus dans le domaine de φ et I soit un segment initial de J.
Alors $R_C(\varphi_{|I}, d) \subseteq R_C(\varphi_{|J}, d)$.

Démonstration. Démontrons cette propriété par induction sur la chronique C.

Nous donnons ici une démonstration sans considérer les attributs, le prédicat P et la fonction f car ils n'ont aucune incidence sur la preuve et ceci nous permet donc d'alléger les notations.

— Si $C = A \in \mathfrak{N}$, alors :
$R_A(\varphi_{|I}, d) = \{(e,t) : \exists i \in I \ \ \varphi_{|I}(i) = (e,t) \wedge e = a \wedge t \leq d\}$
$\subseteq \{(e,t) : \exists i \in J \ \ \varphi_{|J}(i) = (e,t) \wedge e = a \wedge t \leq d\} = R_A(\varphi_{|J}, d)$ car $I \subseteq J$.

— Si $C = C_1 \,||\, C_2$, alors $R_C(\varphi_{|I}, d) = R_{C_1}(\varphi_{|I}, d) \cup R_{C_2}(\varphi_{|I}, d)$.
Par hypothèse d'induction, $R_{C_1}(\varphi_{|I}, d) \subseteq R_{C_1}(\varphi_{|J}, d)$ et $R_{C_2}(\varphi_{|I}, d) \subseteq R_{C_2}(\varphi_{|J}, d)$,
donc $R_C(\varphi_{|I}, d) \subseteq R_{C_1}(\varphi_{|J}, d) \cup R_{C_2}(\varphi_{|J}, d) = R_C(\varphi_{|J}, d)$.

— Si $C = C_1 \& C_2$, alors $R_C(\varphi_{|I}, d) = \{\langle r_1, r_2\rangle : r_1 \in R_{C_1}(\varphi_{|I}, d) \wedge r_2 \in R_{C_2}(\varphi_{|I}, d)\}$.
Soit $r \in R_C(\varphi_{|I})$. Alors $r = \langle r_1, r_2\rangle$ avec $r_1 \in R_{C_1}(\varphi_{|I}, d)$ et $r_2 \in R_{C_2}(\varphi_{|I}, d)$.
Par hypothèse d'induction, $R_{C_1}(\varphi_{|I}, d) \subseteq R_{C_1}(\varphi_{|J}, d)$ et $R_{C_2}(\varphi_{|I}, d) \subseteq R_{C_2}(\varphi_{|J}, d)$,
donc $r_1 \in R_{C_1}(\varphi_{|J}, d)$ et $r_2 \in R_{C_2}(\varphi_{|J}, d)$, donc $r = \langle r_1, r_2\rangle \in R_C(\varphi_{|J}, d)$.
On a donc bien $R_C(\varphi_{|I}, d) \subseteq R_C(\varphi_{|J}, d)$.

— si $C = C_1\, C_2$, alors $R_C(\varphi_{|I}, d) = \{\langle r_1, r_2\rangle : r_1 \in R_{C_1}(\varphi_{|I}, d) \wedge r_2 \in R_{C_2}(\varphi_{|I}, d) \wedge T_{\max}(r_1) < T_{\min}(r_2)\}$.
Soit $r \in R_C(\varphi_{|I}, d)$. Alors $r = \langle r_1, r_2\rangle$ avec $r_1 \in R_{C_1}(\varphi_{|I}, d)$ et $r_2 \in R_{C_2}(\varphi_{|I}, d)$ et $T_{\max}(r_1) < T_{\min}(r_2)$.
Par hypothèse d'induction, de même que dans le cas précédent, on obtient que $r \in R_C(\varphi_{|J}, d)$.
On a donc bien $R_C(\varphi_{|I}, d) \subseteq R_C(\varphi_{|J}, d)$.

— Si $C = (C_1) - [C_2]$, alors $R_C(\varphi_{|I}, d) = \{r_1 : r_1 \in R_{C_1}(\varphi_{|I}, d) \wedge \forall r_2 \in R_{C_2}(\varphi_{|I}, d)\ (\ T_{\min}(r_1) > T_{\min}(r_2) \vee T_{\max}(r_1) \leq T_{\max}(r_2)\ )\}$.
Soit $r_1 \in R_C(\varphi_{|I}, d)$. Alors $r_1 \in R_{C_1}(\varphi_{|I}, d)$ et $\forall r_2 \in R_{C_2}(\varphi_{|I}, d)\ \ (\ T_{\min}(r_1) > T_{\min}(r_2) \vee T_{\max}(r_1) \leq T_{\max}(r_2)\ )$.
Par hypothèse d'induction, $R_{C_1}(\varphi_{|I}, d) \subseteq R_{C_1}(\varphi_{|J}, d)$, et donc $r_1 \in R_{C_1}(\varphi_{|J}, d)$.
Montrons de plus par l'absurde que $\forall r_2 \in R_{C_2}(\varphi_{|J}, d)\ \ (\ T_{\min}(r_1) > T_{\min}(r_2) \vee T_{\max}(r_1) \leq T_{\max}(r_2)\ )$.
Soit $r_2 \in R_{C_2}(\varphi_{|J}, d)$ tel que $\ T_{\min}(r_1) \leq T_{\min}(r_2) \wedge T_{\max}(r_1) > T_{\max}(r_2)$.
En particulier, $T_{\max}(r_2) < T_{\max}(r_1) \leq \max I$ et $r_2 \in R_{C_2}(\varphi_{|J}, d)$,
donc, comme I est un segment initial de J, $r_2 \in R_{C_2}(\varphi_{|I}, d)$,
donc $\exists r_2 \in R_{C_2}(\varphi_{|I}, d)\ \ (\ T_{\min}(r_1) \leq T_{\min}(r_2) \wedge T_{\max}(r_1) > T_{\max}(r_2)\ )$. Absurde.
Donc $r_1 \in R_C(\varphi_{|J}, d)$, et on a donc bien $R_C(\varphi_{|I}, d) \subseteq R_C(\varphi_{|J}, d)$.
— La démonstration pour les autres opérateurs est analogue à celle de la séquence.

□

Remarque 10. En particulier, pour tout $k \in \mathbb{N}$ et tout $d \in \mathbb{R}$,

$$R_C((u_i)_{i \in [\![1,k[\![}, d) \subseteq R_C((u_i)_{i \in [\![1,k+1[\![}, d)$$

Remarque 11. En revanche, la propriété « pour tout I inclus dans le domaine de φ, $R_C(\varphi_{|I}, d) \subseteq R_C(\varphi, d)$ » n'est pas vérifiée pour toute chronique dans laquelle est utilisée l'opérateur d'absence. Le problème se pose si I n'est pas un segment initial du domaine de φ. Par exemple, prenons $C = (A\ D) - [B[$ et posons $\varphi = ((a,1),(b,2),(d,3))$ avec $I = \{1,3\}$. Alors, $R_C(\varphi_{|I}, d) = \{\langle (a,1),(d,3)\rangle\}$ mais $R_C(\varphi, d) = \{\,\}$, donc $R_C(\varphi_{|I}, d) \nsubseteq R_C(\varphi, d)$.

Il en est de même pour la propriété plus générale « pour tout $I \subseteq J$, $R_C(\varphi_{|I}, d) \subseteq R_C(\varphi_{|J}, d)$ » s'il n'y a pas de condition supplémentaire sur I et J.

2.4.3 Associativité, commutativité, distributivité

Les démonstrations des propriétés suivantes se trouvent dans l'Annexe A.

Propriété 3 (associativité). *La disjonction, la conjonction, la séquence, meets et equals sont associatifs.*

Remarque 12.

1. *« overlaps » n'est pas associatif.*
   En effet, considérons le flux $\varphi = ((a,1),(f,2),(d,3),(b,4),(e,5),(g,6))$, et les chroniques $C_1 = A\,B$, $C_2 = D\,E$ et $C_3 = F\ G$.
   On a alors $\mathfrak{F}_{C_1 \text{ overlaps } C_2}(\varphi, 6) = \{\{1,3,4,5\}\}$ et $\mathfrak{F}_{C_2 \text{ overlaps } C_3}(\varphi, 6) = \{\,\}$,
   Donc $\mathfrak{F}_{(C_1 \text{ overlaps } C_2) \text{ overlaps } C_3}(\varphi, 6) = \{\{1,2,3,4,5,6\}\}$,
   mais $\mathfrak{F}_{C_1 \text{ overlaps } (C_2 \text{ overlaps } C_3)}(\varphi, 6) = \{\,\}$.
2. *« starts » n'est pas associatif.*
   En effet, considérons le flux $\varphi = ((a,1),(b,2),(d,3),(e,4))$, et les chroniques $C_1 = A\,D$, $C_2 = A\,B$ et $C_3 = A\ E$.

On a alors $\mathfrak{F}_{C_1 \text{ starts } C_2}(\varphi, 4) = \{\}$ donc $\mathfrak{F}_{(C_1 \text{ starts } C_2) \text{ starts } C_3}(\varphi, 4) = \{\}$,
Mais $\mathfrak{F}_{C_2 \text{ starts } C_3}(\varphi, 4) = \{\{1, 2, 4\}\}$, donc $\mathfrak{F}_{C_1 \text{ starts } (C_2 \text{ starts } C_3)}(\varphi, 4) = \{\{1, 2, 3, 4\}\}$.

3. *« during » n'est pas associatif.*
En effet, considérons le flux $\varphi = ((a, 1), (b, 2), (d, 3), (e, 4), (f, 5), (g, 6))$, et les chroniques $C_1 = B\,E$, $C_2 = D\,F$ et $C_3 = A\,G$.
On a alors $\mathfrak{F}_{C_1 \text{ during } C_2}(\varphi, 6) = \{\}$ donc $\mathfrak{F}_{C_1 \text{ during } (C_2 \text{ during } C_3)}(\varphi, 6) = \{\}$,
Mais $\mathfrak{F}_{C_2 \text{ during } C_3}(\varphi, 6) = \{\{1, 3, 5, 6\}\}$, donc $\mathfrak{F}_{C_1 \text{ during } (C_2 \text{ during } C_3)}(\varphi, 6) = \{\{1, 2, 3, 4, 5, 6\}\}$.

4. *« finishes » n'est pas associatif.*
En effet, considérons le flux $\varphi = ((a, 1), (b, 2), (d, 3), (e, 4))$, et les chroniques $C_1 = B\,E$, $C_2 = D\,E$ et $C_3 = A\,E$.
On a alors $\mathfrak{F}_{C_1 \text{ finishes } C_2}(\varphi, 4) = \{\}$ donc $\mathfrak{F}_{(C_1 \text{ finishes } C_2) \text{ finishes } C_3}(\varphi, 4) = \{\}$,
Mais $\mathfrak{F}_{C_2 \text{ finishes } C_3}(\varphi, 4) = \{\{1, 3, 4\}\}$, donc $\mathfrak{F}_{C_1 \text{ finishes } (C_2 \text{ finishes } C_3)}(\varphi, 4) = \{\{1, 2, 3, 4\}\}$.

5. *Le changement d'état n'est pas associatif.*
En effet, considérons le flux $\varphi = ((a, 1), (b, 2), (b, 3), (d, 4))$, et les chroniques $C_1 = A!!(B!!D)$ et $C_2 = (A!!B)!!D$.
On a alors $\mathfrak{F}_{C_1}(\varphi, 4) = \{\{1, 3, 4\}\}$ mais $\mathfrak{F}_{C_2}(\varphi, 4) = \{\{1, 2, 4\}\}$.

6. *Le cut n'est pas associatif,* puisque le changement d'état est un cut particulier.

Propriété 4 (commutativité). *La disjonction, la conjonction et equals sont commutatifs.*

Remarque 13.
1. *La séquence n'est pas commutative.*
En effet, considérons le flux $\varphi = ((a, 1), (b, 2))$, et les chroniques $A\,B$ et $B\,A$.
On a alors $\mathfrak{F}_{A\,B}(\varphi, 2) = \{\{1, 2\}\}$ mais $\mathfrak{F}_{B\,A}(\varphi, 2) = \{\}$.

2. *meets, overlaps, starts, during, finishes, le cut et le changement d'état ne sont pas commutatifs.*
En effet, comme pour la séquence, les propriétés temporelles qui caractérisent ces opérateurs sont exprimées par des inégalités et ne sont donc pas commutatives.

Propriété 5 (distributivité). *Tous les opérateurs sont distributifs sur la disjonction.*

Remarque 14. *La distributivité de tout opérateur sur un opérateur autre que la disjonction n'est pas vérifiée.*

En effet, considérons deux opérateurs $\circledast$ et $\circledcirc$. Si $\circledast$ est distributif sur $\circledcirc$, alors, en particulier, l'égalité suivante est vérifiée, où A, B et D sont des évènements simples : $A \circledast (B \circledcirc D) \equiv (A \circledast B) \circledcirc (A \circledast D)$. Une reconnaissance de la chronique du membre de gauche ne peut correspondre qu'à un unique évènement a. En revanche, si $\circledcirc$ n'est pas une disjonction, deux évènements a distincts peuvent participer à une même reconnaissance de la chronique du membre de droite. L'équivalence n'est donc pas vérifiée si $\circledcirc$ n'est pas une disjonction, tout opérateur n'est ainsi pas distributif sur un opérateur autre que la disjonction.

Considérons par exemple le cas de la distributivité de la séquence sur la conjonction, avec le flux $\varphi = ((a, 1), (b, 2), (a, 3), (d, 4))$, et les chroniques $A\,(B \& D)$ et $(A\,B) \& (A\,D)$. On a alors $\{1, 2, 3, 4\} \in \mathfrak{F}_{(A\,B) \& (A\,D)}(\varphi, 4)$ mais $\{1, 2, 3, 4\} \notin \mathfrak{F}_{A\,(B \& D)}(\varphi, 4)$.

Pour les opérateurs liés aux durées, et qui n'ont donc pas de sens sur des évènements simples, le même raisonnement s'applique sur les évènements simples qui constituent chaque sous-chronique.

Par exemple, pour le cas de la distributivité de la séquence sur starts, considérons le flux $\varphi = ((a,1),(b,2),(d,3),(e,4),(b,5),(f,6),(g,7))$, et les chroniques $(A\,B)\,((D\,E)$ starts $(F G))$ et $(A\,B\,D\,E)$ starts $(A\,B\,F\,G)$. Alors :

$$\{1,2,3,4,5,6,7\} \in \mathfrak{F}_{(A\,B\,D\,E)\text{ starts }(A\,B\,F\,G)}(\varphi,7)$$

mais :

$$\{1,2,3,4,5,6,7\} \notin \mathfrak{F}_{(A\,B)\,((D\,E)\text{ starts }(F G))}(\varphi,7)$$

2.5 Gestion du temps continu à l'aide d'une fonction Look-ahead

Cette section s'attache à la gestion du temps dans la mise en œuvre du processus de reconnaissance. Des opérations de contraintes temporelles ont été ajoutées au langage et la syntaxe comme la sémantique ont été adaptées pour considérer un modèle de temps continu. Dans une optique d'implémentation, la considération d'un temps continu soulève des problèmes et rend nécessaire la définition d'une fonction « Look-ahead » indiquant un prochain instant $T_C(\varphi,d)$ jusqu'auquel le système n'a pas besoin d'être réexaminé car on a l'assurance qu'aucune reconnaissance ne va se produire jusqu'à cet instant. En effet, les systèmes considérés sont asynchrones, et cette fonction fournit le prochain instant où les chroniques peuvent évoluer. En indiquant le moment où observer le système, cette fonction permet de ne pas avoir à le surveiller constamment pour une reconnaissance éventuelle, ce qui serait impossible du fait du modèle de temps continu (et ce qui serait de toute façon trop exigeant même si le modèle de temps pouvait être discrétisé). La fonction « Look-ahead » est inférée de la sémantique du langage présentée dans la Définition 16 et est définie comme suit :

Définition 19 (fonction « Look-ahead »). Soit $d \in \mathbb{R}$, $C \in \mathfrak{X}$ et φ un flux d'évènements. Nous définissons $T_C(\varphi,d)$ par induction sur la chronique C :

- si $A \in \mathfrak{N}$ et $C = (A,P,f)$, alors $T_C(\varphi,d) = \min\{t : \exists i \in \mathbb{N}\ \exists e \in \mathfrak{N}\ \varphi(i) = (e,t) \wedge t > d\}$;
- pour tout $\circledast \in \{||, \&, \text{“ ”}, (\ ) - [\ ], (\ ) - [\ [, (\ )-]\ ], (\ )-]\ [$, meets, overlaps, starts, during, finishes, equals, !, !!$\}$, si $C = (C_1 \circledast C_2, P, f)$ avec $C_1 \in \mathfrak{X}$ et $C_2 \in \mathfrak{X}$, alors $T_C(\varphi,d) = \min\{T_{C_1}(\varphi,d), T_{C_2}(\varphi,d)\}$;
- si $C = (C_1$ lasts $\delta, P, f)$, alors $T_C(\varphi,d) = T_{C_1}(\varphi,d)$;
- si $C = (C_1$ at most $\delta, P, f)$, alors $T_C(\varphi,d) = T_{C_1}(\varphi,d)$;
- si $C = (C_1$ at least $\delta, P, f)$, alors $T_C(\varphi,d) = T_{C_1}(\varphi,d)$;
- si $C = (C_1$ then $\delta, P, f)$, alors nous définissons tout d'abord l'instant $\tau_{C_1,\delta}(\varphi,d)$ correspondant au plus petit instant $t+\delta$ tel qu'il y a eu une nouvelle reconnaissance de C_1 dans le flux φ à l'instant t et tel que $t \leq d < t+\delta$, comme suit :
$$\tau_{C_1,\delta}(\varphi,d) = \min\{T_{\max}(r_1) + \delta : r_1 \in R_{C_1}(\varphi,d) \wedge T_{\max}(r_1) + \delta > d\}$$
Alors $T_C(\varphi,d) = \min\{\tau_{C_1,\delta}(\varphi,d), T_{C_1}(\varphi,d)\}$;

— si $C = (C_1 \rightarrow x, P, f)$, alors $T_C(\varphi, d) = T_{C_1}(\varphi, d)$;
— si $C = (@C_1, P, f)$, alors $T_C(\varphi, d) = T_{C_1}(\varphi, d)$.

Remarque 15. Notons que l'opérateur « then » se distingue des autres opérateurs du fait que la reconnaissance de la chronique découle du temps qui passe et non de l'occurrence d'un évènement dans le flux. C'est cette chronique qui rend nécessaire la définition de la fonction Look-ahead.

Nous pouvons alors démontrer la propriété suivante, qui caractérise le comportement recherché de la fonction.

Propriété 6 (Look-ahead). *Après l'instant d, l'ensemble des reconnaissances de la chronique C n'évoluera pas avant au moins l'instant* $T_C(\varphi, d)$.

Plus formellement, pour toute chronique $C \in \mathfrak{X}$ *et pour tout instant* $d \in \mathbb{R}$ *:*

$$\forall t_1 \in [d, T_C(\varphi, d)[\;\; R_C(\varphi, d) = R_C(\varphi, t_1)$$

Démonstration. Soit $d \in \mathbb{R}$. Nous montrons cette propriété par induction sur la chronique C.

Pour simplifier les notations, on s'affranchit ici des attributs, du prédicat P et de la fonction f. La démonstration complète est analogue.

Soit $t_1 \in [d, T_C(\varphi, d)[$. Par la Propriété 1, il ne reste à montrer que l'inclusion $R_C(\varphi, d) \supseteq R_C(\varphi, t_1)$.

— Si $C = A \in \mathfrak{N}$.
Par définition, on a $R_A(\varphi, t_1) = \{(e, t) : \exists i \ \varphi(i) = (e, t) \wedge e = a \wedge t \leq t_1\}$,
et $R_A(\varphi, d) = \{(e, t) : \exists i \ \varphi(i) = (e, t) \wedge e = a \wedge t \leq d\}$.
Par l'absurde, soit $(e_0, t_0) \in R_A(\varphi, t_1) \setminus R_A(\varphi, d)$.
On a donc $t_0 < t_1 < T_A(\varphi, d)$, i.e. $t_0 < \min\{t : \exists i \in \mathbb{N} \ \exists e \in \mathfrak{N} \ \varphi(i) = (e, t) \wedge t > d\}$.
Or $t_0 > d$ car $(e_0, t_0) \notin R_A(\varphi, d)$,
donc $t_0 \in \{t : \exists i \in \mathbb{N} \ \exists e \in \mathfrak{N} \ \varphi(i) = (e, t) \wedge t > d\}$ et, en particulier, $t_0 < t_0$. Absurde.
D'où $R_A(\varphi, t_1) \setminus R_A(\varphi, d) = \emptyset$, et donc $R_A(\varphi, d) \supseteq R_A(\varphi, t_1)$.

— Si $C = C_1 \,||\, C_2$.
On note que $t_1 < T_C(\varphi, d) = \min\{T_{C_1}(\varphi, d), T_{C_2}(\varphi, d)\}$ et donc $t_1 < T_{C_1}(\varphi, d)$ et $t_1 < T_{C_2}(\varphi, d)$.
$R_C(\varphi, t_1) = \{\langle r, \perp\rangle : r \in R_{C_1}(\varphi, t_1)\} \cup \{\langle \perp, r\rangle : r \in R_{C_2}(\varphi, t_1)\}$
$= \{\langle r, \perp\rangle : r \in R_{C_1}(\varphi, d)\} \cup \{\langle \perp, r\rangle : r \in R_{C_2}(\varphi, d)\}$ par hypothèse d'induction
$= R_C(\varphi, d)$

— La démonstration relative aux opérateurs de conjonction, de séquence, d'absence, meets, overlaps, starts, during, finishes, equals, lasts δ, at most δ, at least δ, cut, et changement d'état est identique à celle du cas précédent (car la définition de $T_C(\varphi, d)$ est identique pour ces opérateurs).

— Si $C = C_1$ then δ.
On a $R_C(\varphi, d) = \{\langle r_1, (\tau, i)\rangle : i \leq d \wedge r_1 \in R_{C_1}(\varphi, i) \wedge i = T_{\max}(r_1) + \delta\}$,
Et : $R_C(\varphi, t_1) = \{\langle r_1, (\tau, j)\rangle : j \leq t_1 \wedge r_1 \in R_{C_1}(\varphi, j) \wedge j = T_{\max}(r_1) + \delta\}$.
$= R_C(\varphi, d) \cup \{r_1 : \exists j \;\; d < j \leq t_1 \Rightarrow (r_1 \in R_{C_1}(\varphi, j) \wedge j = T_{\max}(r_1) + \delta)\}$
On souhaite montrer que $\{\langle r_1, (\tau, j)\rangle : d < j \leq t_1 \wedge r_1 \in R_{C_1}(\varphi, j) \wedge j = T_{\max}(r_1) + \delta\}$ est

vide. Par l'absurde supposons $\langle r_1, (\tau, j)\rangle$ un élément de cet ensemble.
Alors $d < j \leq t_1$ et $r_1 \in R_{C_1}(\varphi, j) \wedge j = T_{\max}(r_1) + \delta$.
Comme $t_1 < T_C(\varphi, d) \leq T_{C_1}(\varphi, d)$, $j \in]d, t_1]$ implique que $j \in [d, T_{C_1}(\varphi, d)[$.
On peut donc appliquer l'hypothèse d'induction qui nous donne que $R_{C_1}(\varphi, d) = R_{C_1}(\varphi, j)$.
On a donc que $r_1 \in R_{C_1}(\varphi, d)$.
De plus, on a bien que $j = T_{\max}(r_1) + \delta > d$,
Donc $j = T_{\max}(r_1) + \delta \in \{T_{\max}(r_1) + \delta : r_1 \in R_{C_1}(\varphi, d) \wedge T_{\max}(r_1) + \delta > d\}$.
Or $j \leq t_1 < T_{C_1}(\varphi, d) \leq \tau_{C_1,\delta}(\varphi, d) = \min\{T_{\max}(r_1)+\delta : r_1 \in R_{C_1}(\varphi, d) \wedge T_{\max}(r_1)+\delta > d\}$.
Absurde.
Donc $R_C(\varphi, t_1) = R_C(\varphi, d)$.

— Si $C = C_1 \rightarrow x$ ou si $C = @C_1$, la démonstration est immédiate car $T_C(\varphi, d) = T_{C_1}(\varphi, d)$.

□

2.6 Tableau récapitulatif informel des propriétés du langage des chroniques

Le Tableau 2.1 présente un récapitulatif informel de l'ensemble des constructions et propriétés du langage des chroniques qui ont été présentées dans ce chapitre.

TABLEAU 2.1 – Récapitulatif informel des constructions et propriétés du langage des chroniques

Nom	Chronique	Pré-requis $\mathcal{C}_e(C_1)\cap\mathcal{C}_e(C_2)=\{\Diamond\}$	$\mathcal{C}_e$	$\mathcal{C}_r$	Forme de l'arbre	Condition temporelle	Commut.	Assoc.	Look-ahead
évènement simple	A		$\{\Diamond\}$	$\mathcal{C}_e$	(e,t)				$\min\{t{:}\varphi(i){=}(e,t) \wedge t > d\}$
séquence	$C_1\ C_2$	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\max}(r_1) < T_{\min}(r_2)$		•	$\min\{T_{C_1}, T_{C_2}\}$
conjonction	$C_1 \& C_2$	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$		•	•	$\min\{T_{C_1}, T_{C_2}\}$
disjonction	$C_1 \mid\mid C_2$		$\cap$	$\mathcal{C}_e$	$\langle r_1, \bot\rangle$, $\langle \bot, r_2\rangle$		•	•	$\min\{T_{C_1}, T_{C_2}\}$
absence	$(C_1) - [C_2[$	•	$\cup$	$\mathcal{C}_r(C_1)$	$\langle r_1\rangle$	$\forall r_2\ (\ T_{\min}(r_1){>}T_{\min}(r_2)$ $\vee T_{\max}(r_1){<}T_{\max}(r_2)\ )$			$\min\{T_{C_1}, T_{C_2}\}$
meets	C_1 meets C_2	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\max}(r_1) = T_{\min}(r_2)$		•	$\min\{T_{C_1}, T_{C_2}\}$
overlaps	C_1 overlaps C_2	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\min}(r_1){<}T_{\min}(r_2)$ $<T_{\max}(r_1){<}T_{\max}(r_2)$			$\min\{T_{C_1}, T_{C_2}\}$
starts	C_1 starts C_2	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\min}(r_1) = T_{\min}(r_2)$ $\wedge T_{\max}(r_1){<}T_{\max}(r_2)$			$\min\{T_{C_1}, T_{C_2}\}$
during	C_1 during C_2	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\min}(r_1) > T_{\min}(r_2)$ $\wedge T_{\max}(r_1){<}T_{\max}(r_2)$			$\min\{T_{C_1}, T_{C_2}\}$
finishes	C_1 finishes C_2	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\min}(r_1) > T_{\min}(r_2)$ $\wedge T_{\max}(r_1){=}T_{\max}(r_2)$			$\min\{T_{C_1}, T_{C_2}\}$
equals	C_1 equals C_2	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\min}(r_1) = T_{\min}(r_2)$ $\wedge T_{\max}(r_1){=}T_{\max}(r_2)$	•	•	$\min\{T_{C_1}, T_{C_2}\}$
lasts	C lasts δ		$\mathcal{C}_r$	$\mathcal{C}_e$	$\langle r\rangle$	$T_{\max}(r){-}T_{\min}(r){=}\delta$			T_C
at least	C at least δ		$\mathcal{C}_r$	$\mathcal{C}_e$	$\langle r\rangle$	$T_{\max}(r){-}T_{\min}(r){>}\delta$			T_C
at most	C at most δ		$\mathcal{C}_r$	$\mathcal{C}_e$	$\langle r\rangle$	$T_{\max}(r){-}T_{\min}(r){<}\delta$			T_C
then	C then δ		$\mathcal{C}_r$	$\mathcal{C}_e$	$\langle r, (\tau, i)\rangle$	$i = T_{\max}(r){+}\delta$			$\min\{\tau_{C_1,\delta}(\varphi, d), T_{C_1}(\varphi, d)\}$
nommage	$C{\rightarrow}x$		$\mathcal{C}_r$	$\{x, \Diamond\}$	$\langle r\rangle$				T_C
cut	$C_1 ! C_2$	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\max}(r_1) < T_{\min}(r_2)\ \ldots$			$\min\{T_{C_1}, T_{C_2}\}$
changement d'état	$C_1 !! C_2$	•	$\cup$	$\mathcal{C}_e$	$\langle r_1, r_2\rangle$	$T_{\max}(r_1) < T_{\min}(r_2)\ \ldots$			$\min\{T_{C_1}, T_{C_2}\}$
évènement de reco.	$@C$		$\mathcal{C}_r$	$\mathcal{C}_e$	(e,t)				T_C

2.7 Conclusion

Dans ce chapitre, nous avons largement étendu le langage des chroniques présenté dans [CCK11]. Nous avons ajouté de nombreux opérateurs, augmentant ainsi l'expressivité de notre langage. Notamment, nous avons ajouté dix constructions exprimant toutes les contraintes temporelles possibles en transposant dans notre formalisme la logique d'intervalles d'Allen. Nous avons également formalisé la notion de propriété liée à un évènement du flux à analyser. Ceci nous a ensuite permis de doter le langage des chroniques de la possibilité d'exprimer des contraintes sur ces attributs d'évènement.

Pour l'ensemble de ces constructions, nous avons formellement défini la syntaxe et la sémantique associées. Pour ce faire, nous avons été amenés à raffiner la représentation des reconnaissances en passant d'un formalisme ensembliste à un formalisme arborescent conservant davantage d'informations sur la reconnaissance.

Nous avons étudié les principales propriétés du langage étendu des chroniques. Nous avons également défini une fonction de Look-ahead qui permet de formaliser la gestion du temps continu dans le cadre d'une implémentation éventuelle du processus de reconnaissance, en indiquant à quels instants interroger le système.

Nous possédons donc maintenant un cadre théorique formel complet pour effectuer de la reconnaissance de comportements. Il s'agit donc maintenant d'implémenter un processus de reconnaissance de chroniques fondé sur cette base théorique. Nous allons construire deux tels modèles :

— dans les Chapitres 3 et 4, nous allons définir un modèle à l'aide de réseaux de Petri colorés permettant de valider les principes de reconnaissance en faisant notamment ressortir les problèmes de concurrence ;
— dans le Chapitre 5, nous implémentons un modèle de reconnaissance sous la forme d'une bibliothèque C++ appelée Chronicle Recognition Library (crl).

Chapitre 3

Un modèle de reconnaissance en réseaux de Petri colorés dit « à un seul jeton »

Sommaire

Dans le Chapitre 2, nous avons formalisé un langage de description de comportements, le langage des chroniques, et sa sémantique associée, définissant ainsi la notion de reconnaissance d'une chronique dans un flux d'évènements. Il s'agit maintenant de proposer une implémentation du processus de reconnaissance dont il est possible de montrer que l'exécution respecte la sémantique préalablement définie dans la Section 2.3.2. En effet, dans l'optique de traiter des applications critiques (par exemple s'assurer qu'un avion sans pilote respecte bien des procédures de sécurité

données, comme dans la Section 5.3), il faut que le processus de reconnaissance soit fiable et robuste car il n'y a pas de droit à l'erreur. Nous allons développer deux implémentations du système de reconnaissance de comportements permettant cette vérification.

Dans ce chapitre et le suivant, nous commençons par définir, à l'aide de réseaux de Petri colorés, un premier modèle de reconnaissance d'un sous-langage des chroniques restreint aux opérateurs de séquence, de conjonction, de disjonction et d'absence, et qui ne permet donc pas l'expression de contraintes temporelles ni de contraintes sur des attributs d'évènement. Pour chaque chronique, un réseau, construit par induction, calcule l'ensemble des reconnaissances de la chronique sur un flux d'évènements donné. Une version initiale de ce modèle est présentée dans [CCK11], mais il n'est pas totalement formalisé car il faut réaliser la démonstration de son adéquation avec la sémantique des chroniques. Ce modèle, contrairement à ceux présentés dans le Chapitre 4, ne possède qu'un jeton par place. C'est pourquoi nous nous y référons comme le modèle « à un seul jeton ».

Dans ce chapitre, nous complétons la formalisation des réseaux de Petri colorés présentés dans [CCK11] et nous nous attaquons à la vérification de son adéquation [CCKP12a] :

— nous posons une nouvelle définition théorique de la notion de fusion de réseaux de Petri colorés ;
— nous corrigeons les modèles de la conjonction et de l'absence pour qu'ils fonctionnent dans tous les cas de composition ;
— nous proposons une formalisation de la construction des réseaux en faisant ressortir une structure commune à tous les réseaux afin de pouvoir les définir par induction ;
— nous formalisons également les règles d'exécution des réseaux pour bien obtenir les ensembles de reconnaissances des chroniques concernées ;
— nous démontrons la correction des réseaux vis-à-vis de la sémantique ensembliste de la Section 2.3.2 pour les chroniques incluant, au plus, une absence au plus haut niveau.

Dans une première Section (3.1), nous commençons par rappeler la définition des réseaux de Petri colorés en introduisant la notion d'arcs inhibiteurs ainsi qu'une nouvelle définition de fusion. Nous construisons ensuite formellement dans la Section 3.2 les réseaux de Petri colorés modélisant la reconnaissance de chroniques ; puis, dans la Section 3.3, nous décrivons le comportement des réseaux lorsqu'ils suivent une certaine règle d'exécution définie en 3.3.6. Le modèle est alors complètement formalisé. Ceci nous permet de montrer dans la Section 3.4 que, pour des constructions faisant intervenir au plus une absence au plus haut niveau, les reconnaissances produites par les réseaux correspondent exactement à celles définies par la sémantique du langage. Nous achevons ce chapitre en étudiant dans la Section 3.5 la complexité des réseaux construits.

3.1 Définition du formalisme des réseaux de Petri colorés

Commençons par définir le formalisme des réseaux de Petri colorés employé par la suite pour modéliser le processus de reconnaissance. Dans la Section 3.1.1, nous introduisons les notions de type et d'expression. Nous donnons ensuite dans la Section 3.1.2 une définition des réseaux de Petri colorés. Cette définition est complétée dans la Section 3.1.3 par une nouvelle notion de fusion de réseaux, et dans la Section 3.1.4 par des arcs inhibiteurs.

3.1.1 Types et expressions

Il s'agit ici de fournir un cadre formel pour la description des types de données et des fonctions utilisés dans les réseaux (voir la Section 3.2.1 pour la description concrète des types et expressions de nos réseaux).

À partir d'un ensemble de types de base, nous construisons l'ensemble des types de fonction possibles :

Définition 20 (signature d'expression typée). Soit un ensemble $\mathcal{B}$ dont les éléments seront des *types de base*.

On définit inductivement l'*ensemble des types fonctionnels* de $\mathcal{B}$, noté $\mathfrak{T}_\mathcal{B}$, par :

$$\frac{b \in \mathcal{B}}{b \in \mathfrak{T}_\mathcal{B}} \text{ (type de base)} \qquad \frac{\tau_1, \ldots, \tau_n, \tau \in \mathfrak{T}_\mathcal{B}}{\tau_1 \times \cdots \times \tau_n \to \tau \in \mathfrak{T}_\mathcal{B}} \text{ (type fonctionnel)}$$

On définit l'*arité* d'un type $\chi \in \mathfrak{T}_\mathcal{B}$, notée $\text{ari}(\chi)$, par :
— si $\chi \in \mathcal{B}$, alors $\text{ari}(\chi) = 0$
— si $\chi = \tau_1 \times \cdots \times \tau_n \to \tau$, alors $\text{ari}(\chi) = n$

Si $\mathcal{B}$ est un ensemble de types de base, $\mathcal{F}$, un ensemble de fonctions, et $\sigma : \mathcal{F} \longrightarrow \mathfrak{T}_\mathcal{B}$ une fonction de typage, on appelle $\Sigma = (\mathcal{B}, \mathcal{F}, \sigma)$ une *signature d'expression typée*.

À partir de cette signature, on peut définir l'ensemble des expressions qui sont employées dans les gardes et les étiquettes des arcs :

Définition 21 (ensemble des expressions). Soit $\Sigma = (\mathcal{B}, \mathcal{F}, \sigma)$ une signature d'expression typée.

Un ensemble de *variables $\mathcal{B}$-typées* est un couple (V, σ) où V est un ensemble de variables et $\sigma : V \to \mathcal{B}$ est une fonction de typage.

On définit par induction l'ensemble des *V-expressions de* Σ, noté $\text{Expr}_{\Sigma(V)}$, et on prolonge canoniquement la fonction σ à $\text{Expr}_{\Sigma(V)}$ par :

$$\frac{x \in V}{x \in \text{Expr}_{\Sigma(V)}} \text{ (variable)} \qquad \frac{c \in \mathcal{F} \quad \text{ari}(\sigma(c)) = 0}{c \in \text{Expr}_{\Sigma(V)}} \text{ (constante)}$$

$$\frac{f \in \mathcal{F} \quad e_1, \ldots, e_n \in \text{Expr}_{\Sigma(V)} \quad \sigma(f) = \sigma(e_1) \times \cdots \times \sigma(e_n) \to \tau}{f(e_1, \ldots, e_n) \in \text{Expr}_{\Sigma(V)} \quad \sigma(f(e_1, \ldots, e_n)) = \tau} \text{ (fonction)}$$

3.1.2 Réseaux de Petri colorés

Un réseau de Petri coloré est un graphe biparti composé de deux types de nœuds : des *places* représentées par des ellipses, et des *transitions* représentées par des rectangles. Des arcs orientés lient des places à des transitions. Ils sont étiquetés par des expressions. On les appelle *arcs en entrée* et, dans les réseaux que nous utilisons, ils sont étiquetés par des noms de variables. D'autres arcs orientés lient des transitions à des places. On les appelle *arcs en sortie* et ils sont étiquetés par des expressions de fonctions dans lesquelles des variables des arcs d'entrée de la transition, appelées *variables d'entrée*, peuvent apparaître.

Les places sont typées et stockent un ou plusieurs éléments de leur type, appelés *jetons*, qui constituent le *marquage* de la place. Dans les réseaux de Petri de ce chapitre, chaque place ne contiendra qu'un seul jeton. En revanche, dans le Chapitre 4, nous construisons un modèle tirant parti de la possibilité d'avoir plusieurs jetons par place.

Les transitions peuvent être munies de *gardes*, c'est-à-dire de conditions booléennes, dans lesquelles peuvent apparaître les variables d'entrée de la transition. Une transition peut être tirée lorsque les deux conditions suivantes sont réunies :

(i) si elle possède une garde, celle-ci est vérifiée ;

(ii) toutes les places liées par un arc d'entrée à la transition possèdent un jeton (c'est-à-dire qu'une valeur peut être attribuée à chaque variable d'entrée de la transition).

Tirer une transition revient à consommer les jetons indiqués sur les arcs d'entrée de la transition et à appliquer les fonctions étiquetant les arcs de sortie de la transition, c'est-à-dire à ajouter la valeur de ces fonctions au contenu des places liées par des arcs de sortie à la transition. Ainsi, au fur et à mesure que les transitions sont tirées, le marquage des places évolue.

Remarque 16 (multiples tirages successifs possibles de transitions). On remarque donc que, pour un marquage donné d'un réseau, il peut y avoir plusieurs séquences possibles de transitions tirables. Ce comportement non déterministe – dans le sens où il ne mène pas nécessairement toujours au même marquage – peut être souhaité pour exprimer différentes possibilités d'évolution d'un système. Cependant, notre travail sur les réseaux de Petri nécessite d'obtenir systématiquement le même marquage à l'issue du traitement occasionné par un évènement du flux. En effet, on doit pouvoir lire dans celui-ci l'ensemble des reconnaissances de la chronique. Notons que cela n'implique pas qu'il n'existe qu'une seule stratégie de tirage de transitions adaptée, comme on le verra dans la Section 4.2. *Il nous faudra donc soit accompagner chaque réseau d'une stratégie de tirage, soit montrer que toute séquence possible de transitions donne le même marquage final du réseau.*

Définissons maintenant plus formellement un réseau de Petri coloré, d'après la définition de [JK09].

Définition 22 (réseau de Petri coloré). Un *réseau de Petri coloré* N est un 9-uplet $(P, T, A, \mathcal{B}, V, C, G, EX, I)$ où :

1. P est un ensemble fini de *places* ;
2. T est un ensemble fini de *transitions* disjoint de P ;
3. $A \subseteq P \times T \uplus T \times P$ est un ensemble d'*arcs* orientés ;
4. $\mathcal{B}$ est un ensemble fini de *types* ;
5. V est un ensemble fini de *variables* typées par une fonction σ telle que $\forall v \in V\ \sigma(v) \in \mathcal{B}$;
6. $C : P \to \mathcal{B}$ est une *fonction de coloriage* qui à chaque place associe un type ;
7. $G : T \to \mathrm{Expr}_{\Sigma(V)}$ est une *fonction de garde* qui à chaque transition associe une garde de type booléenne et dont les variables sont des éléments de V ;

8. $EX : A \to \mathrm{Expr}_{\Sigma(V)}$ est une *fonction d'expression d'arcs* qui à chaque arc associe une expression telle que, pour tout arc $a \in A$, $EX(a)$ est du même type que la place à laquelle est relié l'arc a ;
9. $I : P \to \mathrm{Expr}_{\Sigma(V)}$ est une *fonction d'initialisation* qui à chaque place associe un marquage initial tel que, pour toute place $p \in P$, $\sigma(I(p)) = C(p)$.

Pour modéliser et exécuter les réseaux de Petri colorés, nous utilisons le logiciel CPN Tools [1] [RWL+03]. La Figure 3.1 présente l'apparence d'un réseau dans l'interface de CPN Tools. Pour simplifier la lecture et la conception de réseaux, CPN Tools dispose d'un système d'*annotation de fusion* (« *fusion tags* ») : deux places possédant la même annotation de fusion [2] doivent être assimilées à une unique place, et possèdent par conséquent le même marquage. Ce système sert notamment à la composition de réseaux, ce que nous exploitons pour la construction inductive de nos réseaux. Il offre également la possibilité d'effectuer des simulations et des analyses d'espace d'états.

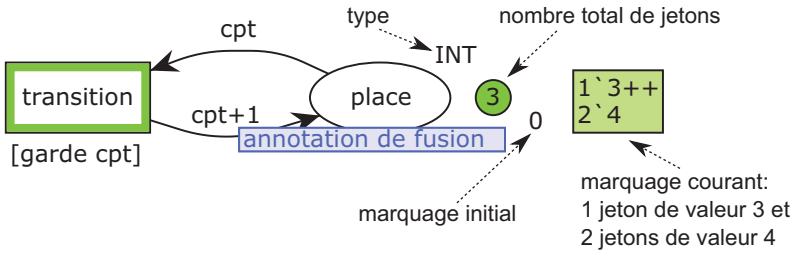


FIGURE 3.1 – Un réseau sur CPN Tools

3.1.3 La fusion de places

Nous modélisons le processus de reconnaissance de chroniques en élaborant des réseaux de Petri colorés associés aux chroniques, et ce de façon modulaire : cette construction se fait par induction sur la structure de la chronique, à l'aide de fusion de places. Il s'agit maintenant de définir formellement ce mécanisme de fusion.

Il existe deux types de fusion pour les réseaux de Petri colorés : la fusion de places, et la fusion de transitions. Cette dernière consiste en l'unification de plusieurs transitions en une unique transition regroupant les arcs d'entrée et de sortie de ces transitions, et dont la garde est la conjonction de l'ensemble des gardes des transitions fusionnées. La transition résultante n'est donc tirable que lorsque l'ensemble des transitions fusionnées le sont. Cette caractéristique n'est pas désirable pour notre modèle et nous n'utilisons donc pas cette fonctionnalité.

La fusion de places se comprend premièrement intuitivement comme une facilité d'écriture : les places fusionnées partagent systématiquement le même marquage initial, le même type, et le même

1. `http://cpntools.org/`
2. Selon le formalisme défini par la suite en 3.1.3, ces deux places appartiennent donc à un même ensemble de fusion de places.

marquage courant, et il s'agit donc du dédoublement d'une unique place facilitant la représentation des arcs d'entrée et de sortie, évitant ainsi l'éventuel « plat de spaghetti ». Cependant, nous verrons que la fusion de places peut également poser des problèmes de sémantique.

Une définition formelle de la fusion est proposée dans [CP92] (Definition 4.3) où sont définis des réseaux de Petri modulaires – Modular Coloured Petri Nets (MCPNs). Ce sont des triplets (S, PF, TF) où :

(i) S est un ensemble fini de modules qui sont des réseaux de Petri disjoints ;

(ii) PF est un ensemble fini d'ensembles de fusion de places vérifiant certaines conditions ;

(iii) TF est un ensemble fini d'ensembles de fusion de transitions.

Nous nous inspirons largement de cette définition, mais nous avons besoin d'y apporter quelques modifications.

D'une part, dans cette définition, un réseau de Petri modulaire est construit à partir d'un ensemble de réseaux de Petri *non modulaires* S dont on souhaite fusionner certaines places et certaines transitions. Cette définition ne convient pas directement à notre construction. En effet, elle est définie par induction, et des fusions *successives* sont donc effectuées. La fusion doit donc se faire à partir d'un ensemble de réseaux de Petri eux-même *modulaires*. Dans [CP92] il est ensuite montré (Théorème 4.9) que, pour tout MCPN, on peut aisément construire un réseau de Petri coloré non modulaire équivalent, ce qui résout notre problème. Cependant, afin de nous affranchir de conversions incessantes, nous établissons dans cette section une définition directe de la fusion de places à partir de MCPN plutôt qu'à partir de réseaux de Petri colorés simples.

D'autre part, un pré-requis sur l'ensemble PF est que toutes les places d'un de ses ensembles de fusions de places pf (c'est-à-dire toutes les places à fusionner ensemble) doivent avoir le même type et le même marquage initial. C'est une condition que l'on retrouve également dans un article de K. Jensen [HJS91]. Cependant, dans le logiciel que nous utilisons, CPN Tools, il est possible de fusionner deux places ayant des types et marquages initiaux différents : c'est la première place sélectionnée qui définit les caractéristiques de la seconde. Nous n'autoriserons pas la fusion de places de types différents car cela pourrait entraîner la construction d'un réseau mal formé, même si les réseaux de départ son bien formés. En revanche, la fusion de places de marquages initiaux différents offre des possibilités intéressantes en permettant d'adapter le marquage initial d'une place selon qu'elle soit fusionnée ou non. Nous aurons besoin d'exploiter cette possibilité, donc nous l'intégrons dans notre définition de la fusion.

Enfin, comme nous ne mettons pas en œuvre de fusion de transitions, nous ne faisons pas apparaître l'ensemble TF.

On obtient ainsi les définitions suivantes, en commençant par poser la notion de MCPN puis en formalisant la fusion.

Définition 23 (réseau de Petri coloré modulaire). Un *réseau de Petri coloré modulaire* (MCPN) est un couple (S, PF) vérifiant les propriétés suivantes :

(i) S est un ensemble fini de *modules* tel que :
— Chaque module $s \in S$ est un réseau de Petri coloré $(P_s, T_s, A_s, \mathcal{B}_s, V_s, C_s, G_s, EX_s, I_s)$.

— Les ensembles des éléments des réseaux sont deux-à-deux disjoints :
$\forall s_1 \in S\ \forall s_2 \in S\ \ [s_1 \neq s_2 \Rightarrow (P_{s_1} \cup T_{s_1}) \cap (P_{s_2} \cup T_{s_2}) = \emptyset]$
Notons que l'on peut renommer les places et les transitions de manière unique en les préfixant par un identifiant de réseau. On pourra donc supposer que les réseaux ont des places et des transitions distinctes sans mettre en œuvre cette préfixation qui conduit à des notations excessivement lourdes.

(ii) $PF \subseteq P \times \mathcal{P}(P)$ est un ensemble fini de *couples de fusion de places*, dont on appellera *place d'initialisation de la fusion* le premier membre, ainsi qu'*ensemble de fusion de places* le second, où $P = \bigcup_{s \in S} P_s$ et $\mathcal{P}(P)$ désigne l'ensemble des parties de P, et tel que :

— Toute place d'initialisation appartient à l'ensemble de fusion de places associé.
$\forall (p_0, E) \in PF\ p_0 \in E$

— Toutes les places d'un ensemble de fusion de places ont même type.
$\forall (p_0, E) \in PF\ \ \forall p_1 \in E\ \forall p_2 \in E\ \ C(p_1) = C(p_2)$

— L'ensemble des ensembles de fusion de places forme une partition de P [3].
$[\ \forall p \in P\ \ \exists p_0 \in P\ \exists E \in \mathcal{P}(P)\ \ ((p_0, E) \in PF \wedge p \in E)\ ]$
$\wedge\ [\ \forall p_{0_1} \in P\ \forall p_{0_2} \in P\ \forall E_1 \subseteq P\ \forall E_2 \subseteq P$
$(\ (p_{0_1}, E_1) \in PF \wedge (p_{0_2}, E_2) \in PF \wedge E_1 \neq E_2\ ) \Rightarrow E_1 \cap E_2 = \emptyset\ ]$

Remarque 17. Notons qu'à tout réseau de Petri coloré N correspond trivialement un MCPN équivalent : il suffit de poser $M = \{S, PF\}$ où $S = \{N\}$ et $PF = \{(p, \{p\}) : p \in P_N\}$.

On définit maintenant une fonction de fusion entre réseaux de Petri modulaires. C'est celle-ci que nous appliquons lors de l'induction pour construire nos réseaux de Petri associés aux chroniques.

Définition 24 (fusion de places). Soit $M = \{(S_1, PF_1), \ldots, (S_n, PF_n)\}$ un ensemble fini de MCPN et soit PF_0 un ensemble fini de couples de fusion de places tels que :

— Les places d'initialisation de fusion des couples de PF_0 sont des places d'initialisation de couples des $PF_1, \ldots, PF_n$:
$\forall p_0 \in P\ \forall E_0 \subseteq P\ [\ (p_0, E_0) \in PF_0 \Rightarrow (\ \exists i \in [\![1, n]\!]\ \exists E_i \subseteq P\ (p_0, E_i) \in PF_i\ )\ ]$

— Les ensembles de fusion de places des couples de PF_0 sont formés de places des MCPN de M :
$\forall p_0 \in P\ \forall E_0 \subseteq P\ [\ (p_0, E_0) \in PF_0 \Rightarrow (\ \forall p \in E_0\ \exists i \in [\![1, n]\!]\ \exists E_i \subseteq P\ ((p_0, E_i) \in PF_i \wedge p \in E_i)\ )\ ]$

Alors on peut définir la *fusion* de ces MCPN *relativement à l'ensemble* PF_0 :
$Fusion(M, PF_0) = (S, PF)$ où :

— $S = \bigcup_{1 \leq i \leq n} S_i$

— $PF = \{(p_0, \bigcup_{p \in E} E(p)) : (p_0, E) \in PF_0\}$ où $E(p)$ est l'unique E_i tel que $(p, E_i) \in PF_i$.

3. Notons que la contrainte que les ensembles de fusion de places soient deux à deux disjoints n'était pas requise dans [CP92] mais découle de notre construction qui autorise la fusion de places à marquages initiaux différents. En effet, si une place appartenait à deux ensembles de fusion de places distincts, deux marquages initiaux différents pourraient lui être associés, ce qui est absurde.

On peut montrer, à l'aide d'une démonstration analogue à celle exposée dans [CP92], que le réseau de Petri suivant ainsi que le MCPN à partir duquel il est construit sont équivalents du point de vue de leur comportement (évolution du marquage, transitions tirables, places atteignables...).

Définition 25 (réseau de Petri coloré équivalent). Pour toute place d'initialisation p_0, $E(p_0)$ désigne l'ensemble de fusion de places associé, et pour toute place p et toute transition t, $s(p)$ et $s(t)$ désignent respectivement le réseau de Petri de S tel que $p \in P_{s(p)}$ et celui tel que $t \in T_{s(t)}$.

Soit $M = (S, PF)$ un réseau de Petri coloré modulaire. On définit le réseau de Petri coloré équivalent par $M^* = (P^*, T^*, A^*, \mathcal{B}^*, V^*, C^*, G^*, EX^*, I^*)$ où :

(i) $P^* = \{p_0 \in P : \exists E_0 \in \mathcal{P}(P)\ (p_0, E_0) \in PF\}$ (on rappelle que $P = \bigcup_{s \in S} P_s$)

(ii) $T^* = \bigcup_{s \in S} T_s$

(iii) $A^* = \{(p_0, t) \in P^* \times T^* : \exists p \in E(p_0)\ (p, t) \in A_{s(p)}\}$
$\cup \{(t, p_0) \in T^* \times P^* : \exists p \in E(p_0)\ (t, p) \in A_{s(p)}\}$

(iv) $\mathcal{B}^* = \bigcup_{s \in S} \mathcal{B}_s$

(v) $V^* = \bigcup_{s \in S} V_s$

(vi) $\mathcal{C}^* : P^* \to \mathcal{B}^*, p_0 \mapsto C_{s(p_0)}(p_0)$

(vii) $G^* : T^* \to \mathrm{Expr}_{\Sigma(V^*)}, t \mapsto G_{s(t)}(t)$

(viii) $EX^* : A^* \to \mathrm{Expr}_{\Sigma(V^*)}$ est défini par :
— pour tous (p_0, t) tels que $\exists p \in E(p_0)\ (p, t) \in A_{s(p)}$, $(p_0, t) \mapsto EX_{s(p)}((p, t))$,
— pour tous (t, p_0) tels que $\exists p \in E(p_0)\ (t, p) \in A_{s(p)}$, $(t, p_0) \mapsto EX_{s(p)}((t, p))$.

(ix) $I^* : P^* \to \mathrm{Expr}_{\Sigma(V^*)},\ p_0 \mapsto I_{s(p_0)}(p_0)$

3.1.4 Arcs inhibiteurs

Dans le Chapitre 4, nous construisons des réseaux de Petri plus complexes où il faut pouvoir spécifier qu'une transition n'est tirable que si une place donnée est *vide*. Cette contrainte ne peut pas s'exprimer par une simple garde sur la transition. On introduit un nouveau type d'arc, l'*arc inhibiteur*, qui permet l'implémentation d'algorithmes fondés sur l'absence ou non de jetons dans une place. Des réseaux équivalents peuvent être construits sans arcs inhibiteurs à l'aide de listes mais cela conduit à des réseaux trop complexes.

On complète donc la Définition 22 des réseaux de Petri colorés comme suit :

Définition 26 (arcs inhibiteurs). Un réseau de Petri coloré muni d'arcs inhibiteurs est un 10-uplet $(P, T, A, \mathcal{B}, V, C, G, EX, I, B)$ où $(P, T, A, \mathcal{B}, V, C, G, EX, I)$ est un réseau de Petri coloré et :

10. $B \subseteq P \times T$ est un ensemble d'*arcs inhibiteurs* orientés.

Un arc inhibiteur n'a d'incidence que sur le tirage des transitions. Notons $Inh(t) = \{p \in P : (p, t) \in B\}$ l'ensemble des places reliées par un arc inhibiteur à une transition $t \in T$. La contrainte suivante s'ajoute pour qu'une transition $t \in T$ soit tirable :

(iii) aucune place inhibitrice $p \in Inh(t)$ ne contient de jeton.

Dans le logiciel CPN Tools, les arcs inhibiteurs sont représentés comme sur la Figure 3.2. Lorsque la place est vide, la transition est tirable, ce qui est indiqué par le liseré vert.

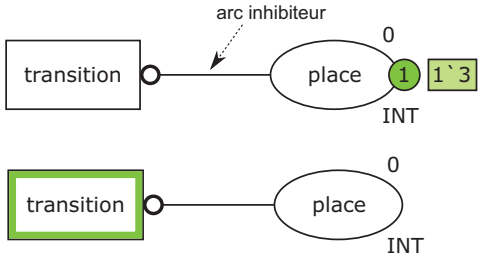


FIGURE 3.2 – Arcs inhibiteurs sur CPN Tools

3.2 Construction formelle des réseaux dits « à un seul jeton »

À l'aide du formalisme des réseaux de Petri colorés munis du mécanisme de fusion décrit dans 3.1, nous construisons par induction un modèle du processus de reconnaissance. Pour chaque chronique C, nous définissons un réseau de Petri coloré associé dont le marquage évolue en fonction du flux d'évènements. Une place du réseau contient l'ensemble des reconnaissances correspondant à la chronique étudiée.

Nous commençons par établir dans la Section 3.2.1 les types des données et fonctions utilisés. Nous donnons ensuite un aperçu de la structure générale des réseaux que nous allons construire (Section 3.2.2). Nous posons dans la Section 3.2.3 un ensemble élémentaire de réseaux de Petri colorés, les briques de base, que nous utilisons pour construire nos réseaux lors d'une induction sur la structure de la chronique dans la Section 3.2.4.

3.2.1 Types et expressions utilisés dans le modèle

Posons maintenant les types et fonctions que nous utilisons dans nos réseaux de Petri en définissant $\Sigma = (\mathcal{B}, \mathcal{F}, \sigma)$ (Définition 20).

Posons $\mathcal{B} = \{\texttt{INT}, \texttt{Event}, \texttt{NCList}\}$, et définissons le type $\texttt{NEvent} = \texttt{Event} \times \texttt{INT}$.

Le type `INT` correspond à un entier, `Event`, à un évènement, et `NCList`, à une liste de listes de couples `NEvent`. Un jeton de type `NCList` correspond à un ensemble de reconnaissances. Chaque élément de la liste `NCList` est une liste de `NEvent`, c'est-à-dire une liste d'évènements datés qui représente une reconnaissance.

Posons $\mathcal{F} = \{+, \max, \text{ANR}, \text{CPR}, \text{rem\_obs}, \text{mixAnd}, \text{last\_ini\_recog}\}$ qui correspond à l'ensemble des fonctions employées dans nos réseaux.

Le Tableau 3.3 décrit brièvement le fonctionnement général de ces fonctions. Une explication axée sur notre utilisation de ces fonctions sera donnée lors de la description de l'exécution des réseaux dans la Section 3.3.

À partir des fonctions de $\mathcal{F}$, nous pouvons alors définir une dernière fonction qui permet de compléter des reconnaissances dans une liste avec une instance d'un évènement :

$$\text{complete} : ((E(a), cpt), curr, inst) \mapsto \text{ANR}(inst, \text{CPR}([[(E(a), cpt + 1)]], curr)$$

Tableau 3.3 – Fonctions utilisées dans nos réseaux

Fonction	Type (image de la fonction σ)	Description
+	`INT` × `INT` → `INT`	Addition usuelle.
max	`INT` × `INT` → `INT`	Entier maximum entre deux entiers.
ANR	`NCList` × `NCList` → `NCList`	*(Add New Recognition)* Ajoute le contenu de la seconde liste à la première liste, *s'il n'y est pas encore.*
CPR	`NEvent` × `NCList` → `NCList`	*(Complete Partial Recognition)* Renvoie la liste de listes de couples dont chaque liste de couples a été complétée par le couple supplémentaire $(E(a), cpt + 1)$.
rem_obs	`INT` × `NCList` → `NCList`	Renvoie la liste de laquelle on a d'abord supprimé toutes les listes de couples dont un des couples $(E(a), cpt)$ a un indice cpt inférieur à l'entier n qui est en argument.
mergeAndNew	`NCList` × `NCList` × `NCList` × `NCList` → `NCList`	Renvoie la dernière liste à laquelle a été ajoutée une combinaison explicitée par la suite des trois premières listes.
startsub	`NCList` → `NCList`	Renvoie [[ ]] si l'argument est une liste non vide, [ ] sinon.
chg_win	`INT` × `NCList` → `INT`	Renvoie le maximum entre l'entier en argument et le plus grand instant de reconnaissance des reconnaissances de la liste.
concatabs	`NCList` × `NCList` × `NCList` × `INT` → `NCList`	Effectue les combinaisons séquentielles possibles entre les deux premières listes, puis les ajoute à la troisième liste si elles sont nouvelles (le critère de nouveauté étant déterminé par une comparaison avec l'entier).

3.2.2 Structure générale des réseaux « à un seul jeton »

Nous construisons donc inductivement sur la structure du langage un réseau de Petri coloré pour chaque chronique. Il permet de calculer, en fonction d'un flux d'évènements, l'ensemble de reconnaissances associé.

Pour pouvoir réaliser cette construction *inductive*, il faut que les réseaux soient modulaires. Cette caractéristique se traduit par le fait que tous les réseaux associés aux chroniques ont une structure générale identique permettant ainsi de les combiner. Cette structure est présentée dans la Figure 3.4.

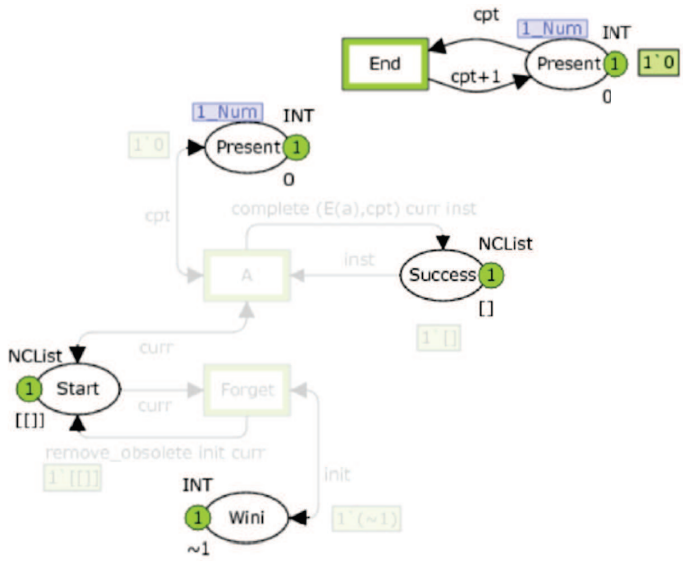


FIGURE 3.4 – Structure des réseaux

Chaque réseau possède un compteur d'évènements (place `Present` et transition `End`) ainsi que quatre places principales :

- la place `Present`, de type `INT`, est fusionnée avec le compteur d'évènements et contient un entier correspondant au nombre d'évènements déjà traités ;
- la place `Start` est de type `NCList`, elle contient une liste de reconnaissances qui seront complétées par le réseau ;
- la place `Success` est également de type `NCList`, elle contient les reconnaissances de la place `Start` complétées par le réseau ;
- la place `Wini`, de type `INT`, contient un entier qui sert de repère dans le cas d'une absence, permettant de déterminer les reconnaissances qu'il faut alors supprimer de la place `Start`.

Chaque place contient exactement un seul jeton d'où le nom de modèle « à un seul jeton ». Les places `Start` et `Success` sont marquées d'un jeton contenant une liste des reconnaissances. Les reconnaissances vont circuler dans les réseaux au fur et à mesure de l'évolution du flux d'évènements. Pour une chronique complexe, une reconnaissance est d'abord vide [ ] dans la place `Start`

du réseau général, puis elle va petit à petit être complétée par les évènements pertinents du flux. Au fur et à mesure qu'elle est complétée, elle va transiter de la place `Start` à la place `Success` des sous-réseaux, passant de sous-réseau en sous réseau pour arriver, lorsque la reconnaissance est complète, à la place `Success` du réseau général. Les réseaux « à un seul jeton » se lisent de gauche à droite, ce qui correspond au circuit des reconnaissances.

Ce sont ces quatre places principales que l'on fusionne tour à tour entre plusieurs réseaux et avec les briques élémentaires que nous allons définir dans la Section 3.2.3. Pour la gestion correcte d'absences dans une séquence (ce qui est détaillé dans la Section 3.3.5), les places `Wini` sont en fait divisées en deux catégories : les places `WiniIn` et les places `WiniOut` qui permettent de fusionner correctement les places pour construire nos réseaux. Lors de la construction des réseaux, nous définissons donc cinq fonctions qui à chaque chronique C associent les cinq types de place principaux `Present`(C), `Start`(C), `Success`(C), `WiniIn`(C) et `WiniOut`(C) du réseau correspondant à la chronique C.

3.2.3 Briques de base

Définissons maintenant les quelques réseaux élémentaires que nous utilisons comme briques de base dans la construction inductive de notre modèle de reconnaissance.

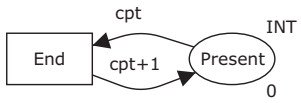


Figure 3.5 – Compteur d'évènements

Compteur Ce réseau élémentaire, noté `CPT` et présenté dans la Figure 3.5, fait office de compteur d'évènements. Il est composé d'une place `Present` dans laquelle est stockée la valeur de ce compteur, et d'une transition `End` qui incrémente le compteur à chaque fois qu'elle est tirée.

Opérateur `AND` Ce réseau élémentaire, noté $\texttt{OP}_{\texttt{AND}}$ et présenté dans la Figure 3.6, calcule l'ensemble des reconnaissances de la conjonction de deux chroniques C_1 et C_2. On donne ici une description très succincte du fonctionnement de cet opérateur. La Section 3.3.4 détaille davantage le déroulement d'une reconnaissance de conjonction.

Le réseau est composé de six places. Dans deux places, `Operand1` et `Operand2`, sont stockées les reconnaissances respectives de C_1 et de C_2. Lorsque la transition `AND` est tirée, les reconnaissances de C_1 et celles de C_2 sont combinées de façon à récupérer dans la place `Success` les reconnaissances de $C_1 \& C_2$.

Si la conjonction forme la seconde partie d'une séquence, c'est-à-dire si la chronique étudiée est de la forme $\cdots (C_3\ C_1 \& C_2) \cdots$, nous ne souhaitons pas que le réseau commence à reconnaitre C_1 et C_2 tant que la première partie de la séquence (à savoir C_3) n'a pas été reconnue. La transition `Sub`

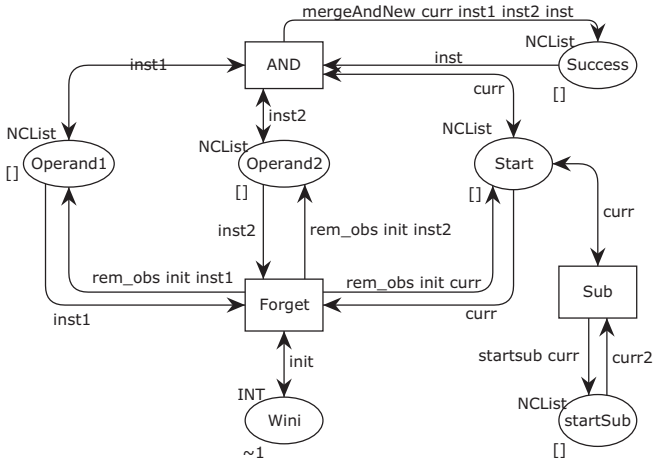


FIGURE 3.6 – Opérateur `AND`

sert à initialiser les places `Start` de C_1 et de C_2 afin de contrôler la mise en route du mécanisme de reconnaissance de C_1 et de C_2.

La transition `Forget` sert dans le cas de l'absence.

La place `Success` de ce réseau joue un rôle central car elle stocke les reconnaissances de $C_1 \& C_2$. Nous aurons donc besoin de nous y référer pour effectuer des fusions. Pour cela, nous utiliserons `Success(AND)` pour `Success`, de même que `Start(AND)` pour `Start` et `Wini(AND)` pour `Wini`.

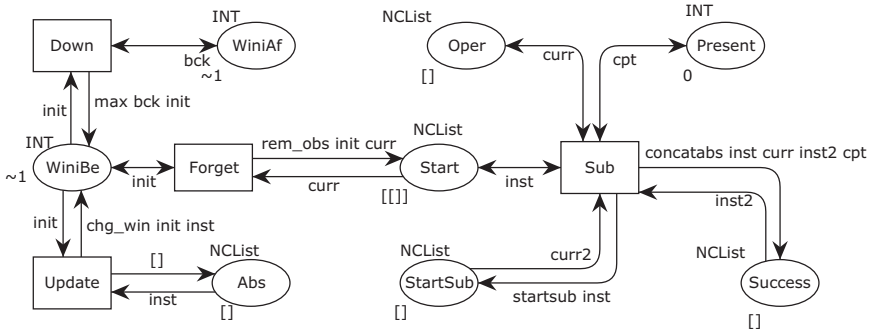


FIGURE 3.7 – Opérateur `ABS`

Opérateur `ABS` Ce réseau élémentaire, noté $\texttt{OP}_{\texttt{ABS}}$ et présenté dans la Figure 3.7, sert à composer deux réseaux de Petri correspondant aux chroniques C_1 et C_2 pour obtenir les reconnaissances de $(C_1) - [C_2[$. Le rôle de ce réseau est double :

— La partie de gauche du réseau est chargée d'assurer la gestion de l'entier repère stocké dans les places `Wini` du reste du réseau. Rappelons que cet entier permet d'établir si certaines

reconnaissances doivent être supprimées car invalidées par une absence. La place `WiniBe` stocke l'indice suivant celui de la dernière reconnaissance de C_2 et est mise à jour par le tirage de la transition `Update`. Cet indice sert à supprimer les reconnaissances partielles de C_1 qui ne doivent pas être complétées car C_2 a été reconnue. La place `WiniAf` sert dans le cas où $(C_1) - [C_2[$ est elle-même imbriquée dans une autre absence : elle propage grâce à la transition `Down` la valeur du `Wini` de la seconde absence, mais la valeur de `Wini` de la première absence (celle dans `WiniBe`) n'est en revanche pas propagée à l'extérieur de celle-ci.

— La partie de droite doit son origine au caractère modulaire de nos réseaux. Elle est chargée de recombiner les reconnaissances de l'absence (qui sont stockées dans la place `Oper`) avec les reconnaissances pouvant les précéder (qui sont dans la place `Start`). Ce genre de combinaison est nécessaire lorsque la chronique étudiée contient une absence mais à un niveau de profondeur non nul, par exemple lorsqu'une absence est composée avec une séquence (comme $D\ ((A\ B) - [C[)$). Il s'agit d'une combinaison analogue à celle effectuée par la fonction mergeAndNew pour l'opérateur $\texttt{OP}_{\texttt{AND}}$ mais avec une contrainte temporelle supplémentaire : la combinaison doit être séquentielle. Il n'est pas possible de faire transiter les reconnaissances précédant l'absence à travers le réseau d'absence car la portée de l'absence doit être délimitée. La brique `ABS` permet donc de marquer les bornes de l'absence et d'isoler les reconnaissances jusqu'à ce qu'elles soient prêtes à être combinées. Comme dans l'opérateur `AND`, la place `StartSub` sert à activer le réseau de l'absence. En effet, lorsque l'on cherche à reconnaître $D\ ((A\ B) - [C[)$ par exemple, on ne souhaite pas commencer à reconnaître $(A\ B) - [C[$ tant qu'il n'y a pas de reconnaissance de D à compléter. La transition `Sub` permet donc à la fois de mettre à jour la liste des reconnaissances globales de la place `Success`, et d'activer le réseau de l'absence si nécessaire.

Une description plus élaborée du mécanisme de l'absence et des nombreuses problématiques qui lui sont associées est donnée dans la Section 3.3.5.

3.2.4 Construction par induction

Avec les types et expressions définis dans la Section 3.2.1 et les réseaux élémentaires de la Section 3.2.3, nous pouvons maintenant construire notre modèle en réseau de Petri colorés du processus de reconnaissance. Pour chaque chronique C, nous définissons par induction un réseau de Petri coloré $N(C)$ qui calcule les reconnaissances de C. Chaque réseau résulte de la fusion d'un compteur d'évènements et de sous-réseaux. La construction par induction se fait donc en deux étapes. Pour une chronique C, nous définissons d'abord une réseau $N'(C)$ qui correspond au mécanisme global de reconnaissance sans le compteur, puis nous réalisons une fusion de $N'(C)$ avec le compteur pour définir $N(C)$. Ce sont les réseaux $N'(C)$ qui sont utilisés comme sous-réseaux dans l'induction comme nous le verrons par la suite.

Comme évoqué dans la Section 3.2.2, dans la construction par induction, certaines des places `Present`, `Start`, `Success` et `Wini` jouent un rôle dans la composition des réseaux. En effet, quelle que soit la chronique C, chaque réseau $N(C)$ a la même structure globale que nous avons présentée dans la Figure 3.4. Formellement, nous définissons en parallèle de $N(C)$ les places $\texttt{Present}(C)$, $\texttt{Start}(C)$, $\texttt{Success}(C)$, $\texttt{WiniIn}(C)$ et $\texttt{WiniOut}(C)$ qui délimitent cette structure et qui sont donc

utilisées si l'on compose le réseau avec un autre réseau.

Dans cette section, nous présentons la construction formelle de nos réseaux de reconnaissance de chroniques en expliquant les fusions effectuées et le mécanisme global de chaque réseau. Dans les Sections 3.3.1 à 3.3.5, nous donnons ensuite une explication plus détaillée du mécanisme de chaque réseau.

Si $C = A \in \mathfrak{N}$

Dans le cas d'un évènement simple, ce réseau élémentaire utilisé pour sa reconnaissance correspond exactement à $N'(C)$.

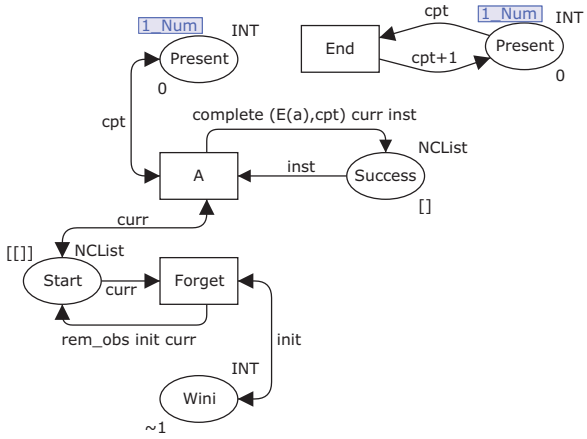


FIGURE 3.8 – Réseau correspondant à la chronique A

La Figure 3.8 représente le réseau $N(A)$ relatif à l'évènement simple A.

Comme évoqué dans la description de la structure générale des réseaux (Section 3.2.2), la place `Start` contient les reconnaissances devant être complétées par le réseau. Dans l'exemple présenté ici, il s'agit uniquement de reconnaître A. Le marquage de la place `Start` est donc [[ ]] : la reconnaissance à compléter est la reconnaissance vide [ ] qui pourra évoluer en une reconnaissance de A en transitant dans le réseau.

La place `Wini` et la transition `Forget` sont utilisées si le réseau est fusionné pour construire une chronique plus complexe incluant une absence.

On définit la structure du réseau :

$$\begin{aligned} \texttt{Present}(C) &= \texttt{Present} \\ \texttt{Start}(C) &= \texttt{Start} \\ \texttt{Success}(C) &= \texttt{Success} \\ \texttt{WiniOut}(C) &= \texttt{Wini} \\ \texttt{WiniIn}(C) &= \emptyset^{\,4} \end{aligned}$$

Puis on pose [5] :

$$N(C) = \text{Fusion}(\{N'(C), \texttt{CPT}\}, \{(\texttt{Present}(C), \{\texttt{Present}(C), \texttt{Present}(\texttt{CPT})\})\}) \tag{3.1}$$

Si $C = C_1\ C_2$

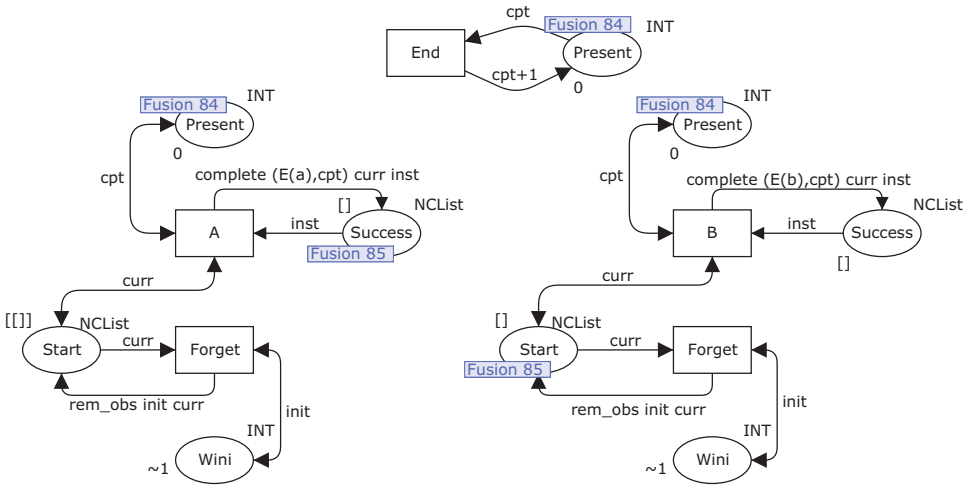


Figure 3.9 – Réseau correspondant à la chronique $A\ B$

Afin de modéliser une séquence $C_1\ C_2$ (comme la séquence $A\ B$ dont le réseau est représenté Figure 3.9), nous fusionnons la place `Success` du réseau $N(C_1)$ avec la place `Start` du réseau $N(C_2)$. Le marquage initial de la place $\texttt{Start}(C_2)$ change donc : il prend le marquage initial de la place $\texttt{Success}(C_1)$, c'est-à-dire [ ]. Ainsi, le réseau ne commence pas à reconnaître C_2 tant que ce n'est pas pour compléter une reconnaissance de C_1.

Remarque 18 (marquages initiaux des places `Start`). Dans nos réseaux, il y a deux marquages initiaux possibles pour une place `Start` :

- la liste qui contient la liste vide, [[ ]], indique que le réseau peut activer son mécanisme et compléter la reconnaissance partielle vide [ ] – on dit alors que le réseau est *activé* ;
- la liste vide [ ] indique qu'il n'y a encore aucune reconnaissance partielle à compléter, et tant que le marquage n'est pas modifié, le mécanisme du réseau ne peut pas opérer et on dit qu'il n'est *pas activé*.

Les marquages initiaux des places `Start` permettent donc de contrôler précisément l'activation des différentes parties d'un réseau, pour ne pas activer la reconnaissance d'un évènement tant que

4. Ceci signifie qu'il n'y a pas de place `WiniIn` dans les réseaux d'évènement simple.
5. Par la suite, nous effectuons des fusions de réseaux de Petri colorés et de MCPN. Du fait de la Remarque 17, nous ne prenons pas la peine de redéfinir un MCPN équivalent pour les quelques réseaux de Petri non modulaires mis en jeu.

ce n'est pas nécessaire. Par exemple, dans le cas de la séquence $C_1\ C_2$, nous avons vu qu'il n'est pas nécessaire de commencer à reconnaître les évènements relatifs à C_2 tant qu'une reconnaissance complète de C_1 n'est pas disponible pour être complétée.

Les différents marquages initiaux sont rendus possibles par la fonctionnalité de fusion évoquée dans la Section 3.1.3 qui détermine le marquage initial des places fusionnées.

On pose :

$$\begin{aligned} N'(C) = \ & \text{Fusion}(\{N'(C_1), N'(C_2)\}, \\ & \{\ (\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2)\}), \\ & \ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Start}(C_2)\}), \\ & \ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2), \texttt{WiniOut}(C_2)\})\ \}) \end{aligned}$$

On définit la structure du réseau :

$$\begin{aligned} \texttt{Present}(C) &= \texttt{Present}(C_1) \\ \texttt{Start}(C) &= \texttt{Start}(C_1) \\ \texttt{Success}(C) &= \texttt{Success}(C_2) \\ \texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1) \\ \texttt{WiniIn}(C) &= \texttt{WiniOut}(C_2) \end{aligned}$$

Puis nous fusionnons avec le compteur d'évènements comme dans l'équation (3.1).

Sur la Figure 3.9, on remarque que les places $\texttt{Present}(A)$, $\texttt{Present}(B)$, et $\texttt{Present}(\texttt{CPT})$ ont la même annotation de fusion, à savoir `Fusion_84`. De même pour $\texttt{Success}(A)$ et $\texttt{Start}(B)$ qui appartiennent à un même ensemble de fusion annoté `Fusion_85`.

Si $C = C_1 \mid\mid C_2$

Afin de modéliser la disjonction (comme $A \mid\mid B$ dont le réseau est représenté Figure 3.10), les deux réseaux $N'(C_1)$ et $N'(C_2)$ fonctionnent en parallèle. Nous fusionnons donc les places `Start` des deux réseaux et les places `Success` des deux réseaux.

On pose :

$$\begin{aligned} N'(C) = \ & \text{Fusion}(\{N'(C_1), N'(C_2)\}, \\ & \{\ (\texttt{Start}(C_1), \{\texttt{Start}(C_1), \texttt{Start}(C_2)\}), \\ & \ (\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2)\}), \\ & \ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Success}(C_2)\}), \\ & \ (\texttt{WiniOut}(C_1), \{\texttt{WiniOut}(C_1), \texttt{WiniOut}(C_2)\}), \\ & \ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2)\})\ \}) \end{aligned}$$

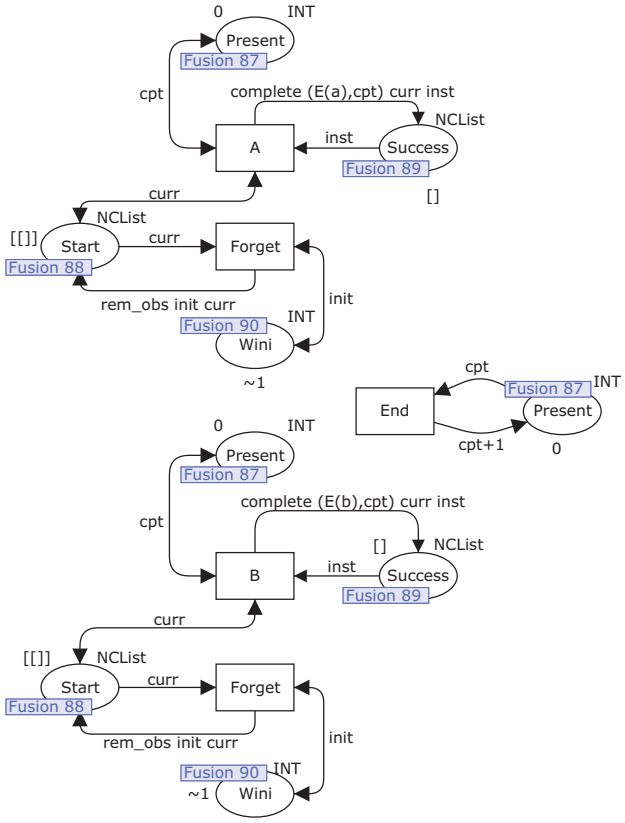


Figure 3.10 – Réseau correspondant à la chronique $A \,||\, B$

On définit la structure du réseau :

$$
\begin{aligned}
\texttt{Present}(C) &= \texttt{Present}(C_1) \\
\texttt{Start}(C) &= \texttt{Start}(C_1) \\
\texttt{Success}(C) &= \texttt{Success}(C_1) \\
\texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1) \\
\texttt{WiniIn}(C) &= \begin{cases} \texttt{WiniIn}(C_1) & \text{si } \texttt{WiniIn}(C_1) \neq \emptyset \\ \texttt{WiniIn}(C_2) & \text{sinon} \end{cases}
\end{aligned}
$$

Puis nous fusionnons avec le compteur d'évènements comme dans l'équation (3.1).

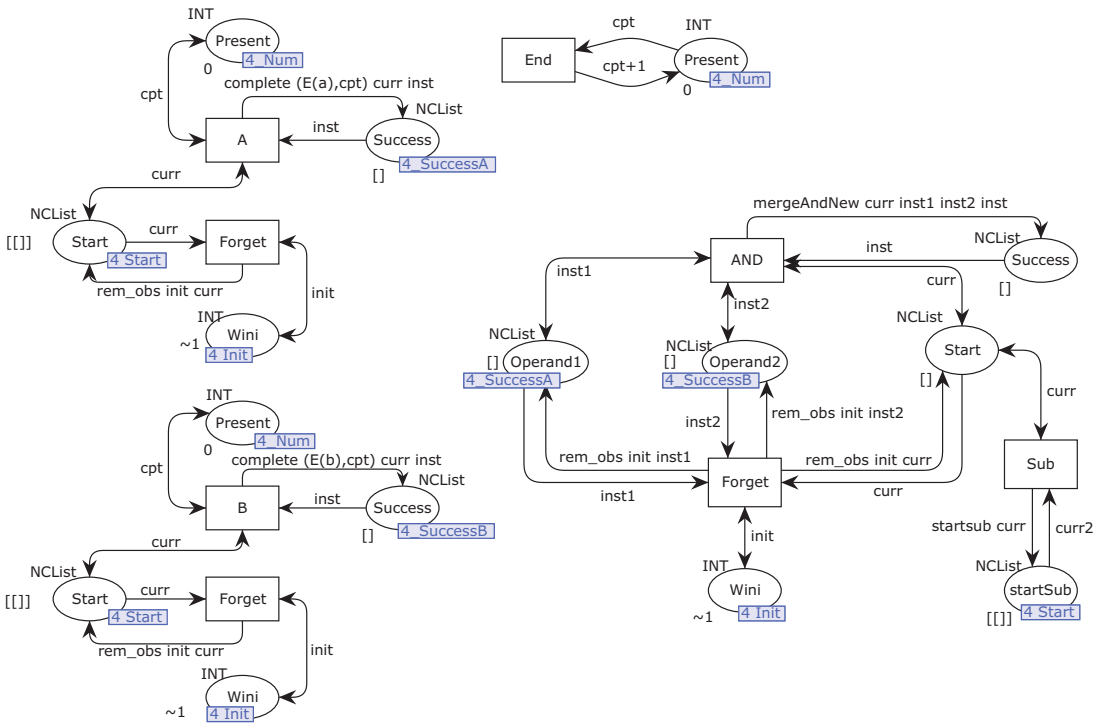


FIGURE 3.11 – Réseau correspondant à la chronique $A\&B$

Si $C = C_1 \& C_2$

Pour modéliser une conjonction $C_1\&C_2$ (comme $A\&B$ dont le réseau est représenté Figure 3.11), les réseaux $N'(C_1)$ et $N'(C_2)$ fonctionnent aussi en parallèle, donc nous fusionnons les places `Start` des deux réseaux. Nous fusionnons les places `Success` des deux réseaux avec des places de l'opérateur $\texttt{OP}_{\texttt{AND}}$ afin de construire les reconnaissances de $C_1\&C_2$.

On pose :

$$
\begin{aligned}
N'(C) = \ & \text{Fusion}(\{N'(C_1), N'(C_2), \texttt{OP}_{\texttt{AND}}\}, \\
& \{\ (\texttt{Start}(C_1), \{\texttt{Start}(C_1), \texttt{Start}(C_2)\}), \\
& \ (\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2)\}), \\
& \ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Operand1}\}), \\
& \ (\texttt{Operand2}, \{\texttt{Operand2}, \texttt{Success}(C_2)\}), \\
& \ (\texttt{WiniOut}(C_1), \{\texttt{WiniOut}(C_1), \texttt{WiniOut}(C_2), \texttt{Wini(AND)}\}, \\
& \ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2)\})\ \})
\end{aligned}
$$

On définit la structure du réseau :

$$
\begin{aligned}
\texttt{Present}(C) &= \texttt{Present}(C_1) \\
\texttt{Start}(C) &= \texttt{Start}(C_1) \\
\texttt{Success}(C) &= \texttt{Success}(\texttt{AND}) \\
\texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1) \\
\texttt{WiniIn}(C) &= \begin{cases} \texttt{WiniIn}(C_1) & \text{si } \texttt{WiniIn}(C_1) \neq \emptyset \\ \texttt{WiniIn}(C_2) & \text{sinon} \end{cases}
\end{aligned}
$$

Puis nous fusionnons avec le compteur d'évènements comme dans l'équation (3.1).

Si $C = (C_1) - [C_2[$

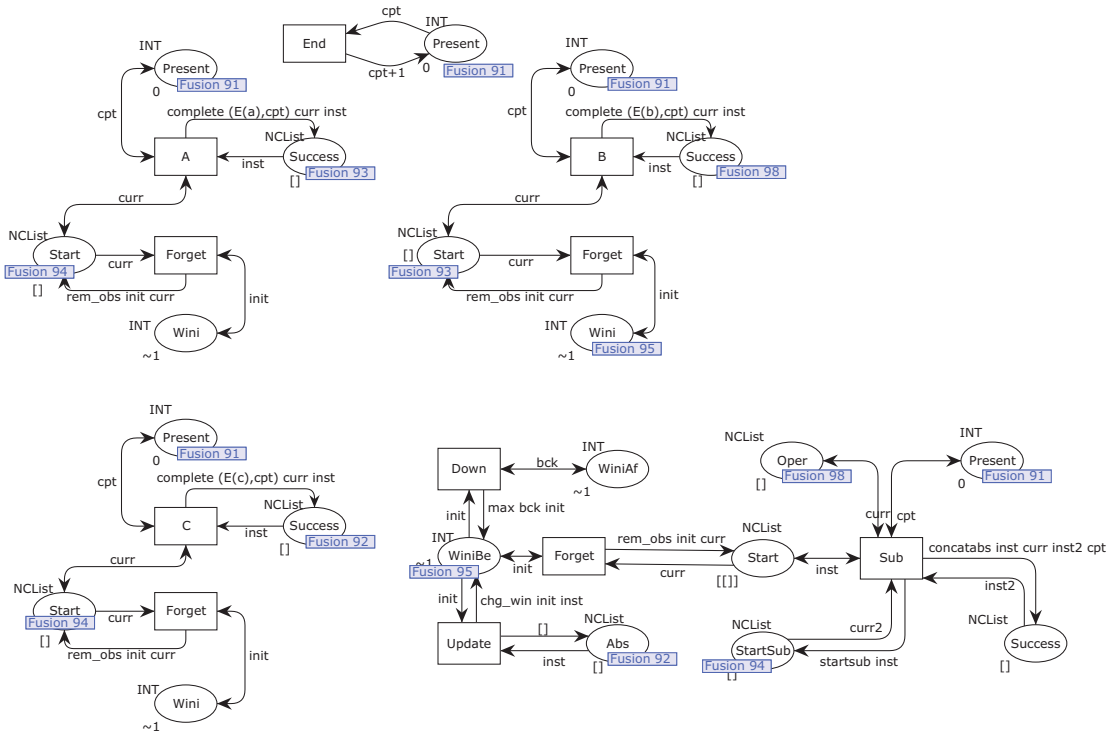


Figure 3.12 – Réseau correspondant à la chronique $(A\,B) - [C[$

Afin de modéliser l'absence $(C_1) - [C_2[$ (comme $(A\,B) - [C[$ dont le réseau est représenté Figure 3.12), nous fusionnons les places `Wini` du réseau $N(C_1)$ et `Success` du réseau $N(C_2)$ avec des places de l'opérateur $\texttt{OP}_{\texttt{ABS}}$ afin de rendre la chronique C_2 « interdite ».

On pose :

$$
\begin{aligned}
N'(C) = \ \text{Fusion}(\{&N'(C_1), N'(C_2), \texttt{OP}_{\texttt{ABS}}\},\\
&\{\ (\texttt{Start}(C_1), \{\texttt{Start}(C_1), \texttt{Start}(C_2), \texttt{StartSub}\}),\\
&\ (\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2), \texttt{Present}(\texttt{ABS})\}),\\
&\ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Oper}\}),\\
&\ (\texttt{Success}(C_2), \{\texttt{Success}(C_2), \texttt{Abs}\}),\\
&\ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniBe}\})\ \})
\end{aligned}
$$

On définit la structure du réseau :

$$
\begin{aligned}
\texttt{Present}(C) &= \texttt{Present}(C_1)\\
\texttt{Start}(C) &= \texttt{Start}(\texttt{ABS})\\
\texttt{Success}(C) &= \texttt{Success}(\texttt{ABS})\\
\texttt{WiniOut}(C) &= \texttt{WiniAf}\\
\texttt{WiniIn}(C) &= \emptyset
\end{aligned}
$$

Puis nous fusionnons avec le compteur d'évènements comme dans l'équation (3.1), ce qui complète la formalisation de la construction de nos réseaux.

3.3 Formalisation et description de l'exécution des réseaux

Dans cette section, nous présentons le fonctionnement des réseaux que nous venons de définir en prenant appui sur des exemples d'exécution. Nous définissons ensuite une stratégie formelle pour le tirage des transitions car toutes les transitions de nos réseaux sont en permanence tirables mais toute séquence de transitions tirée ne mène pas au marquage recherché, à savoir celui où l'on peu correctement lire les ensembles de reconnaissance.

Décrivons maintenant le fonctionnement de ces réseaux. Pour chacune des constructions précédentes, nous présentons l'ensemble de ses places, puis les effets de ses transitions. Nous expliquons ensuite la stratégie de tirage à adopter pour obtenir les ensembles de reconnaissance corrects. Cette stratégie de tirage est formalisée dans la Section 3.3.6.

3.3.1 Reconnaissance d'un évènement simple

Étudions le comportement d'un réseau reconnaissant un évènement simple sur l'exemple de la Figure 3.13 qui correspond à la chronique $A \in \mathfrak{N}$.

Il est composé de deux sous-réseaux : le réseau de compteur d'évènements de la Figure 3.5, et le réseau relatif à la transition `A`. Notons que l'annotation de fusion `1_Num` indique que les deux places `Present` sont fusionnées.

Places Les places `Present` des deux sous-réseaux sont fusionnées, et ont donc le même marquage. Elles sont de type `INT` et contiennent un entier qui correspond à la valeur du compteur d'évènement.

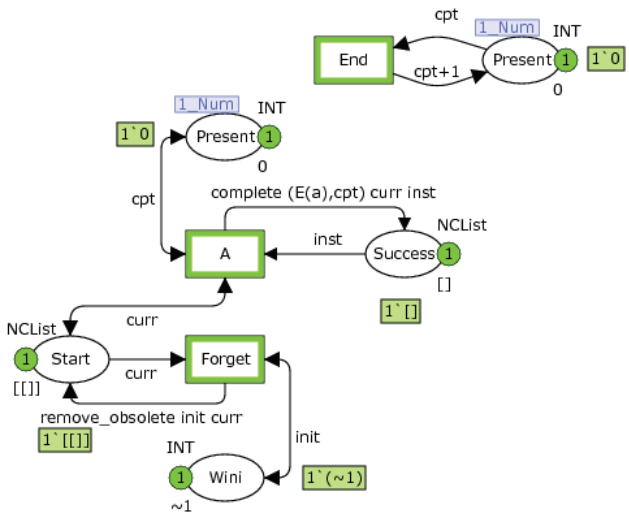


Figure 3.13 – Réseau correspondant à la chronique A

Les places `Start` et `Success` sont de type `NCList`, c'est-à-dire qu'elles contiennent une liste d'instances de chroniques. La place `Start` a un rôle dans la composition des réseaux pour des chroniques complexes. Ici, son marquage est constant, égal à son marquage initial $[\,[\,]\,]$. La place `Success` contient la liste des reconnaissances de la chronique A.

La partie du réseau composée de la place `Wini` et de la transition `Forget` est utilisée dans le cas de l'absence. Son fonctionnement est donc explicité dans la section de la reconnaissance de l'absence (3.3.5).

Transitions Lorsque la transition `End` est tirée, l'entier de la place `Present` est incrémenté de 1. La valeur du compteur est apposée aux évènements pour les distinguer entre eux, comme il est détaillé par la suite. Il faut donc augmenter le compteur à chaque évènement du flux. Au tirage de la transition `End`, le marquage du réseau évolue comme suit :

$$\begin{matrix}\texttt{Start}\\ \texttt{Success}\\ \texttt{Present}\end{matrix}\begin{bmatrix} curr \\ inst \\ cpt \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} curr \\ inst \\ \mathbf{cpt+1} \end{bmatrix}$$

Lorsque la transition `A` est tirée, le contenu de la place `Success` est modifié : la liste des reconnaissances déjà présente dans la place `Success` est complétée par une nouvelle reconnaissance, notée $(E(a), cpt+1)$ (que l'on notera aussi E_a^{cpt+1}) où cpt est la valeur du compteur d'évènements. Ci-dessous, nous avons simplifié la définition de la fonction complete en intégrant le fait que, ici, $curr = [\,[\,]\,]$. La fonction ANR (Add New Recognition) ajoute la nouvelle reconnaissance $[E_a^{cpt+1}]$ à la liste $inst$.

$$\begin{matrix}\texttt{Start}\\ \texttt{Success}\\ \texttt{Present}\end{matrix}\begin{bmatrix} curr \\ inst \\ cpt \end{bmatrix} \xrightarrow{\texttt{A}} \begin{bmatrix} curr \\ \mathbf{ANR(inst, [\,[E_a^{cpt+1}]\,])} \\ cpt \end{bmatrix}$$

Stratégie de tirage Considérons un flux φ. Ce sont les évènements du flux φ qui déterminent la suite de transitions à tirer. Si un évènement de nom différent de A a lieu, seule la transition `End` est tirée : seul le compteur d'évènement est incrémenté. Si un évènement de nom A a lieu, la transition `A` est tirée, de façon à ajouter l'évènement à la liste des reconnaissances, puis la transition `End` est tirée pour incrémenter le compteur.

Exemple 8. Soit $\varphi = ((b,1),(\mathbf{a},\mathbf{2}),(d,3),(\mathbf{a},\mathbf{4}))$ avec $a, b, d \in \mathfrak{N}$ où nous souhaitons reconnaître la chronique A.

La liste des transitions à tirer correspondant au flux φ est [`End`, `A`, `End`, `End`, `A`, `End`].

Le marquage des places du réseau évolue comme suit :

$$\begin{matrix}\texttt{Start}\\ \texttt{Success}\\ \texttt{Present}\end{matrix}\begin{bmatrix} [\,] \\ [\,] \\ 0 \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} [\,] \\ [\,] \\ \mathbf{1} \end{bmatrix} \xrightarrow{\texttt{A}} \begin{bmatrix} [\,] \\ [\,\mathbf{[E_a^2]}\,] \\ 1 \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} [\,] \\ [\,[E_a^2]\,] \\ \mathbf{2} \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} [\,] \\ [\,[E_a^2]\,] \\ \mathbf{3} \end{bmatrix}$$

$$\xrightarrow{\texttt{A}} \begin{bmatrix} [\,] \\ [\,\mathbf{[E_a^2]}, \mathbf{[E_a^4]}\,] \\ 3 \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} [\,] \\ [\,[E_a^2], [E_a^4]\,] \\ \mathbf{4} \end{bmatrix}$$

On obtient bien deux reconnaissances, $[E_a^2]$ et $[E_a^4]$ dues à $(a,2)$ et $(a,4)$.

3.3.2 Reconnaissance d'une séquence

Nous allons étudier le réseau de Petri de la Figure 3.14 qui correspond à la chronique $A\ B$ où $A, B \in \mathfrak{N}$ pour examiner le déroulement d'une séquence.

Il est composé de trois sous-réseaux : le réseau de compteur d'évènements, le réseau relatif à la transition `A` et le réseau relatif à la transition `B`.

Places Comme précédemment, les places `Present` des trois sous-réseaux sont fusionnées, de type `INT`, et contiennent un entier correspondant au compteur d'évènements.

La place `Start` du réseau A a un marquage constant, égal à son marquage initial [[ ]].

La place `Success` du réseau A est fusionnée avec la place `Start` du réseau B. Les deux sous-réseaux A et B fonctionnent donc en série. La place `Start` du réseau B n'a donc plus un marquage constant [[ ]]. Son marquage initial est la liste vide [ ] qui est le marquage initial de `Success`(A). Comme évoqué dans la Remarque 18, ceci implique que le mécanisme du réseau relatif à B ne peut pas être effectif tant que le marquage initial n'a pas été modifié car il n'y a pas de reconnaissance partielle à compléter. Au fur et à mesure de l'exécution du réseau, `Start`(B) pourra contenir une liste contenant une liste non vide : il s'agira des reconnaissances partielles de $A\,B$, c'est-à-dire des reconnaissances de A, qu'il faut compléter par des reconnaissances de B.

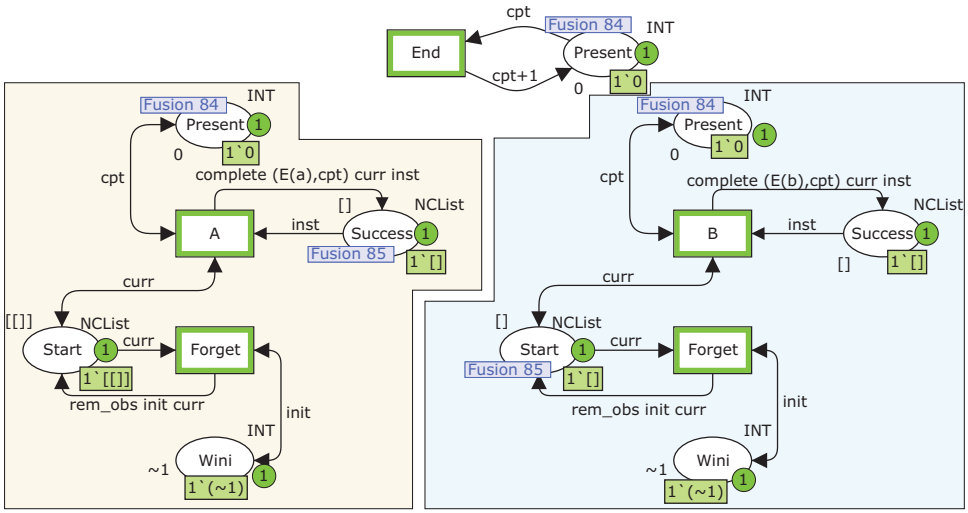


Figure 3.14 – Réseau correspondant à la chronique $A\ B$

La place `Success` du réseau B contient les reconnaissances de la chronique $A\,B$.

Comme précédemment, on ignore pour le moment les parties des réseaux A et B composées de la place `Wini` et de la transition `Forget` et relatives à l'absence.

Transitions Lorsque la transition `End` est tirée, le compteur d'évènements est incrémenté de 1.

Lorsque la transition `A` est tirée, une nouvelle reconnaissance de A est ajoutée à la liste contenue dans la place `Success`, comme dans le réseau précédent.

Lorsque la transition `B` est tirée, la reconnaissance de B complète les reconnaissances de A qui se trouvent dans la place `Start`(B) (à l'aide de la fonction CPR - Complete Partial Recognition) pour former des reconnaissances de $A\,B$ qui sont ajoutées à la liste de la place `Success`(B). En effet, la fonction complete prend en argument la variable $curr_B$ qui correspond au contenu de la place `Start` et complète les reconnaissances partielles qui s'y trouvent. C'est pour cela que le marquage initial de `Success`(A) et `Start`(B) est $[[]]$ et non $[]$: la fonction complete va compléter la liste vide, il n'y a pas encore de reconnaissance de A.

$$\begin{matrix}\texttt{Start(A)}\\ \texttt{Start(B)}\\ \texttt{Success(B)}\\ \texttt{Present}\end{matrix}\begin{bmatrix} curr_A \\ curr_B \\ inst_B \\ cpt \end{bmatrix} \xrightarrow{\texttt{B}} \begin{bmatrix} curr_A \\ curr_B \\ \mathbf{ANR(inst_B, CPR([[E_b^{cpt+1}]], curr_B))} \\ cpt \end{bmatrix}$$

Stratégie de tirage Si un évènement de nom différent de A et de B a lieu, seule la transition `End` est tirée, et donc seul le compteur d'évènements est incrémenté. Si un évènement de nom A a lieu, la transition `A` est tirée de façon à ajouter l'évènement à la liste des reconnaissances partielles, puis la transition `End` est tirée. De même, si un évènement de nom B a lieu, la transition `B` est tirée

de façon à compléter les reconnaissances partielles et obtenir des reconnaissances de $A\,B$, puis la transition `End` est tirée.

Exemple 9. Soit $\varphi = ((b,1),(\mathbf{a},\mathbf{2}),(d,3),(\mathbf{a},\mathbf{4}),(\mathbf{b},\mathbf{5}))$ avec $a,b,d \in \mathfrak{N}$ où l'on souhaite reconnaître la chronique $A\ B$.

La liste des transitions à tirer correspondant au flux φ est [`B`, `End`, `A`, `End`, `End`, `A`, `End`, `B`, `End`].

Le marquage des places du réseau évolue comme suit (on rappelle que les places `Success`(`A`) et `Start`(`B`) sont fusionnées et ont donc le même marquage) :

$$\begin{array}{l}\texttt{Start(A)}\\ \texttt{Success(A)}\\ \texttt{Success(B)}\\ \texttt{Present}\end{array}\begin{bmatrix}[\,[\,]\,]\\ [\,]\\ [\,]\\ 0\end{bmatrix}\xrightarrow{\texttt{B}}\begin{bmatrix}[\,[\,]\,]\\ [\,]\\ [\,]\\ 0\end{bmatrix}\xrightarrow{\texttt{End}}\begin{bmatrix}[\,[\,]\,]\\ [\,]\\ [\,]\\ \mathbf{1}\end{bmatrix}\xrightarrow{\texttt{A}}\begin{bmatrix}[\,[\,]\,]\\ [\,[\mathbf{E_a^2}]\,]\\ [\,]\\ 1\end{bmatrix}\xrightarrow{\texttt{End}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E_a^2]\,]\\ [\,]\\ \mathbf{2}\end{bmatrix}$$

$$\xrightarrow{\texttt{End}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E_a^2]\,]\\ [\,]\\ \mathbf{3}\end{bmatrix}\xrightarrow{\texttt{A}}\begin{bmatrix}[\,[\,]\,]\\ [\,[\mathbf{E_a^2}],[\mathbf{E_a^4}]\,]\\ [\,]\\ 3\end{bmatrix}\xrightarrow{\texttt{End}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E_a^2],[E_a^4]\,]\\ [\,]\\ \mathbf{4}\end{bmatrix}$$

$$\xrightarrow{\texttt{B}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E_a^2],[E_a^4]\,]\\ [\,[\mathbf{E_a^2},\mathbf{E_b^5}],[\mathbf{E_a^4},\mathbf{E_b^5}]\,]\\ 4\end{bmatrix}\xrightarrow{\texttt{End}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E_a^2],[E_a^4]\,]\\ [\,[E_a^2,E_b^5],[E_a^4,E_b^5]\,]\\ \mathbf{5}\end{bmatrix}$$

Remarquons que le premier tirage de la transition `B` ne modifie pas le marquage du réseau. Ceci est dû au fait que le marquage de la place `Start`(`B`) est [ ] et qu'il n'y a donc aucune liste, même vide, à compléter. Comme il s'agit d'une séquence, on ne commence pas à reconnaître B tant que l'on n'a pas reconnu A.

Le cas particulier $A\ A$ Il est important de vérifier que le cas particulier $A\ A$ ne pose pas de problème dans la gestion des différentes combinaisons pour les reconnaissances et qu'il faut bien deux occurrences distinctes de A pour reconnaître la séquence. Ceci est garanti par la fonction complete qui ne complète que les reconnaissances datant d'un instant inférieur ou égal à l'instant courant *cpt* du compteur d'évènements.

Étudions le mécanisme sur le flux $((a,1),(a,2))$. On dénomme A_1 et A_2 les deux réseaux fusionnés pour former $N(A\ A)$. À la suite du tirage de la transition $\texttt{A}_1$ pour le traitement de l'évènement $(a,1)$, le marquage de `Success`(A_1) (qui est aussi celui de `Start`(A_2)) est $[[E_a^1]]$. Lorsque l'on tire $\texttt{A}_2$, la fonction complete examine l'instant de chacune des reconnaissances à compléter. Il n'y a pour le moment que $[E_a^1]$ à compléter et son instant de reconnaissance est 1. La compteur d'évènements est encore à 0 donc la contrainte n'est pas vérifiée ($\neg\ 0 \geq 1$) et $[E_a^1]$ ne peut être complétée par $(a,1)$. On tire alors la transition `End` et le compteur passe à 1. $[E_a^1]$ peut donc maintenant être complétée. Pour traiter $(a,2)$, on tire $\texttt{A}_1$ ce qui modifie le marquage de `Success`(A_1) à $[[E_a^1],[E_a^2]]$ et on tire $\texttt{A}_2$ ce qui ne peut compléter que $[E_a^1]$. On obtient alors $[[E_a^1,E_a^2]]$

comme marquage de `Success`(A_2) ; on a bien une unique reconnaissance et le cas particulier $A\ A$ est correctement traité.

3.3.3 Reconnaissance d'une disjonction

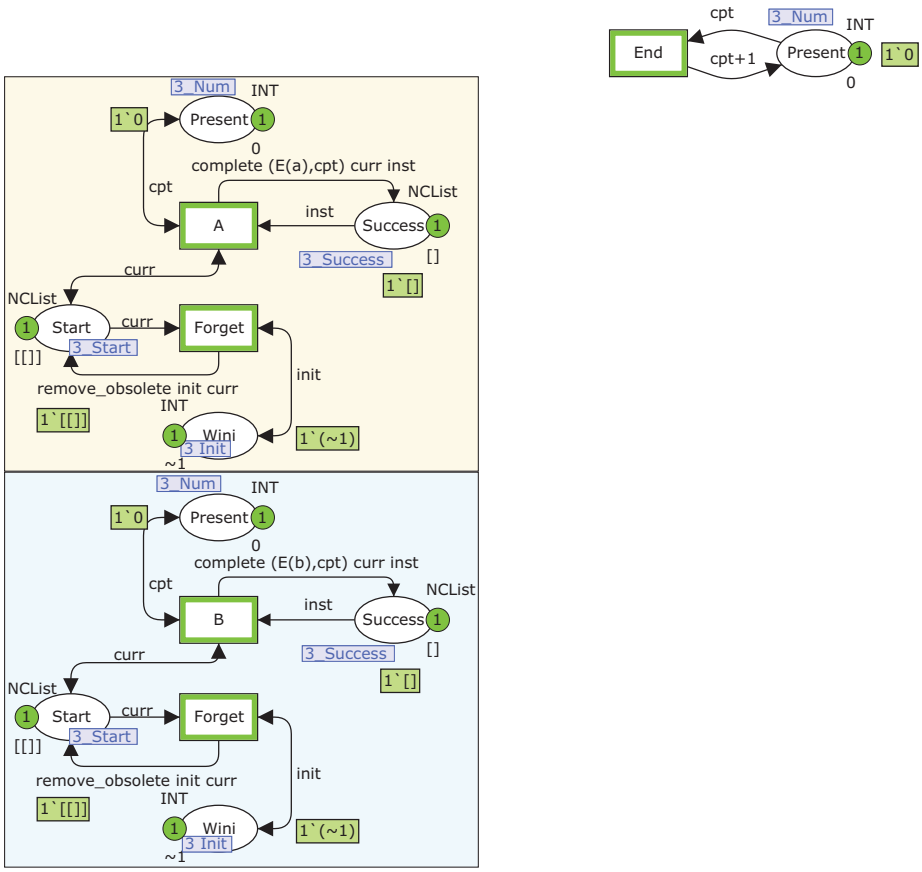


Figure 3.15 – Réseau correspondant à la chronique $A \mid\mid B$

Étudions le comportement d'une disjonction à travers le réseau de Petri de la Figure 3.15 qui correspond à la chronique $A \mid\mid B$ où $A, B \in \mathfrak{N}$.

Il est composé de trois sous-réseaux : le réseau de compteur d'évènements, le réseau relatif à la transition `A` et le réseau relatif à la transition `B`.

Places Comme précédemment, les places `Present` des trois sous-réseaux sont fusionnées, de type `INT`, et contiennent un entier correspondant au compteur d'évènements.

Les places `Start` des réseaux A et B sont fusionnées et ont ici un marquage constant, égal à $[\,[\,]\,]$. Les deux sous-réseaux fonctionnent donc en parallèle.

Les places `Success` des réseaux A et B sont aussi fusionnées. Elles contiennent la liste des reconnaissances de $A \,||\, B$.

Comme précédemment, on ignore pour le moment les parties des réseaux A et B composées de la place `Wini` et de la transition `Forget`.

Transitions Lorsque la transition `End` est tirée, le compteur d'évènements est incrémenté de 1.

Lorsque la transition `A` est tirée, une nouvelle reconnaissance de A est ajoutée à la liste contenue dans la place `Success`. Une reconnaissance de A est une reconnaissance de $A \,||\, B$.

De même, lorsque la transition `B` est tirée, une nouvelle reconnaissance de B est ajoutée à la liste contenue dans la place `Success`. Ainsi, la place `Success` contient toutes les reconnaissances de A et toutes celles de B, ce qui correspond à toutes les reconnaissances de $A \,||\, B$.

Stratégie de tirage Si un évènement de nom différent de A et de B a lieu, seule la transition `End` est tirée, et donc seul le compteur d'évènements est incrémenté. Si un évènement de nom A (respectivement B) a lieu, la transition `A` (respectivement `B`) est tirée de façon à ajouter l'évènement à la liste des reconnaissances de la place `Success`, puis la transition `End` est tirée.

Exemple 10. Soit $\varphi = ((\mathbf{b}, \mathbf{1}), (\mathbf{a}, \mathbf{2}), (d, 3), (\mathbf{a}, \mathbf{4}))$ avec $a, b, d \in \mathfrak{N}$ où nous souhaitons reconnaître $A \,||\, B$.

La liste des transitions à tirer correspondant au flux φ est $[\texttt{B}, \texttt{End}, \texttt{A}, \texttt{End}, \texttt{End}, \texttt{A}, \texttt{End}]$.

Le marquage des places du réseau évolue comme suit :

$$\begin{array}{l} \texttt{Start} \\ \texttt{Success} \\ \texttt{Present} \end{array} \begin{bmatrix} [\,[\,]\,] \\ [\,] \\ 0 \end{bmatrix} \xrightarrow{\texttt{B}} \begin{bmatrix} [\,[\,]\,] \\ [\,[\mathbf{E}_{\mathbf{b}}^{\mathbf{1}}]\,] \\ 0 \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} [\,[\,]\,] \\ [\,[E_b^1]\,] \\ \mathbf{1} \end{bmatrix} \xrightarrow{\texttt{A}} \begin{bmatrix} [\,[\,]\,] \\ [\,[\mathbf{E}_{\mathbf{b}}^{\mathbf{1}}], [\mathbf{E}_{\mathbf{a}}^{\mathbf{2}}]\,] \\ 1 \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} [\,[\,]\,] \\ [\,[E_b^1], [E_a^2]\,] \\ \mathbf{2} \end{bmatrix}$$

$$\xrightarrow{\texttt{End}} \begin{bmatrix} [\,[\,]\,] \\ [\,[E_b^1], [E_a^2]\,] \\ \mathbf{3} \end{bmatrix} \xrightarrow{\texttt{A}} \begin{bmatrix} [\,[\,]\,] \\ [\,[\mathbf{E}_{\mathbf{b}}^{\mathbf{1}}], [\mathbf{E}_{\mathbf{a}}^{\mathbf{2}}], [\mathbf{E}_{\mathbf{a}}^{\mathbf{4}}]\,] \\ 3 \end{bmatrix} \xrightarrow{\texttt{End}} \begin{bmatrix} [\,[\,]\,] \\ [\,[E_b^1], [E_a^2], [E_a^4]\,] \\ \mathbf{4} \end{bmatrix}$$

3.3.4 Reconnaissance d'une conjonction

Détaillons le processus de reconnaissance d'une conjonction à travers le réseau de Petri de la Figure 3.16 qui correspond à la chronique $A\&B$ où $A, B \in \mathfrak{N}$.

Il est composé de quatre sous-réseaux : le réseau de compteur d'évènements, le réseau de l'opérateur de conjonction $\texttt{OP}_{\texttt{AND}}$, le réseau relatif à la transition `A` et celui relatif à la transition `B`.

Places Comme précédemment, les places `Present` sont fusionnées, de type `INT`, et contiennent un entier correspondant au compteur d'évènements.

Les places `Start` des réseaux A et B sont fusionnées et ont ici un marquage constant, égal à $[\,[\,]\,]$. Nous souhaitons reconnaître A et B dans un ordre quelconque donc les deux réseaux fonctionnent en parallèle.

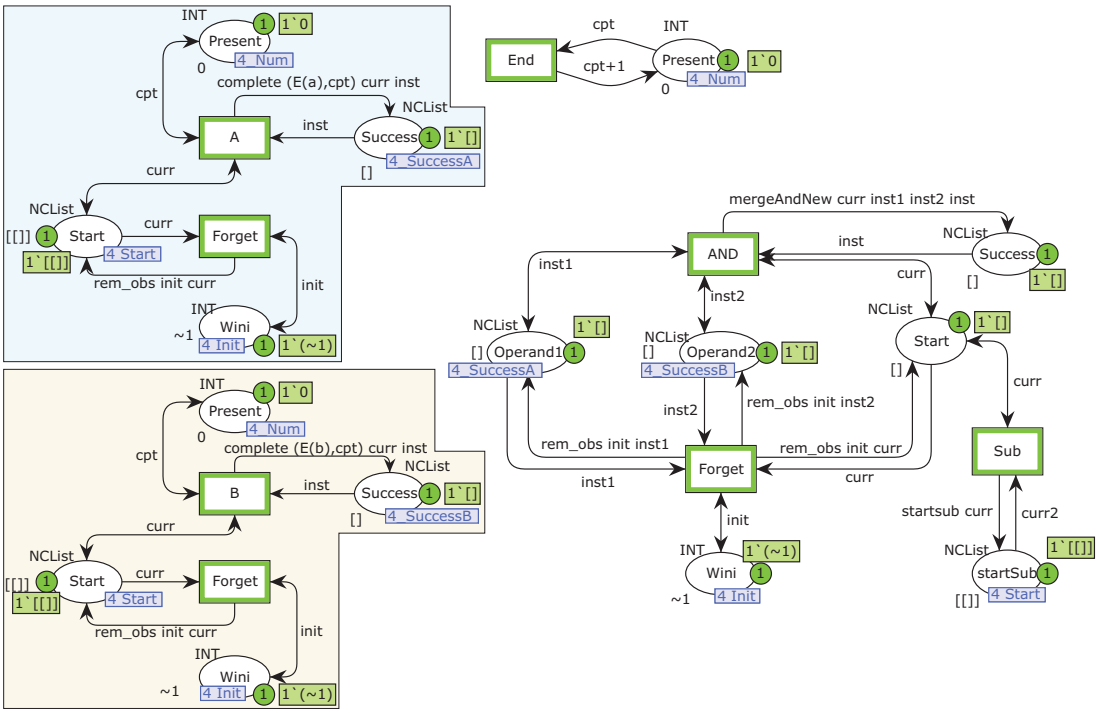


Figure 3.16 – Réseau correspondant à la chronique $A\&B$

La place `Success` du réseau A (respectivement B) est fusionnée avec la place `Operand1` (respectivement `Operand2`) du réseau de conjonction. Elle contient les reconnaissances de A (respectivement B) qui sont des reconnaissances partielles de $A\&B$. Il reste alors à effectuer les combinaisons de ces reconnaissances partielles pour former les reconnaissances de $A\&B$. C'est le rôle de l'opérateur $\texttt{OP}_{\texttt{AND}}$.

Ces combinaisons, c'est-à-dire les reconnaissances de $A\&B$, sont stockées dans la place `Success(AND)`.

Le calcul de ces combinaisons n'est pas évident et présente plusieurs difficultés :

— Nous souhaitons respecter la multiplicité des reconnaissances, donc réaliser *toutes* les combinaisons possibles sans créer de doublons injustifiés (*i.e.* sans engendrer deux témoins d'une unique reconnaissance). C'est un enjeu qui n'apparaît que peu dans le réseau présenté ici car nous manipulons des listes de reconnaissances donc la complexité de l'algorithme est dissimulée dans l'implémentation de la fonction mergeAndNew. La difficulté est plus apparente dans le Chapitre 4 où l'on manipule un jeton par reconnaissance ce qui conduit à un réseau beaucoup plus conséquent pour l'opérateur `AND`.

— La détermination des bonnes combinaisons dans une conjonction lors de la composition avec une séquence – ici nous prenons l'exemple de la chronique C_{deb} $(A\&B)$ – est complexe et soulève une double problématique :

— D'une part, la première idée intuitive dans le cas d'une séquence consiste à fusionner la place `Success`(C_{deb}) avec les places `Start`(A) et `Start`(B), c'est-à-dire définir que `Start`($A\&B$) = `Start`(A) [6]. C'est d'ailleurs ce qui est réalisé dans [CCK11]. Cependant, nous obtenons alors, dans les places `Success`(A) et `Success`(B), des reconnaissances de C_{deb} A et de C_{deb} B qu'il faut ensuite recombiner en reconnaissances de C_{deb} ($A\&B$). Ceci n'est pas réalisable si l'on n'a pas stocké par ailleurs la partie de la reconnaissance correspondant au préfixe commun C_{deb}. Nous introduisons donc la place `Start(AND)` qui stocke les reconnaissances de C_{deb}. L'opérateur d'absence, au travers de la transition `AND` et de la fonction mergeAndNew, combine alors les reconnaissances de C_{deb}, A et B, qui arrivent séparément, pour former les reconnaissances de C_{deb} ($A\&B$).

— D'autre part, nous ne souhaitons pas commencer à reconnaître $A\&B$ tant que C_{deb} n'a pas été reconnue. Pour ce faire, nous introduisons la place `StartSub` que nous fusionnons avec les places `Start`(A) et `Start`(B). Lorsque `Success`(C_{deb}) contient une liste vide (c'est-à-dire lorsque l'on n'a pas encore reconnu C_{deb}), `StartSub` contient également une liste vide [ ]. A et B ne peuvent alors pas encore être reconnus car il n'y a rien à compléter dans les places `Start` correspondantes. Inversement, lorsque `Success`(C_{deb}) contient une ou plusieurs reconnaissances, `StartSub` est marquée [[ ]]. Les réseaux $N'(A)$ et $N'(B)$ peuvent alors compléter la liste vide [ ] avec des reconnaissances respectivement de A et de B.

Comme précédemment, on ignore pour le moment les parties des réseaux A et B composées de la place `Wini` et de la transition `Forget`.

Transitions Lorsque la transition `End` est tirée, le compteur d'évènements est incrémenté de 1.

Lorsque la transition `A` (respectivement `B`) est tirée, une nouvelle reconnaissance de A (respectivement B) est ajoutée à la liste contenue dans la place `Operand1` (respectivement `Operand2`).

Lorsque la transition `AND` est tirée, les reconnaissances de A dans `Operand1`, les reconnaissances de B dans `Operand2`, et les reconnaissances d'une éventuelle séquence à compléter dans `Start(AND)` sont combinées pour former des reconnaissances de $A\&B$ intégrées éventuellement dans une reconnaissance plus complexe. Ces nouvelles reconnaissances sont ajoutées à la liste de la place `Success` à l'aide de la fonction mergeAndNew. Cette fonction ajoute uniquement les nouvelles reconnaissances de $A\&B$. En effet, dans `Operand1` et dans `Operand2` se trouvent toutes les reconnaissances de A et de B, et si l'on se contentait de faire tous les couples possibles de reconnaissances de A et de reconnaissances de B, nous obtiendrions, entre autre, des reconnaissances déjà ajoutées précédemment. Notons qu'il y existe un critère simple pour déterminer s'il faut ajouter une reconnaissance : une nouvelle reconnaissance de $A\&B$ est une reconnaissance où soit la reconnaissance de A, soit la reconnaissance de B vient d'apparaître, c'est-à-dire qui est de la forme E_a^{cpt+1} ou E_b^{cpt+1} où cpt est la valeur actuelle du compteur d'évènements.

La transition `Sub` actualise le contenu des places `Operand1` et `Operand2` selon le contenu de la place `Start(AND)` afin de ne pas activer les réseaux relatifs à A et B lorsque cela est inutile, comme évoqué précédemment.

6. Et donc `Start`($A\&B$) = `Start`(B) car `Start`(A) et `Start`(B) sont fusionnées.

Stratégie de tirage On commence par tirer la transition `Sub` pour mettre à jour l'activation des réseaux A et B. Ensuite, si un évènement de nom différent de A et de B a lieu, seule la transition `End` est tirée, et donc seul le compteur d'évènements est incrémenté. Au contraire, si un évènement de nom A (respectivement B) a lieu, la transition `A` (respectivement `B`) est tirée de façon à ajouter l'évènement à la liste des reconnaissances de la place `Operand1` (respectivement `Operand2`). La transition `AND` est ensuite tirée pour créer les nouvelles reconnaissances éventuelles de $A\&B$ et les insérer dans la liste de la place `Success`, puis la transition `End` est tirée.

Exemple 11. Soit $\varphi = ((\mathbf{b}, \mathbf{1}), (\mathbf{a}, \mathbf{2}), (d, 3), (\mathbf{b}, \mathbf{4}))$ avec $a, b, d \in \mathfrak{N}$ où l'on souhaite reconnaître $A\&B$.

La liste des transitions à tirer correspondant au flot φ est [`Sub`, `B`, `AND`, `End`, `Sub`, `A`, `AND`, `End`, `Sub`, `End`, `Sub`, `B`, `AND`, `End`].

Le marquage des places du réseau évolue comme suit :

$$\begin{array}{l}\texttt{StartSub}\\ \texttt{Operand1}\\ \texttt{Operand2}\\ \texttt{Success}\\ \texttt{Present}\end{array}\begin{bmatrix}[\,[\,]\,]\\ [\,]\\ [\,]\\ [\,]\\ 0\end{bmatrix}\xrightarrow{\texttt{Sub,B}}\begin{bmatrix}[\,[\,]\,]\\ [\,]\\ [\,[\mathbf{E^1_b}]\,]\\ [\,]\\ 0\end{bmatrix}\xrightarrow{\texttt{AND}}\begin{bmatrix}[\,[\,]\,]\\ [\,]\\ [\,[E^1_b]\,]\\ [\,]\\ 0\end{bmatrix}\xrightarrow{\texttt{End}}\begin{bmatrix}[\,[\,]\,]\\ [\,]\\ [\,[E^1_b]\,]\\ [\,]\\ \mathbf{1}\end{bmatrix}\xrightarrow{\texttt{Sub,A}}\begin{bmatrix}[\,[\,]\,]\\ [\,[\mathbf{E^2_a}]\,]\\ [\,[E^1_b]\,]\\ [\,]\\ 1\end{bmatrix}$$

$$\xrightarrow{\texttt{AND}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E^2_a]\,]\\ [\,[E^1_b]\,]\\ [\,[\mathbf{E^1_b}, \mathbf{E^2_a}]\,]\\ 1\end{bmatrix}\xrightarrow{\texttt{End,Sub,End}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E^2_a]\,]\\ [\,[E^1_b]\,]\\ [\,[E^1_b, E^2_a]\,]\\ \mathbf{3}\end{bmatrix}\xrightarrow{\texttt{Sub,B}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E^2_a]\,]\\ [\,[\mathbf{E^1_b}], [\mathbf{E^4_b}]\,]\\ [\,[E^1_b, E^2_a]\,]\\ 3\end{bmatrix}$$

$$\xrightarrow{\texttt{AND,End}}\begin{bmatrix}[\,[\,]\,]\\ [\,[E^2_a]\,]\\ [\,[E^1_b], [E^4_b]\,]\\ [\,[\mathbf{E^1_b}, \mathbf{E^2_a}], [\mathbf{E^2_a}, \mathbf{E^4_b}]\,]\\ \mathbf{4}\end{bmatrix}$$

Le cas particulier de $A\&A$ De même que pour la séquence, il est intéressant de vérifier que le cas particulier $A\&A$ est correctement traité par ce modèle de la conjonction et que l'on obtient bien le bon nombre de reconnaissances.

On considère le flux $\varphi = ((a, 1), (a, 2))$. D'après la sémantique du langage définie dans la Section 2.3.2, le flux φ doit donner lieu à quatre reconnaissances de $A\&A$: $\langle (a, 1), (a, 1)\rangle$, $\langle (a, 2), (a, 2)\rangle$, $\langle (a, 1), (a, 2)\rangle$, et $\langle (a, 2), (a, 1)\rangle$.

Si on appelle $S(a)$ la liste des transitions à tirer pour traiter l'occurrence d'un évènement a dans le flux pour la chronique $A\&A$, le marquage du réseau évolue comme suit :

$$\begin{array}{l}\texttt{Start}\\ \texttt{Success}\\ \texttt{Operand1}\\ \texttt{Operand2}\end{array}\xrightarrow{S(a)}\begin{bmatrix}[\,[\,]\,]\\ [\,[\mathbf{E^1_a}, \mathbf{E^1_a}]\,]\\ [\,[\mathbf{E^1_a}]\,]\\ [\,[\mathbf{E^1_a}]\,]\end{bmatrix}\xrightarrow{S(a)}\begin{bmatrix}[\,[\,]\,]\\ [\,[E^1_a, E^1_a], [\mathbf{E^1_a}, \mathbf{E^2_a}], [\mathbf{E^2_a}, \mathbf{E^1_a}], [\mathbf{E^2_a}, \mathbf{E^2_a}]\,]\\ [\,[E^1_a], [\mathbf{E^2_a}]\,]\\ [\,[E^1_a], [\mathbf{E^2_a}]\,]\end{bmatrix}$$

On obtient donc bien les quatre reconnaissances.

3.3.5 Reconnaissance d'une absence

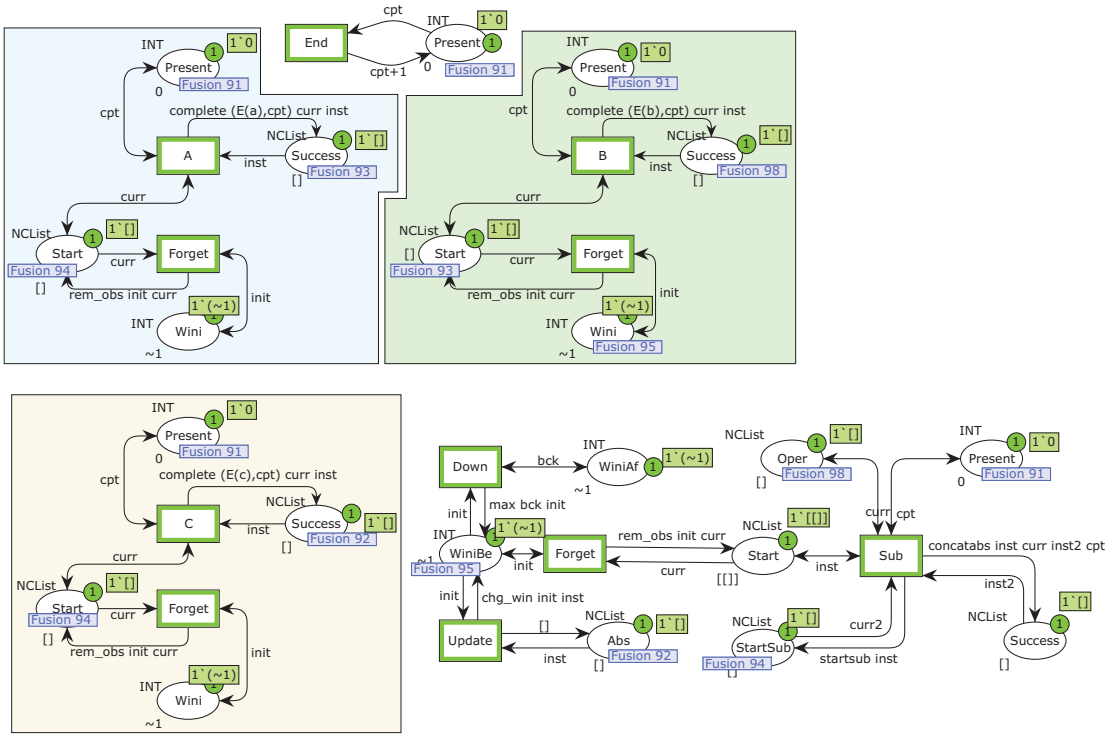


FIGURE 3.17 – Réseau correspondant à la chronique $(A\ B) - [C[$

Étudions maintenant le cas de l'absence à travers le réseau de Petri de la Figure 3.17 qui correspond à la chronique $(A\ B) - [C[$ où $A, B, C \in \mathfrak{N}$.

Il est composé de quatre sous-réseaux : le réseau de compteur d'évènements, le réseau d'absence $\texttt{OP}_{\texttt{ABS}}$, le réseau relatif à la chronique $A\,B$ (composé du réseau A et du réseau B) et le réseau relatif à la transition `C`.

Places Comme précédemment, les places `Present` sont fusionnées, de type `INT`, et contiennent un entier correspondant au compteur d'évènements.

La place `Start` du réseau $A\,B$ (c'est-à-dire la place `Start` du réseau A) et celle du réseau C sont fusionnées et ont ici un marquage constant, égal à $[[]]$. Cela permet de synchroniser l'activation des réseaux A B et C.

La place `Success` du réseau $A\,B$ (c'est-à-dire la place `Success` du réseau B) contient les reconnaissances de $(A\ B) - [C[$ et est fusionnée avec la place `Oper` de l'opérateur `ABS` pour que

les reconnaissances de $(A\ B) - [C[$ puissent être recombinées si la chronique est insérée dans une séquence comme $D\ (A\ B) - [C[$ (nous détaillons ce cas particulier par la suite). S'il n'y a pas de séquence avant l'absence, les reconnaissances de `Oper` sont simplement transférées dans `Success(ABS)` (elles viennent en fait compléter une liste vide).

La place `Abs` du réseau d'absence et `Success`(C) sont fusionnées et contiennent les reconnaissances de C, qui est dans notre exemple la chronique interdite.

Pour la chronique $(A\ B) - [C[$, les reconnaissances de C vont invalider toutes les reconnaissances partielles de $(A\ B)$ durant lesquelles un C a eu lieu. Pour implémenter ce processus, nous allons supprimer les reconnaissances de A ayant été suivies d'une occurrence de C afin qu'elles ne puissent être complétées par un B. Ainsi, nous n'obtenons que des reconnaissances de $(A\ B)$ vérifiant la contrainte qu'aucun C n'a eu lieu entre A et B. La place `Wini` présente dans tous les réseaux définis jusqu'alors contient un entier qui sert de repère pour les suppressions des reconnaissances invalidées. Elle contient l'entier correspondant à la valeur, incrémentée de 1, du compteur d'évènements lors de la dernière reconnaissance de la chronique « interdite » (ici, C). Cette valeur correspond à l'indice du compteur à partir duquel les reconnaissances de $(A\ B)$ peuvent à nouveau être considérées comme valides, c'est-à-dire sans occurrence de C, et peuvent être ajoutées à la liste des reconnaissances de $(A\ B) - [C[$.

Dans l'opérateur `ABS`, il s'agit de mettre à jour le contenu de ces places `Wini`. Il faut tenir compte de deux choses :

1. non seulement de l'absence qui est considérée (ici, $-[C[$) ;
2. mais aussi de la possibilité que la chronique soit imbriquée dans une autre absence, comme par exemple dans la chronique $(\ ((A\ B) - [C[)\ E\ ) - [D[$ dont le réseau est représenté Figure 3.18.

En d'autres termes, il faut faire attention aux bornes de l'absence. Comme nous le détaillerons par la suite, dans la chronique $(\ ((A\ B) - [C[)\ E\ ) - [D[$, l'occurrence d'un D doit avoir une influence sur tout $((A\ B) - [C[)\ E$ mais l'occurrence d'un C ne doit avoir d'influence que sur $(A\ B)$. Ainsi, une fois que la séquence $(A\ B)$ est reconnue, l'occurrence d'un C ne doit pas supprimer les reconnaissances complètes de $(A\ B)$ ni celles de E, mais seulement les reconnaissances de A non encore complétées par un B.

L'opérateur `ABS` fonctionne donc comme une diode ne permettant à la valeur de `Wini` de se propager que dans un seul sens. Il possède deux places `Wini` :

1. la place `WiniBe` en charge de l'absence considérée à ce niveau (ici, $-[C[$) et qui est donc fusionnée avec `Wini`$(A\ B)$ pour supprimer les reconnaissances obsolètes de A ;
2. la place `WiniAf` qui propage vers `WiniBe` les absences éventuelles de niveau supérieur mais qui bloque la propagation en sens inverse :
   — lorsqu'il n'y a pas d'autre absence, comme dans le cas de la Figure 3.17 avec $(A\ B) - [C[$, le marquage de `WiniAf` est constant égal à -1 et n'aura donc aucun effet sur le réseau,
   — lorsqu'il y a des absences de niveau supérieur (comme dans le cas de la Figure 3.18 avec la chronique $(\ ((A\ B) - [C[)\ E\ ) - [D[$), la place `WiniAf` est fusionnée avec les places `Wini` des niveaux supérieurs et permet ainsi de faire redescendre l'entier qui y est stocké, sans que l'entier de `WiniBe` ne puisse jamais remonter.

Le fonctionnement des places `WiniBe` et `WiniAf` est illustré dans l'Exemple 13.

Transitions Lorsque la transition `End` est tirée, le compteur d'évènements est incrémenté de 1.

Lorsque la transition `A` (respectivement `B` et `C`) est tirée, une nouvelle reconnaissance de A (respectivement B et C) est ajoutée à la liste contenue dans la place `Success` du réseau A (respectivement B et C).

Lorsque la transition `Update` est tirée, le contenu de la place `WiniBe` est mis à jour avec l'indice du compteur de la dernière reconnaissance de C si celui-ci est plus grand que la valeur actuelle *init* stockée dans la place `WiniBe`.

Lorsque la transition `Down` est tirée, l'entier de repère, stocké dans `WiniAf` et provenant d'une absence de plus haut niveau, est transmis à `WiniBe` s'il est plus grand que la valeur actuelle qui y est stockée. L'existence de deux places distinctes `WiniBe` et `WiniAf` et de la transition `Down` est nécessaire pour contrôler la propagation des entiers repères des `Wini`, ce qui n'aurait pas été possible avec une simple fusion de `WiniBe` et `WiniAf`.

Lorsque la transition `Sub` est tirée :

— le réseau d'absence est activé selon le contenu de `Start(ABS)` ;
— les combinaisons des reconnaissances de `Start(ABS)` (une liste vide ou des reconnaissances non vides dans la structure de la chronique globale étudiée) avec des reconnaissances de $(C_1) - [C_2[$ qui sont dans `Oper` sont effectuées séquentiellement et ajoutées à la liste de reconnaissances de `Success(ABS)`.

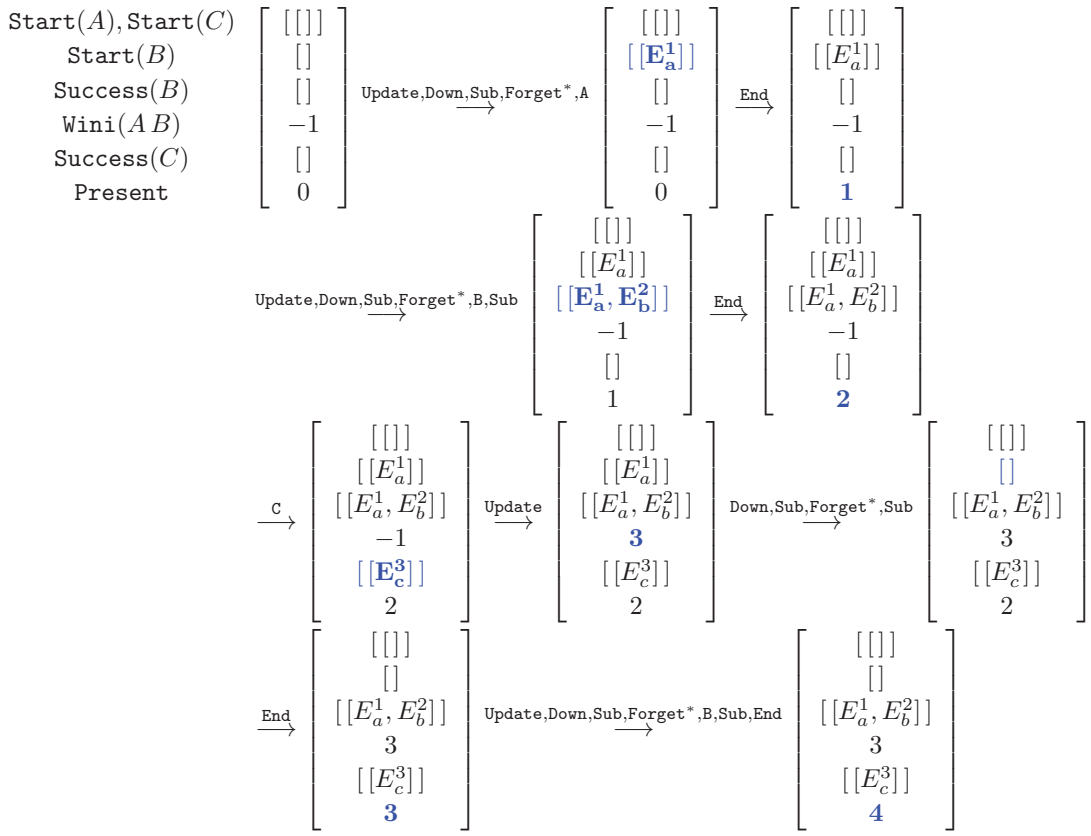
Lorsque la transition `Forget` du réseau B est tirée, les reconnaissances de A d'indice inférieur strictement à *init* sont supprimées de la place `Start`(B) et donc de `Success`(A). Ainsi, si une occurrence de A est suivie d'une occurrence de C, l'occurrence de A sera oubliée, de telle sorte qu'une reconnaissance de B ne puisse venir la compléter en une reconnaissance de $(A\ B) - [C[$.

Stratégie de tirage Si un évènement de nom différent de A, B et C a lieu, seule la transition `End` est tirée, et donc seul le compteur d'évènements est incrémenté. Si un évènement de nom C a lieu, on tire alors la transition `C`. On tire ensuite `Update` puis `Down` pour mettre à jour le contenu des places `Wini` du réseau, et `Sub` pour éventuellement activer le réseau. Les transitions `Forget` du réseau sont ensuite tirées pour supprimer les éventuelles reconnaissances rendues obsolètes par les nouvelles valeurs de `Wini`. Si un évènement de nom A (respectivement B) a lieu, la transition `A` (respectivement `B`) est tirée. On tire ensuite de nouveau `Sub` pour effectuer les combinaisons appropriées puis on tire finalement la transition `End` pour incrémenter le compteur.

Exemple 12. Soit $\varphi = ((\mathbf{a}, \mathbf{1}), (\mathbf{b}, \mathbf{2}), (c, 3), (b, 4))$ avec $a, b, c \in \mathfrak{N}$ où l'on souhaite reconnaître la chronique $(A\ B) - [C[$ représentée dans la Figure 3.17.

La liste des transitions à tirer correspondant au flot φ est [`Update`, `Down`, `Sub`, `Forget`*, `A`, `Sub`, `End`, `Update`, `Down`, `Sub`, `Forget`*, `B`, `Sub`, `End`, `C`, `Update`, `Down`, `Sub`, `Forget`*, `Sub`, `End`, `Update`, `Down`, `Sub`, `Forget`*, `B`, `Sub`, `End`].

Le marquage des places du réseau évolue comme suit :



Cas particulier de deux absences imbriquées Comme détaillé précédemment, c'est dans les chroniques contenant au moins deux absences imbriquées qu'apparaît l'utilité de la transition `Down` dans l'opérateur d'absence. Celle-ci assure un fonctionnement en « diode », ne permettant à la valeur de `Wini` de se propager que dans un seul sens. Sur l'exemple de la chronique $(\ ((A\,B) - [C[)\,E\ ) - [D[$ présentée Figure 3.18, en l'absence de la transition `Down`, dès que la chronique C est reconnue la valeur de `Wini` peut se propager en dehors de la chronique $(A\,B) - [C[$. Des reconnaissances valides seraient alors supprimées. L'exemple suivant illustre le fonctionnement en diode de l'opérateur d'absence.

Exemple 13. Considérons la chronique $(\ ((A\,B) - [C[)\,E\ ) - [D[$ et le flux $\varphi = ((a,1),(b,2),(d,3),(e,4),(\mathbf{a},\mathbf{5}),(\mathbf{b},\mathbf{6}),(c,7),(\mathbf{e},\mathbf{8}),(d,9))$ où $a,b,c,d,e \in \mathfrak{N}$.

On ne détaille pas ici toute la suite des transitions à tirer mais on note $S(e)$ les transitions à tirer pour notre chronique suite à l'évènement e.

Le marquage des places du réseau évolue comme suit. Nous nous intéressons en particulier aux places `WiniBe` et `WiniAf` de la première absence $-[C[$. Ici, `WiniBe` est fusionnée avec `Wini`(A) et `Wini`(B), alors que `WiniAf` est fusionnée avec `Wini`(E). Rappelons qu'ici `Success`$(B) =$ `Success`$((A\,B) - [C[)$ et `Success`$(E) =$ `Success`$((\ ((A\,B) - [C[)\,E\ ) - [D[)$.

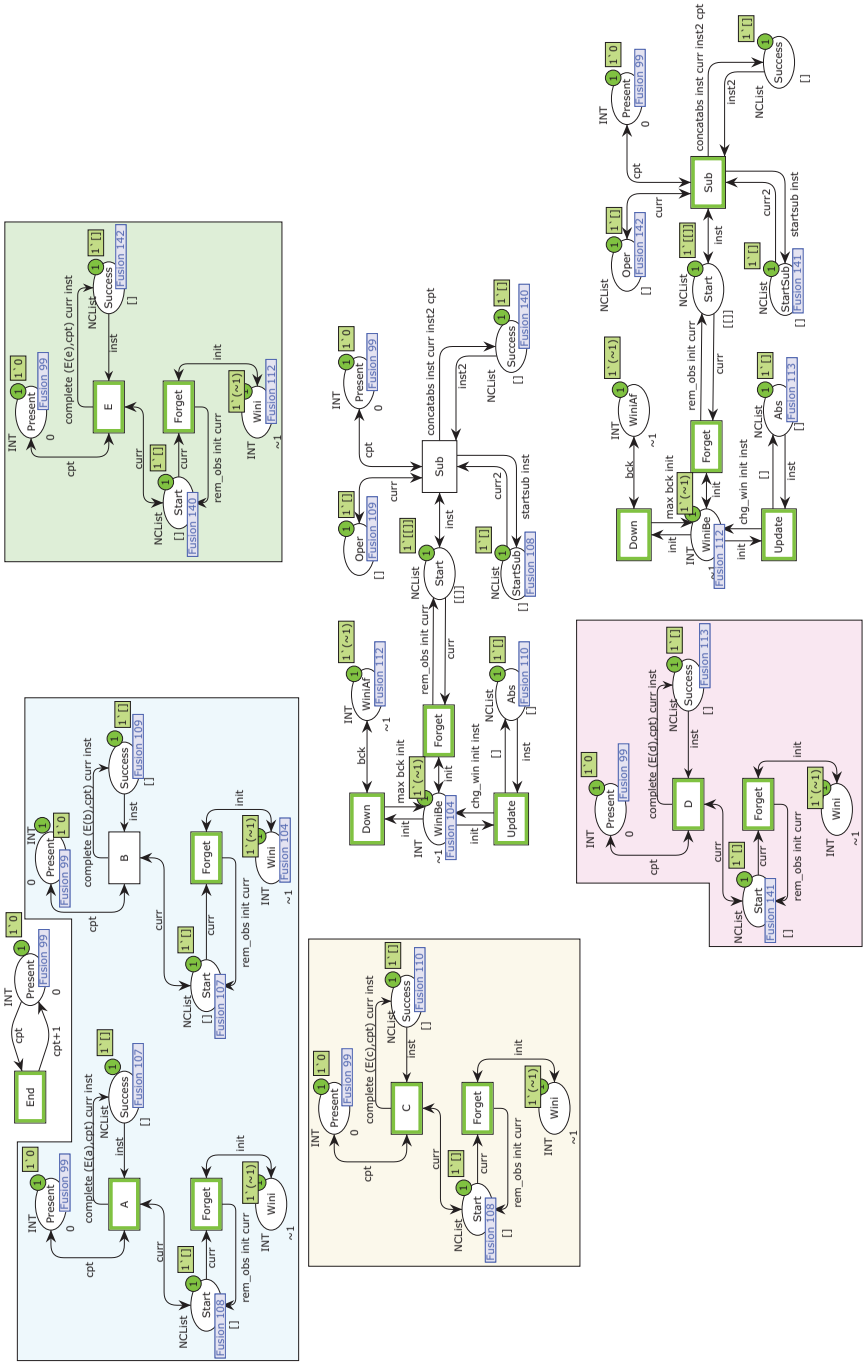


FIGURE 3.18 – Réseau correspondant à la chronique (((*A B*) − [*C*]) *E*) − [*D*[

$$
\begin{array}{l}
\texttt{Success}(A) \\
\texttt{Success}(B) \\
\quad \texttt{WiniBe} \\
\quad \texttt{WiniAf} \\
\texttt{Success}(C) \\
\texttt{Success}(D) \\
\texttt{Success}(E)
\end{array}
\begin{bmatrix} [\,] \\ [\,] \\ -1 \\ -1 \\ [\,] \\ [\,] \\ [\,] \end{bmatrix}
\xrightarrow{S(a),S(b)}
\begin{bmatrix} [\,[\mathbf{E^1_a}]\,] \\ [\,[\mathbf{E^1_a}, \mathbf{E^2_b}]\,] \\ -1 \\ -1 \\ [\,] \\ [\,] \\ [\,] \end{bmatrix}
\xrightarrow{S(d)}
\begin{bmatrix} [\,] \\ [\,] \\ \mathbf{3} \\ \mathbf{3} \\ [\,] \\ [\,[\mathbf{E^3_d}]\,] \\ [\,] \end{bmatrix}
\xrightarrow{S(e)}
\begin{bmatrix} [\,] \\ [\,] \\ 3 \\ 3 \\ [\,] \\ [\,[E^3_d]\,] \\ [\,] \end{bmatrix}
$$

$$
\xrightarrow{S(a),S(b)}
\begin{bmatrix} [\,[\mathbf{E^5_a}]\,] \\ [\,[\mathbf{E^5_a}, \mathbf{E^6_b}]\,] \\ 3 \\ 3 \\ [\,] \\ [\,[E^3_d]\,] \\ [\,] \end{bmatrix}
\xrightarrow{S(c)}
\begin{bmatrix} [\,] \\ [\,[E^5_a, E^6_b]\,] \\ \mathbf{7} \\ 3 \\ [\,[\mathbf{E^7_c}]\,] \\ [\,[E^3_d]\,] \\ [\,] \end{bmatrix}
\xrightarrow{S(e)}
\begin{bmatrix} [\,] \\ [\,[E^5_a, E^6_b]\,] \\ 7 \\ 3 \\ [\,[E^7_c]\,] \\ [\,[E^3_d]\,] \\ [\,[\mathbf{E^5_a}, \mathbf{E^6_b}, \mathbf{E^8_e}]\,] \end{bmatrix}
$$

$$
\xrightarrow{S(d)}
\begin{bmatrix} [\,] \\ [\,] \\ \mathbf{9} \\ \mathbf{9} \\ [\,[E^7_c]\,] \\ [\,[\mathbf{E^3_d}], [\mathbf{E^9_d}]\,] \\ [\,[E^5_a, E^6_b, E^8_e]\,] \end{bmatrix}
$$

Cas particulier d'une absence imbriquée dans une séquence Comme évoqué précédemment, la partie droite de l'opérateur d'absence, qui combine séquentiellement les reconnaissances de `Start(ABS)` avec les reconnaissances de l'absence qui sont dans `Oper`, est conçue pour traiter les cas d'imbrication d'une absence dans une autre chronique, en particulier dans une séquence. Considérons à titre d'exemple la chronique D $((A\ B) - [C[)$. Dans le modèle de [CCK11], les places `Success`(D) et `Start`(A) sont simplement fusionnées, donc, dans le réseau de l'absence, circulent des reconnaissances partielles de $(D\ A\ B)$ en attente de complétion. Ceci pose problème du fait du mécanisme de l'absence et des transitions `Forget`. En effet, pour savoir si une reconnaissance doit être supprimée, l'instant de début de reconnaissance est comparé à la valeur de l'entier repère `Wini`, et s'il lui est inférieur, elle doit être supprimée car un comportement interdit s'est produit pendant la reconnaissance. Dans le cas d'une reconnaissance partielle de $(D\ A\ B)$, l'instant de début correspondant à l'instant de reconnaissance de D, mais effectuer le test sur cet instant ne mène pas au résultat recherché car D est en dehors de la portée de l'absence (ce ne serait pas le cas si la chronique étudiée était $(D\ A\ B) - [C[$). Or, dans une reconnaissance partielle, il n'est pas trivial de déterminer à quelles parties de la chronique étudiée correspondent les différents évènements mis en jeu, et il n'est donc pas évident de délimiter les bornes de l'absence.

Pour résoudre ce problème, nous avons introduit la partie droite de l'opérateur d'absence qui permet de conserver séparément les reconnaissances de D et les reconnaissances de $(A\ B) - [C[$ jusqu'à ce qu'elles soient complètes et puissent être combinées par la transition `Sub`. Le réseau

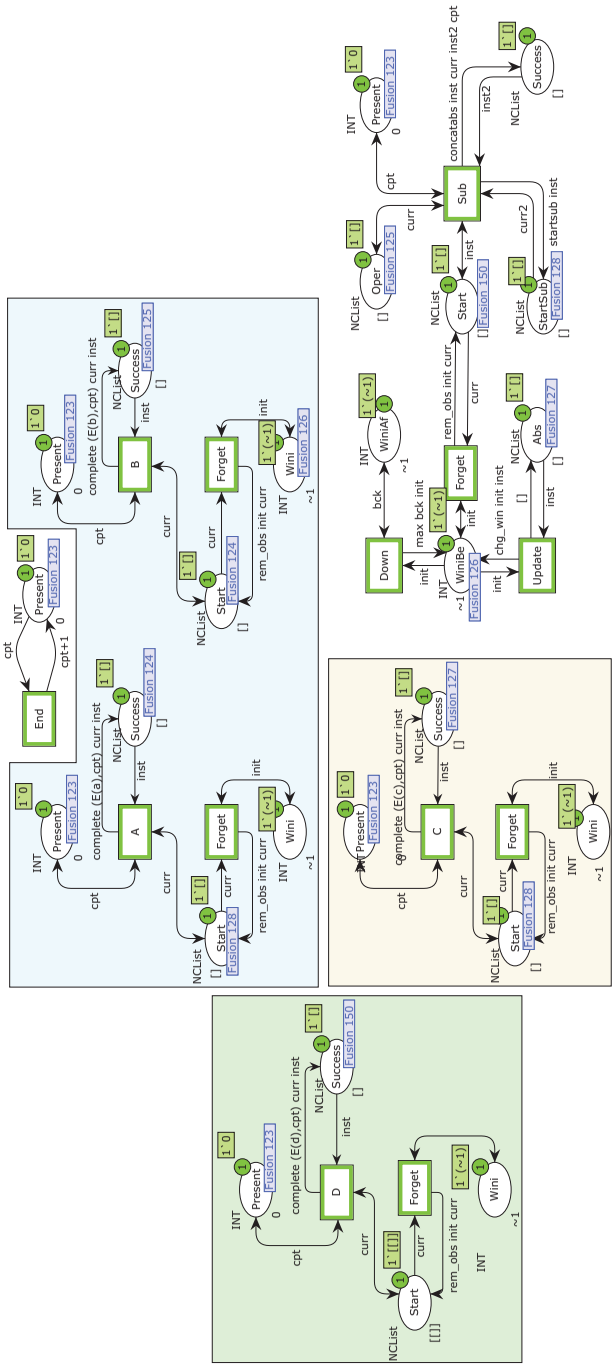


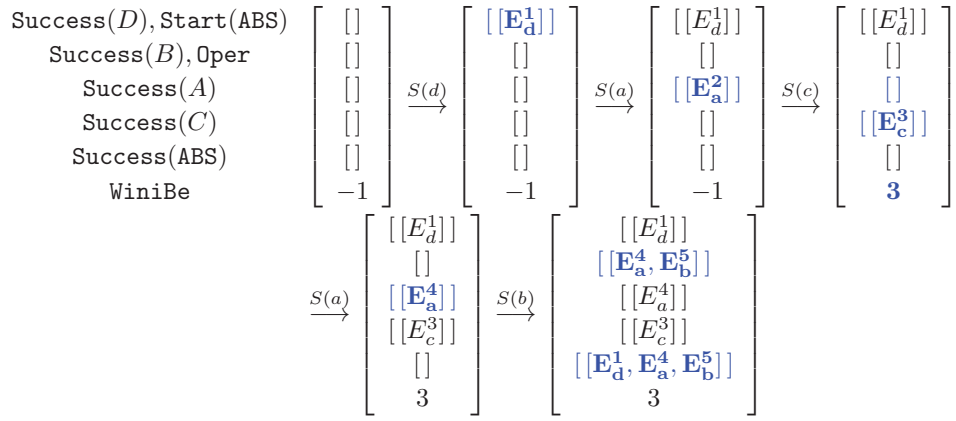
FIGURE 3.19 – Réseau correspondant à la chronique D $((A\ B) - [C[)$

reconnaissant la chronique D $((A\ B) - [C[)$ est présenté Figure 3.19 et l'exemple suivant illustre l'évolution du marquage du réseau dans ce cas.

Exemple 14. Considérons la chronique D $((A\ B) - [C[)$ et le flux $\varphi = ((\mathbf{d}, \mathbf{1}), (a, 2), (c, 3), (\mathbf{a}, \mathbf{4}), (\mathbf{b}, \mathbf{5}))$ où $a, b, c, d \in \mathfrak{N}$.

On ne détaille pas ici toute la suite des transitions à tirer mais on note $S(e)$ les transitions à tirer pour notre chronique suite à l'évènement e.

Le marquage des places du réseau évolue comme suit.



3.3.6 Définition formelle de la stratégie de tirage

Dans les réseaux de Petri modélisant le processus de reconnaissance de chroniques que nous venons de définir, toutes les transitions sont systématiquement tirables. Comme évoqué dans la Remarque 16 (Section 3.1.2), nous souhaitons obtenir un marquage précis pour un réseau à l'issue du traitement d'un évènement du flux. C'est dans ce marquage que l'on peut lire l'ensemble des reconnaissances de la chronique. Dans nos réseaux, toutes les transitions sont toujours tirables mais toute séquence de tirages ne mène pas toujours au marquage recherché. Nous allons donc définir formellement une stratégie de tirage des transitions, associée à chaque réseau, et donnant, pour chaque évènement du flux, la suite des transitions à tirer pour obtenir le marquage recherché à l'issue du traitement de chaque évènement. Notons que cette stratégie n'est pas unique et qu'elle ne permet pas de tirer parti de la concurrence des réseaux de Petri que nous évoquerons dans le chapitre suivant.

Définissons donc formellement la stratégie de tirage dont un aperçu a été donné dans la description précédente du fonctionnement des réseaux. Pour ce faire, nous devons tout d'abord définir une fonction auxiliaire $Forget_C$ qui correspond à la liste des transitions `Forget` de $N(C)$, convenablement ordonnée, que l'on tirera lors d'une absence.

Définition 27 (stratégie de tirage des transitions `Forget`). On définit par induction sur la chronique C la fonction auxiliaire $Forget_C$:

— Si $C = A \in \mathfrak{N}$, alors $Forget_C = [\texttt{Forget}]$

— Si $C = C_1\, C_2$, alors $Forget_C = Forget_{C_2} :: Forget_{C_1}$
— Si $C = C_1 \,||\, C_2$, alors $Forget_C = Forget_{C_2} :: Forget_{C_1}$
— Si $C = C_1 \& C_2$, alors $Forget_C = [\texttt{ForgetAnd}] :: Forget_{C_2} :: Forget_{C_1}$
— Si $C = (C_1) - [C_2]$, alors $Forget_C = Forget_{C_2} :: [\texttt{Down}] :: Forget_{C_1}$

À l'aide de cette stratégie de tirage des transitions `Forget`, on peut maintenant définir la stratégie de tirage générale pour tous nos réseaux.

Définition 28 (stratégie de tirage). On définit par induction sur la chronique C la *stratégie de tirage* S_C où, pour tout évènement $e \in \mathfrak{N}$, $S_C(e)$ correspond à la suite des transitions à tirer dans le réseau $N(C)$ pour traiter l'occurrence d'un évènement e.

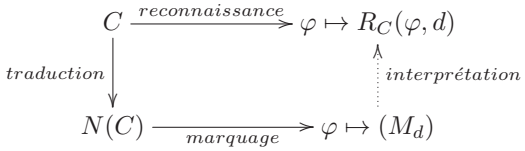
Pour ce faire, nous définissons une fonction auxiliaire S'_C qui donne toutes les transitions à tirer, exceptée la transition `End`. Celle-ci doit être tirée à la fin et une seule fois par évènement. Elle ne peut donc être intégrée à l'induction et il faut l'ajouter a posteriori.

Pour tout $e \in \mathfrak{N}$:
— Si $C = A \in \mathfrak{N}$, alors $S'_C(e) = \begin{cases} [\texttt{A}] & \text{si } e = A \\ [\,] & \text{sinon} \end{cases}$
— Si $C = C_1\, C_2$, alors $S'_C(e) = S'_{C_1}(e) :: S'_{C_2}(e)$
— Si $C = C_1 \,||\, C_2$, alors $S'_C(e) = S'_{C_1}(e) :: S'_{C_2}(e)$ [7]
— Si $C = C_1 \& C_2$, alors $S'_C(e) = [\texttt{Sub}] :: S'_{C_1}(e) :: S'_{C_2}(e) :: [\texttt{AND}]$
— Si $C = (C_1) - [C_2]$, alors $S'_C(e) = S'_{C_2}(e) :: [\texttt{Update}, \texttt{Down}, \texttt{Sub}] :: Forget_{C_1} :: S'_{C_1}(e) :: [\texttt{Sub}]$

Pour incrémenter finalement le compteur d'évènements, on pose maintenant, pour tout $e \in \mathfrak{N}$, $S_C(e) = S'_C(e) :: [\texttt{End}]$.

3.4 Démonstration de la correction du modèle « à un seul jeton »

Dans le Chapitre 2, nous avons défini, pour chaque chronique C, une fonction qui à un flux d'évènements φ et instant d associe l'ensemble $R_C(\varphi, d)$ des reconnaissances de la chronique dans ce flux (*reconnaissance* dans le diagramme ci-dessous). Dans les sections précédentes, nous avons modélisé la reconnaissance de chroniques en construisant formellement, pour chaque chronique C, un réseau de Petri $N(C)$ lui correspondant (*traduction*). En fonction du flux d'évènements φ, le marquage (M_d) de ce réseau évolue (*marquage*), et il est alors possible de lire dans celui-ci les reconnaissances de la chronique C dans le flux φ (*interprétation*). Ce système est représenté par le diagramme suivant :



7. Contrairement au cas de la séquence, l'ordre entre C_1 et C_2 est ici sans importance.

L'objectif est de démontrer que ce diagramme commute, c'est-à-dire que les reconnaissances obtenues par les réseaux de Petri correspondent effectivement exactement aux reconnaissances théoriques $R_C(\varphi, d)$.

On se propose de montrer ce résultat d'adéquation dans le cas restreint de l'ensemble des chroniques composées d'au plus une absence au plus haut niveau.

Nous allons commencer par rappeler la sémantique ensembliste avec laquelle nous souhaitons montrer l'adéquation des réseaux. Nous devons adapter la sémantique de la Définition 16 (p.55) car la sémantique opérationnelle fournie par les réseaux est plus limitée sur les trois aspects suivants :

(i) le processus de reconnaissance modélisé par nos chroniques s'applique à un langage restreint réduit aux opérateurs de séquence, disjonction, conjonction et absence, sans possibilité d'exprimer de contrainte sur des attributs d'évènements ;

(ii) dans le modèle en réseaux de Petri colorés, il n'y a pas de notion de temps continu comme les évènements sont ordonnés par le compteur d'évènement qui est discret ;

(iii) dans les deux formalismes, les reconnaissances ne sont pas modélisées de la même manière :
— dans la sémantique ensembliste arborescente du Chapitre 2, une reconnaissance est représentée par un arbre,
— dans le modèle en réseaux de Petri colorés, une reconnaissance est une liste d'évènements.

Nous allons donc adapter la sémantique du Chapitre 2 à ses trois points de la manière suivante :

(i) nous restreignons le contenu des ensembles de reconnaissances aux reconnaissances elles-mêmes, il n'est plus nécessaire de définir l'ensemble d'attributs X_r associé à une reconnaissance r, et nous nous limitons aux opérateurs modélisés ;

(ii) nous nous plaçons dans le modèle de temps discret des entiers $\mathbb{N}$;

(iii) nous adoptons une modélisation de reconnaissances sous la forme de multi-ensembles qui sont directement en correspondance avec les listes des réseaux.

Nous obtenons ainsi la sémantique suivante :

Définition 29 (ensembles de reconnaissances des chroniques modélisés en réseaux de Petri). Soit $C \in \mathfrak{X}$ une chronique. L'*ensemble des reconnaissances de C sur le flux d'évènements φ jusqu'à la date d* est noté $R_C(\varphi, d)$ et est défini par induction comme suit, où $d \in \mathbb{N}$, $A \in \mathfrak{N}$ et $C_1, C_2 \in \mathfrak{X}$:

— $R_A(\varphi, d) = \{\{(e,t)\} : e = A \wedge \exists i\ \varphi(i) = (e,t) \wedge t \leq d\}$
— $R_{C_1||C_2}(\varphi, d) = R_{C_1}(\varphi, d) \cup R_{C_2}(\varphi, d)$
— $R_{C_1\ C_2}(\varphi, d) = \{r_1 \uplus r_2 : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_2}(\varphi, d) \wedge T_{\max}(r_1) < T_{\min}(r_2)\}$
— $R_{C_1 \& C_2}(\varphi, d) = \{r_1 \uplus r_2 : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_2}(\varphi, d)\}$
— $R_{(C_1)-[C_2[}(\varphi, d) = \{r_1 : r_1 \in R_{C_1}(\varphi, d)$
$\wedge \forall r_2 \in R_{C_2}(\varphi, d)\ (\ T_{\min}(r_1) > T_{\min}(r_2) \vee T_{\max}(r_1) \leq T_{\max}(r_2)\ )\}$

Nous allons d'abord montrer un petit lemme reformulant l'expression de l'ensemble des reconnaissances de la séquence, ce qui permet de simplifier les démonstrations qui suivent.

Lemme 1. *Soit C_1 et C_2 deux chroniques. Soit $i \in \mathbb{R}$ et φ un flux d'évènements. Alors :*

$$R_{C_1\,C_2}(\varphi, i) = \{r_1 \uplus r_2 : 0 < d < j \wedge r_1 \in R_{C_1}(\varphi, j) \wedge T_{\min}(r_2) = j \wedge r_2 \in R_{C_2}(\varphi, i)\}$$

Démonstration. Par définition :
$R_{C_1\,C_2}(\varphi, i) = \{r_1 \uplus r_2 : r_1 \in R_{C_1}(\varphi, i) \wedge r_2 \in R_{C_2}(\varphi, i) \wedge T_{\max}(r_1) < T_{\min}(r_2)\}$.

On pose $j = T_{\min}(r_2) + 1$.

Le résultat découle de l'équivalence :
$(r_1 \in R_{C_1}(\varphi, i) \wedge T_{\max}(r_1) < T_{\min}(r_2)) \Leftrightarrow (0 < j < i \wedge r_1 \in R_{C_1}(\varphi, j-1) \wedge T_{\min}(r_2) = j)$ □

Pour démontrer le résultat d'adéquation nous aurons besoin de nous référer à la notion intuitive de sous-chronique que nous formalisons ci-dessous.

Définition 30 (ensemble des sous-chroniques d'une chronique). L'ensemble des *sous-chroniques* d'une chronique C est le plus petit ensemble E tel que :

- $C \in E$.
- si $(C_1\,C_2) \in E$, alors $C_1 \in E$ et $C_2 \in E$.
- si $(C_1\,||\,C_2) \in E$, alors $C_1 \in E$ et $C_2 \in E$.
- si $(C_1 \& C_2) \in E$, alors $C_1 \in E$ et $C_2 \in E$.
- si $((C_1) - [C_2]) \in E$, alors $C_1 \in E$ et $C_2 \in E$.

Nous introduisons maintenant une notation qui nous permettra de nous référer formellement au marquage de nos réseaux.

Définition 31 (notation de marquage). Soit C_1 une chronique et soit C_2 une sous-chronique de C_1. Soit $\varphi = (u_i)_{i \in I}$ un flux d'évènements. Pour tout $i \in I$, on définit $M_i^{\mathtt{p}(C_2)}(C_1)$ comme le marquage, dans le réseau $N(C_1)$, de la place $\mathtt{p}(C_2)$ avant le tirage des transitions correspondant au i^e évènement.

Lorsque $C_1 = C_2$, on notera $M_i^{\mathtt{p}}(C_1)$ pour $M_i^{\mathtt{p}(C_1)}(C_1)$.

Remarque 19. $M_i^{\mathtt{p}(C_2)}(C_1)$ ne correspond pas forcément à $M_i^{\mathtt{p}(C_2)}(C_2)$. En effet, dans $M_i^{\mathtt{p}(C_2)}(C_1)$, on considère une place dénotée $\mathtt{p}(C_2)$ du sous-réseau associé à C_2 dans le réseau $N(C_1)$, il peut donc y avoir des transitions dans $N(C_1)$ non relatives à C_2 qui modifient le contenu des places de C_2.

C'est typiquement le cas des places `Start` dans le cadre d'une séquence dès le marquage initial (c'est-à-dire pour $i = 1$) : on a $M_1^{\mathtt{Start}(C_2)}(C_1\ C_2) = 1`[\ ]$ alors que $M_1^{\mathtt{Start}(C_2)}(C_2) = 1`[\ [\ ]\ ]$.

Commençons par montrer le lemme suivant qui nous permettra ensuite de déduire le théorème d'adéquation recherché.

Lemme 2. *Soit C une chronique sans absence.*

Soit $k \in \mathbb{N}$. Soit $\varphi = (u_i)_{i \in [\![1,k]\!]}$ un flux d'évènements.

On suppose que les marquages successifs des places $\mathtt{Start}(C)$ et $\mathtt{Wini}(C)$ sont respectivement $M_i^{\mathtt{Start}}(C)$ et $M_i^{\mathtt{Wini}}(C)$ pour $i \in [\![1, k+1]\!]$.

Alors les marquages de la place `Success`(C) *sont, pour tout* $i \in [\![1, k+1]\!]$ *:*

$$M_i^{\texttt{Success}}(C) = \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(C) \wedge r' \in R_C(\varphi, i-1) \wedge T_{\min}(r') = h \tag{3.2}$$

$$\wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C) < j_1) \tag{3.3}$$

$$\wedge r \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r), T_{\min}(r')[\![\ M_l^{\texttt{Wini}}(C) < T_{\min}(r)\} \tag{3.4}$$

Commençons par donner une compréhension intuitive du sens de ce lemme. Tout d'abord, comme l'on souhaite effectuer une démonstration par induction, il est naturel de prendre des marquages quelconques des places `Start` et `Wini`, pour pouvoir appliquer l'hypothèse d'induction dans toutes les configurations que nous avons dans nos réseaux. D'autre part, l'équation donnant le marquage de la place `Success` s'interprète comme suit :

(3.2) r' est une reconnaissance de C qui vient compléter la reconnaissance partielle r (pouvant être vide, selon les cas) présente dans `Start` ;

(3.3) entre tous indices j_1 et j_2 consécutifs dans r', il n'y a pas eu de marquage de `Wini` supérieur ou égal à j_1, ce qui signifie que r' n'a pu être supprimée par un mécanisme d'absence ;

(3.4) entre $T_{\max}(r)$ et $T_{\min}(r')$ le marquage de `Wini` ne dépasse pas $T_{\max}(r)$, donc r n'a pu être supprimée par un mécanisme d'absence avant d'avoir été complétée.

Démonstration. Montrons ce lemme par induction sur la chronique C. Soit $k \in \mathbb{N}$.

Cas 1 : $C = A \in \mathfrak{N}$

Procédons par récurrence sur i en montrant d'abord que :
$M_i^{\texttt{Success}}(A) = \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(A) \wedge r' \in R_A(\varphi, i-1) \wedge T_{\min}(r') = h\}$.

— $M_1^{\texttt{Success}}(A) = \emptyset = \bigcup_{0<h<1} \{r \uplus r' : r \in M_h^{\texttt{Start}}(A) \wedge r' \in R_A(\varphi, i-1) \wedge T_{\min}(r') = h\}$

— Par hypothèse de récurrence, on a que :

$$M_i^{\texttt{Success}}(A) = \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(A) \wedge r' \in R_A(\varphi, i-1) \wedge T_{\min}(r') = h\}$$
$$= \bigcup_{0<h<i} \{r \uplus \{(a,h)\} : r \in M_h^{\texttt{Start}}(A)\}$$

On a donc :

$$M_{i+1}^{\texttt{Success}}(A) = M_i^{\texttt{Success}}(A) \uplus \{r \uplus \{(a,i)\} : r \in M_i^{\texttt{Start}}(A)\}$$
$$= \bigcup_{0<h<i+1} \{r \uplus r' : r \in M_h^{\texttt{Start}}(A) \wedge r' \in R_A(\varphi, i) \wedge T_{\min}(r') = h\}$$

Par principe de récurrence, on a donc bien que, pour tout $i \in [\![1, k+1]\!]$,
$M_i^{\texttt{Success}}(A) = \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(A) \wedge r' \in R_A(\varphi, i-1) \wedge T_{\min}(r') = h\}$.

De plus, toute reconnaissance de la chronique simple A est un singleton, donc pour tout $r' \in R_A(\varphi, i-1)$ la proposition suivante est toujours vérifiée :
$\forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(A) < j_1)$

De plus, la fonction rem_obs assure que, dès que la transition `Forget` est tirée, $\forall r \in M_h^{\texttt{Start}}(A) \ (r = \emptyset \vee T_{\min}(r) > M_h^{\texttt{Wini}}(A))$.

On a donc bien le résultat, pour tout $i \in [\![1, k+1]\!]$.

Cas 2 : $C = C_1 \,||\, C_2$

Soit $i \in [\![1, k+1[\![$.

Par construction des réseaux, on prend $M_i^{\texttt{Start}}(C) = M_i^{\texttt{Start}}(C_1) = M_i^{\texttt{Start}}(C_2)$,
donc $M_i^{\texttt{Success}}(C) = M_i^{\texttt{Success}}(C_1) \cup M_i^{\texttt{Success}}(C_2)$.
De plus, $M_i^{\texttt{Wini}}(C) = \max\{M_i^{\texttt{Wini}}(C_1), M_i^{\texttt{Wini}}(C_2)\}$.
On peut alors appliquer l'hypothèse de récurrence à C_1 et C_2, ce qui nous donne :

$$\begin{aligned}
M_i^{\texttt{Success}}(C) = \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(C_1) \wedge r' \in R_{C_1}(\varphi, i-1) \wedge T_{\min}(r') = h \\
\wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_1) < j_1) \\
\wedge r \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r), T_{\min}(r')[\![\ M_l^{\texttt{Wini}}(C_1) < T_{\min}(r)\} \\
\uplus \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(C_2) \wedge r' \in R_{C_2}(\varphi, i-1) \wedge T_{\min}(r') = h \\
\wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_2) < j_1) \\
\wedge r \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r), T_{\min}(r')[\![\ M_l^{\texttt{Wini}}(C_2) < T_{\min}(r)\} \\
= \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(C) \wedge r' \in R_{C_1||C_2}(\varphi, i-1) \wedge T_{\min}(r') = h \\
\wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C) < j_1) \\
\wedge r \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r), T_{\min}(r')[\![\ M_l^{\texttt{Wini}}(C) < T_{\min}(r)\}
\end{aligned}$$

Cas 3 : $C = C_1 \, C_2$

Soit $i \in [\![1, k+1[\![$. Soit $j \in [\![1, i[\![$.

Par construction des réseaux, on prend $M_j^{\texttt{Start}}(C) = M_j^{\texttt{Start}}(C_1)$,
et on a $M_j^{\texttt{Wini}}(C) = \max\{M_j^{\texttt{Wini}}(C_1), M_j^{\texttt{Wini}}(C_2)\}$.
D'où, par hypothèse de récurrence sur C_1 :

$$\begin{aligned}
M_j^{\texttt{Success}}(C_1) = \bigcup_{0<h<j} \{r_1 \uplus r_1' : r_1 \in M_h^{\texttt{Start}}(C) \wedge r_1' \in R_{C_1}(\varphi, j-1) \wedge T_{\min}(r_1') = h \\
\wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r_1' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_1) < j_1) \\
\wedge r_1 \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r_1), T_{\min}(r_1')[\![\ M_l^{\texttt{Wini}}(C_1) < T_{\min}(r_1)\}.
\end{aligned}$$

De plus, $M_i^{\texttt{Start}}(C_2) = M_i^{\texttt{Success}}(C_1)$ et $M_i^{\texttt{Success}}(C) = M_i^{\texttt{Sucess}}(C_2)$, donc par hypothèse de récurrence sur C_2 :

$$\begin{aligned}
M_i^{\texttt{Success}}(C) = \bigcup_{0<j<i} \{r_2 \uplus r_2' : r_2 \in M_j^{\texttt{Success}}(C_1) \wedge r_2' \in R_{C_2}(\varphi, i-1) \wedge T_{\min}(r_2') = j \\
\wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r_2' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_2) < j_1) \\
\wedge r_2 \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r_2), T_{\min}(r_2')[\![\ M_l^{\texttt{Wini}}(C_2) < T_{\min}(r_2)\} \\
= \{r_1 \uplus r_1' \uplus r_2' : 0 < j < i \wedge 0 < h < j \wedge r_1 \in M_h^{\texttt{Start}}(C) \wedge r_1' \in R_{C_1}(\varphi, j-1) \\
\wedge T_{\min}(r_1') = h \wedge T_{\min}(r_2') = j \wedge r_2' \in R_{C_2}(\varphi, i-1) \\
\text{(1a)} \quad \wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r_1' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_1) < j_1) \\
\text{(1b)} \quad \wedge r_1 \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r_1), T_{\min}(r_1')[\![\ M_l^{\texttt{Wini}}(C_1) < T_{\min}(r_1) \\
\text{(2a)} \quad \wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r_2' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_2) < j_1)
\end{aligned}$$

$$\text{(2b)} \qquad \wedge\, r_2 \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r_2), T_{\min}(r'_2)[\![\ M_l^{\texttt{Wini}}(C_2) < T_{\min}(r_2)\}$$

Remarquons maintenant que (1a) $\wedge$ (1b) $\wedge$ (2a) $\wedge$ (2b) est équivalente à
$\forall j_1 \forall j_2\ ([\![j_1, j_2]\!] \cap (r'_1 \uplus r'_2) = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C) < j_1)$
$\wedge\, r_1 \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r_1), T_{\min}(r'_1 \uplus r'_2)[\![\ M_l^{\texttt{Wini}}(C) < T_{\min}(r_1)$
En effet :

(i) L'implication ($\Leftarrow$) est immédiate.

(ii) Pour la réciproque ($\Rightarrow$) :
- Montrons d'abord que $\forall j_1 \forall j_2\ ([\![j_1, j_2]\!] \cap (r'_1 \uplus r'_2) = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C) < j_1)$.
  Soit j_1 et j_2 tels que $[\![j_1, j_2]\!] \cap (r'_1 \uplus r'_2) = \{j_1, j_2\}$.
  Si $\{j_1, j_2\} \subset r'_1$ ou $\{j_1, j_2\} \subset r'_2$, on a le résultat par (1a) ou (2a).
  Sinon, comme $T_{\max}(r'_1) < T_{\min}(r'_2)$, la seule possibilité est $j_1 = T_{\max}(r'_1)$ et $j_2 = T_{\min}(r'_2)$, ce qui est traité par (2b).
- (1b) et le fait que $T_{\min}(r'_1 \uplus r'_2) = T_{\min}(r'_1)$ donnent que :
  $r_1 \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r_1), T_{\min}(r'_1 \uplus r'_2)[\![\ M_l^{\texttt{Wini}}(C) < T_{\min}(r_1)$.

Remarquons aussi que $\begin{cases} (0 < j < i \wedge 0 < h < j) \Rightarrow 0 < h < i \\ (T_{\min}(r'_1) = h \wedge r'_1 \in R_{C_1}(\varphi, j-1)) \Rightarrow 0 < h < j \end{cases}$
On a donc que :
$(0 < j < i \wedge 0 < h < j \wedge r'_1 \in R_{C_1}(\varphi, i-1) \wedge T_{\min}(r'_1) = h)$
$\Leftrightarrow (0 < j < i \wedge 0 < h < i \wedge r'_1 \in R_{C_1}(\varphi, i-1) \wedge T_{\min}(r'_1) = h)$
D'où :
$$\begin{aligned} M_i^{\texttt{Success}}(C) = \{r_1 \uplus r'_1 \uplus r'_2 : {} & 0 < j < i \wedge 0 < h < i \wedge r_1 \in M_h^{\texttt{Start}}(C) \wedge r'_1 \in R_{C_1}(\varphi, j-1) \\ & \wedge\, T_{\min}(r'_1) = h \wedge T_{\min}(r'_2) = j \wedge r'_2 \in R_{C_2}(\varphi, i-1) \\ & \wedge \forall j_1 \forall j_2\ ([\![j_1, j_2]\!] \cap (r'_1 \uplus r'_2) = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C) < j_1 \\ & \wedge\, r_1 \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r'_1 \uplus r'_2), T_{\min}(r'_1 \cup r'_2)[\![\ M_l^{\texttt{Wini}}(C) < T_{\min}(r_1)\} \end{aligned}$$
En appliquant le Lemme 1 et en posant $r' = r'_1 \uplus r'_2$ on obtient alors le résultat.

Cas 4 : $C = C_1 \& C_2$

La définition ensembliste de la fonction mergeAndNew est la suivante :
$\text{mergeAndNew}(r_0, r_1, r_2, r) = r \uplus \{a \uplus b \uplus c : a \in r_0 \wedge b \in r_1 \wedge c \in r_2\}$.
Procédons par récurrence sur i.
- $M_1^{\texttt{Success}}(C) = \emptyset$.
- $M_{i+1}^{\texttt{Success}}(C)$
  $= M_i^{\texttt{Success}}(C) \uplus \{r \uplus a \uplus b : r \in M_i^{\texttt{Start}}(C) \wedge a \in M_i^{\texttt{Success}}(C_1) \wedge b \in M_i^{\texttt{Success}}(C_2)\}$
  $$\begin{aligned} = M_i^{\texttt{Success}}(C) \uplus \{r \uplus r'_1 \uplus r'_2 : {} & r \in M_i^{\texttt{Start}}(C) \\ & \wedge\, 0 < h < i+1 \wedge r'_1 \in R_{C_1}(\varphi, i) \wedge T_{\min}(r'_1) = h \\ & \wedge\, 0 < k < i+1 \wedge r'_2 \in R_{C_2}(\varphi, i) \wedge T_{\min}(r'_2) = k \\ & \wedge \forall j_1 \forall j_2\ ([\![j_1, j_2]\!] \cap r'_1 = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_1) < j_1) \\ & \wedge \forall j_1 \forall j_2\ ([\![j_1, j_2]\!] \cap r'_2 = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_2) < j_1)\} \end{aligned}$$
  Soit j_1 et j_2 tels que $[\![j_1, j_2]\!] \cap (r'_1 \uplus r'_2) = \{j_1, j_2\}$.
  - si $\{j_1, j_2\} \subset r'_1$ ou $\{j_1, j_2\} \subset r'_2$, alors $\forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_1) < j_1$;
  - si $j_1 = T_{\max}(r'_1)$ et $j_2 = T_{\min}(r'_2)$: par l'absurde, supposons que $\exists l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(C_1) \geq j_1$, alors la reconnaissance r'_1, terminée à j_1, est supprimée par le tirage de la tran-

sition `Forget` avant que la reconnaissance r'_2 ne soit terminée, celle-ci ne peut donc plus compléter r'_1 lors du tirage de la transition `And` ;
— si $j_1 \neq T_{\max}(r'_1)$ alors $\exists j'_2 \in r'_1$ $j'_2 > j_2$ (car j_1 et j_2 sont consécutifs), donc $\forall l \in [\![j_1, j'_2[\![$ $M_l^{\texttt{Wini}}(C_1) < j_1$;
— si $j_1 = T_{\max}(r'_1)$ et $j_2 \neq T_{\min}(r'_2)$, comme j_1 et j_2 sont consécutifs, $T_{\min}(r'_2) < j_1$, donc $\forall l \in [\![T_{\min}(r'_2), j_2[\![$ $M_l^{\texttt{Wini}}(C_1) < j_1$ et $[\![j_1, j_2[\![\subseteq [\![T_{\min}(r'_2), j_2[\![$.

De plus, de même que dans le Cas 1, la fonction rem_obs assure que, dès que la transition `Forget` est tirée, $\forall r \in M_h^{\texttt{Start}}(A)$ $(r = \emptyset \vee \max r > M_h^{\texttt{Wini}}(A))$.
On a donc le résultat en posant $r' = r'_1 \uplus r'_2$.

□

Nous allons maintenant étendre ce lemme aux chroniques ayant au plus une absence au plus haut niveau, à l'aide du concept suivant :

Définition 32 (partie positive d'une chronique). On définit par induction sur la chronique C l'ensemble $\mathrm{Pos}(C)$:
- Si $C = A \in \mathfrak{N}$ ou $C = C_1\ C_2$ ou $C = C_1||C_2$ ou $C = C_1 \& C_2$, alors $\mathrm{Pos}(C) = C$.
- Si $C = (C_1) - [C_2[$, alors $\mathrm{Pos}(C) = C_1$.

Pour toute chronique sans absence, $\mathrm{Pos}(C) = C$, donc on peut transcrire le Lemme 2 en se plaçant dans les mêmes conditions comme suit :

$$M_i^{\texttt{Success}}(C) = \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(\mathrm{Pos}(C)) \wedge r' \in R_{\mathrm{Pos}(C)}(\varphi, i-1) \wedge T_{\min}(r') = h$$
$$\wedge \forall j_1 \forall j_2\ ([\![j_1, j_2]\!] \cap r' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(\mathrm{Pos}(C)) < j_1)$$
$$\wedge r \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r), T_{\min}(r')[\![\ M_l^{\texttt{Wini}}(\mathrm{Pos}(C)) < T_{\min}(r)\}$$

De plus, si $C = (C_1) - [C_2[$, $M_i^{\texttt{Success}}(C) = M_i^{\texttt{Success}}(C_1)$ si l'on prend comme hypothèse les marquages $M_i^{\texttt{Start}}(C_1) = M_i^{\texttt{Start}}(C_1)$ et $M_i^{\texttt{Wini}}(C_1) = M_i^{\texttt{Wini}(C_1)}(C)$. On peut donc étendre le lemme :

Lemme 3. *Soit C une chronique sans absence ou une chronique de la forme $(C_1) - [C_2[$ avec C_1 et C_2 des chroniques sans absence.*

Soit $k \in \mathbb{N}$. Soit $\varphi = (u_i)_{i \in [\![1,k]\!]}$ un flux d'évènements.

On suppose que les marquages successifs des places $\texttt{Start}(C)$ et $\texttt{Wini}(C)$ sont respectivement $M_i^{\texttt{Start}}(C)$ et $M_i^{\texttt{Wini}}(C)$ pour $i \in [\![1, k+1]\!]$.

Alors les marquages de la place $\texttt{Success}(C)$ sont, pour tout $i \in [\![1, k+1]\!]$:

$$M_i^{\texttt{Success}}(C) = \bigcup_{0<h<i} \{r \uplus r' : r \in M_h^{\texttt{Start}}(\mathrm{Pos}(C)) \wedge r' \in R_{\mathrm{Pos}(C)}(\varphi, i) \wedge T_{\min}(r') = h$$
$$\wedge \forall j_1 \forall j_2\ ([\![j_1, j_2]\!] \cap r' = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_l^{\texttt{Wini}}(\mathrm{Pos}(C)) < j_1)$$
$$\wedge r \neq \emptyset \Rightarrow \forall l \in [\![T_{\max}(r), T_{\min}(r')[\![\ M_l^{\texttt{Wini}}(\mathrm{Pos}(C)) < T_{\min}(r)\}$$

Ce lemme nous permet de montrer le théorème recherché :

Théorème 1 (Théorème d'adéquation). *Soit C une chronique sans absence ou une chronique de la forme $(C_1) - [C_2[$ avec C_1 et C_2 des chroniques sans absence.*

Soit $k \in \mathbb{N}$. Soit $\varphi = (u_i)_{i\in[\![1,k]\!]}$ un flux d'évènements.
Alors les marquages de la place $\texttt{Success}(C)$ *sont, pour tout $i \in [\![1,k]\!]$:*

$$M_{i+1}^{\texttt{Success}}(C) = R_C(\varphi, i)$$

Démonstration. Soit $k \in \mathbb{N}$. Soit $\varphi = (u_i)_{i\in[\![1,k]\!]}$ un flux d'évènements.
Raisonnons par disjonction de cas.

Cas 1 : Si C est une chronique sans absence.

Alors $\text{Pos}(C) = C$ et, pour tout $i \in [\![1,k]\!]$, $M_{i+1}^{\texttt{Start}}(C) = \emptyset$ et $M_{i+1}^{\texttt{Wini}}(C) = -1$.

On a donc bien, pour $i \in [\![1,k]\!]$:
$$\begin{aligned} M_{i+1}^{\texttt{Success}}(C) &= \bigcup_{0<h<i+1} \{r' : r' \in R_C(\varphi, i) \wedge T_{\min}(r') = h\} \\ &= R_C(\varphi, i) \end{aligned}$$

Cas 2 : Si $C = (C_1) - [C_2[$ où C_1 et C_2 sont des chroniques sans absence.

D'après le Lemme 3, et comme pour tout $h \in [\![1,k]\!]$, $M_{h+1}^{\texttt{Start}}(C) = \emptyset$ on a, pour $i \in [\![1,k]\!]$:
$$\begin{aligned} M_{i+1}^{\texttt{Success}}(C) = \bigcup_{0<h<i+1} \{r_1 : r_1 \in R_{C_1}(\varphi, i) \wedge T_{\min}(r') = h \\ \wedge \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r_1 = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_{l+1}^{\texttt{Wini}}(C_1) < j_1)\} \end{aligned}$$

Montrons que, pour $r_1 \in R_{C_1}(\varphi, i)$:

$$\begin{aligned} & \forall j_1 \forall j_2 \ ([\![j_1, j_2]\!] \cap r_1 = \{j_1, j_2\} \Rightarrow \forall l \in [\![j_1, j_2[\![\ M_{l+1}^{\texttt{Wini}}(C_1) < j_1) && (3.5) \\ \iff \quad & \forall r_2 \in R_{C_2}(\varphi, i) \ (T_{\min}(r_1) > T_{\min}(r_2) \vee T_{\max}(r_1) \leq T_{\max}(r_2)) && (3.6) \end{aligned}$$

($\Rightarrow$) Par l'absurde, on suppose que $\exists r_2 \in R_{C_2}(\varphi, i) \ (T_{\min}(r_1) \leq T_{\min}(r_2) \wedge T_{\max}(r_1) > T_{\max}(r_2))$,
alors, comme $T_{\min}(r_1) \leq T_{\min}(r_2) \leq T_{\max}(r_2) < T_{\max}(r_1)$,
on peut trouver j_1 et j_2 consécutifs dans r_1 et tels que $T_{\max}(r_2) \in [\![j_1, j_2[\![$,
donc, si on applique (3.5), $M_{T_{\max}(r_2)+1}^{\texttt{Wini}}(C_1) < j_1$,
donc $T_{\max}(r_2) < j_1$, absurde.

($\Leftarrow$) Par contraposée, on suppose que $\exists j_1 \ \exists j_2 \ ([\![j_1, j_2]\!] \cap r_1 = \{j_1, j_2\} \wedge \exists l \in [\![j_1, j_2[\![\ M_{l+1}^{\texttt{Wini}}(C_1) \geq j_1$,
or $M_{l+1}^{\texttt{Wini}}(C_1) \geq j_1 \iff \exists r_2 \in R_{C_2}(\varphi, l) \ T_{\min}(r_2) \geq j_1$,
donc $\exists r_2 \in R_{C_2}(\varphi, i) \ (T_{\min}(r_2) \geq j_1 \geq T_{\min}(r_1) \wedge T_{\max}(r_2) < j_2 \leq T_{\max}(r_1))$

On a donc bien montré que (3.5) $\iff$ (3.6),
donc $M_{i+1}^{\texttt{Success}}(C) = R_C(\varphi, i)$.

□

Nous avons donc bien montré l'adéquation entre la sémantique dénotationnelle du Chapitre 2 et la sémantique opérationnelle en réseaux de Petri colorés pour les constructions faisant intervenir au plus une absence au plus haut niveau.

3.5 Étude de la taille des réseaux

Nous allons achever ce chapitre par une étude de la taille des réseaux que nous avons construits. Pour ce faire, il nous faut d'abord introduire la notion de complexité d'une chronique et celle de la taille d'un réseau comme suit :

Définition 33 (mesure de complexité d'une chronique). La *mesure de complexité* d'une chronique est un entier défini par induction :

— $C = A \in \mathfrak{N}$ est de mesure de complexité 1.

— $C = C_1\, C_2$, $C = C_1 \mid\mid C_2$, $C = C_1 \& C_2$, et $C = (C_1) - [C_2[$ sont de mesure de complexité $n_1 + n_2$ où n_1 et n_2 sont respectivement les mesures de complexité de C_1 et C_2.

Définition 34 (taille d'un réseau). On définit la *taille* d'un réseau de Petri par un triplet (p, t, a) dont les éléments correspondent respectivement au nombre de places, nombre de transitions, et nombre d'arcs du réseau.

Soit C une chronique. On notera par le triplet $(p(C), t(C), a(C))$ la taille du réseau $N(C)$ associé.

Propriété 7. *La taille des réseaux associés aux chroniques s'exprime comme suit par induction :*

— *Si* $C = A \in \mathfrak{N}$, *alors* $(p(A), t(A), a(A)) = (4, 3, 9)$.

— *Si* $C = C_1\ C_2$, *alors*
$(p(C), t(C), a(C)) = (p(C_1) + p(C_2) - 3, t(C_1) + t(C_2) - 1, a(C_1) + a(C_2) - 2)$.

— *Si* $C = C_1 \mid\mid C_2$, *alors*
$(p(C), t(C), a(C)) = (p(C_1) + p(C_2) - 5, t(C_1) + t(C_2) - 1, a(C_1) + a(C_2) - 2)$.

— *Si* $C = C_1 \& C_2$, *alors*
$$\begin{aligned}(p(C), t(C), a(C)) &= (p(C_1) + p(C_2) - 4 + 2, t(C_1) + t(C_2) - 1 + 3, a(C_1) + a(C_2) - 2 + 15)\\ &= (p(C_1) + p(C_2) - 2, t(C_1) + t(C_2) + 2, a(C_1) + a(C_2) + 13).\end{aligned}$$

— *Si* $C = (C_1) - [C_2[$, *alors*
$$\begin{aligned}(p(C), t(C), a(C)) &= (p(C_1) + p(C_2) - 2 + 3, t(C_1) + t(C_2) - 1 + 4, a(C_1) + a(C_2) - 2 + 17)\\ &= (p(C_1) + p(C_2) + 1, t(C_1) + t(C_2) + 3, a(C_1) + a(C_2) + 15).\end{aligned}$$

La taille des réseaux est donc linéaire en fonction de la complexité de la chronique.

Formellement, pour toute $C \in \mathfrak{X}$, $(p(C), t(C), a(C)) = (\theta(n), \theta(n), \theta(n))$ *où* n *est la mesure de complexité de la chronique* C.

3.6 Conclusion

Dans ce chapitre, nous avons développé un modèle de reconnaissance de chroniques en réseaux de Petri colorés, dit « à un seul jeton » car les ensembles de reconnaissances sont représentés par un unique jeton contenant la liste des reconnaissances. Pour chaque chronique, nous avons construit un réseau associé par induction en composant les réseaux correspondant aux sous-chroniques. Le marquage de ce réseau évolue en fonction du flux d'évènements à analyser et, après le traitement de chaque évènement, on peut lire l'ensemble des reconnaissances de la chronique dans l'une des places du réseau. Un tel modèle avait déjà été présenté dans [CCK11], mais la construction des réseaux

n'y était pas complètement formalisée et des problèmes subsistaient pour certaines compositions des réseaux de conjonction et d'absence.

Nous avons donc complété la formalisation de la construction de ces réseaux, et nous les avons fait évoluer de façon à ce que toutes les compositions de réseaux produisent les bonnes reconnaissances. Pour ce faire, nous avons introduit la notion de structure générale du réseau, qui fournit les places principales que nous fusionnons lors du processus de construction, et nous avons enrichi les opérateurs de conjonction et d'absence afin de pouvoir traiter tous les cas de composition possibles.

Nous avons également formalisé le fonctionnement de nos réseaux. En effet, dans ce modèle, l'ensemble des transitions des réseaux sont en permanence tirables mais toute séquence de transitions ne mène pas au résultat voulu, à savoir l'ensemble correct des reconnaissances de la chronique étudiée. Nous avons donc défini par induction une stratégie de tirage que nous avons illustrée, et qui définit les suites de transitions à tirer en fonction de l'évènement du flux à traiter.

Pour finir, nous avons démontré la correction de nos réseaux, lorsque la stratégie de tirage est suivie, pour les constructions faisant intervenir au plus une absence au plus haut niveau, c'est-à-dire que nous avons montré que les reconnaissances fournies par nos réseaux correspondent bien à la sémantique dénotationnelle que nous avons posée dans le Chapitre 2.

Nous disposons donc maintenant d'un modèle de reconnaissance de chroniques en réseaux de Petri colorés entièrement formalisé mais non autonome dans le sens où il faut manuellement tirer les transitions pour suivre la stratégie de tirage définie et obtenir les ensembles de reconnaissances corrects. Dans le Chapitre 4, nous allons faire évoluer ce modèle afin d'y intégrer la stratégie de tirage et la gestion du flux d'évènements, et obtenir ainsi un modèle qui peut fonctionner en autonomie et sur lequel nous pouvons utiliser des outils d'analyse de réseaux de Petri afin de mettre en avant certaines caractéristiques.

Chapitre 4

Un modèle de reconnaissance contrôlé en réseaux de Petri colorés

Sommaire

Dans le Chapitre 3, nous avons construit formellement des réseaux de Petri colorés modélisant le processus de reconnaissance de chroniques. Un réseau est associé à chaque chronique et une stratégie de tirage formelle est définie. Elle indique la suite adaptée des transitions à tirer pour obtenir le marquage dans lequel l'ensemble des reconnaissances de la chronique peut être lu. Une telle stratégie de tirage est nécessaire car l'ensemble des transitions de nos réseaux est toujours tirable mais toute séquence de transitions ne mène pas au résultat voulu. L'objectif de ce chapitre est de faire évoluer notre modèle de reconnaissance pour intégrer dans nos réseaux un système de contrôle sur le tirage des transitions et les rendre ainsi autonomes. Celui-ci doit permettre

de s'affranchir de la stratégie de tirage, et ce en gérant directement le flux d'évènements et en n'activant que certaines transitions au fur et à mesure.

Pour l'élaboration de ces évolutions, nous nous fixons trois contraintes importantes :

— conserver un système de réseaux *modulaire* pour pouvoir maintenir une construction inductive du modèle calquée sur la structure du langage ;

— pour remplir notre objectif, garantir la « *confluence* » vers un unique marquage à l'issue du traitement autonome de chaque évènement du flux et donc assurer l'obtention de l'ensemble de reconnaissance correct à chaque étape ;

— en même temps, préserver un maximum de *concurrence* dans le tirage des transitions associées à chaque évènement.

En d'autres termes, nous établissons une méthode pour obtenir un marquage unique qui corresponde à chaque évènement traité, tout en maintenant un maximum de concurrence quant à la séquence des transitions à tirer pour arriver à ce marquage.

Pour obtenir un tel modèle « auto-contrôlé », nous nous y prenons en deux temps :

1. Dans le modèle du Chapitre 3, les ensembles de reconnaissances sont représentés par un unique jeton contenant la liste des reconnaissances. Dans un premier temps, nous éclatons cette liste pour obtenir un jeton par reconnaissance, ce qui permet d'introduire des premiers mécanismes de contrôle avec des gardes sur certaines transitions. Nous appelons donc ce second modèle « modèle multi-jetons »
2. Nous ajoutons ensuite, dans un second temps, une structure de gestion du flux d'évènements qui permet d'achever l'auto-contrôle des réseaux. Nous appelons donc ce dernier modèle « modèle contrôlé ».

Dans la Section 4.1, nous commençons donc par construire formellement le modèle dit « multi-jetons » puis nous analysons le degré de contrôle acquis pas l'éclatement des jetons de listes de reconnaissances en plusieurs jetons. Ensuite, dans la Section 4.2, nous construisons formellement l'évolution suivante de notre modèle, à savoir le modèle « contrôlé » qui représente la réalisation des objectifs que nous nous sommes fixés (modularité, confluence, concurrence) et nous achevons ce chapitre en les illustrant avec l'analyse des graphes d'espace d'états.

4.1 Construction et fonctionnement des réseaux dits « multi-jetons »

Reprenons donc les réseaux de Petri présentés dans le Chapitre 3. Ceux-ci fonctionnent avec un unique jeton par place, les multiples reconnaissances étant stockées dans une liste. Dans l'objectif d'implémenter une première forme de contrôle du tirage des transitions, nous construisons, à partir de ces anciens réseaux, des réseaux « multi-jetons » où chaque jeton correspond à *une* reconnaissance. Cela revient donc à « éclater » le jeton de notre modèle précédent, qui contient une liste de reconnaissances, en autant de jetons représentant chacun une unique reconnaissance. Ceci permet de tirer meilleur parti des fonctionnalités des réseaux de Petri colorés en introduisant ensuite des gardes sur les transitions pour que celles-ci n'aient d'effet que sur certains jetons.

Dans cette section, nous reprenons la construction des réseaux présentés dans le Chapitre 3 en l'adaptant au changement de représentation des reconnaissances. Nous commençons par poser les nouveaux types et expressions nécessaires (Section 4.1.1). Nous décrivons la nouvelle structure globale de nos réseaux (Section 4.1.2), puis nous définissons les nouvelles briques de base (Section 4.1.3) nécessaires pour ensuite construire formellement notre nouveau modèle (Section 4.1.4). Nous évaluons ensuite le degré de contrôle atteint sur le tirage des transitions et donnons une stratégie de tirage adaptée (Section 4.1.5).

4.1.1 Types et expressions utilisés dans les réseaux multi-jetons

Posons maintenant les nouveaux types et fonctions que nous utilisons dans nos réseaux de Petri « multi-jetons ».

Nous reprenons les types de base définis en 3.1.1 mais en remplaçant le type `NCList` qui correspondait à une liste de reconnaissances par le type `IndChronInst` qui désigne une unique reconnaissance. Nous ajoutons également le type `MkdChronInst` qui est le type `IndChronInst` marqué d'informations supplémentaires nécessaires dans le cas de la conjonction.

On pose $\mathcal{B} = \{\texttt{INT}, \texttt{Event}, \texttt{IndChronInst}, \texttt{MkdChronInst}\}$, toujours avec $\texttt{NEvent} = \texttt{Event} \times \texttt{INT}$.

`IndChronInst` est une instance de chronique, il s'agit d'un couple (l, n) où l est une liste de `NEvent` et n est un entier que nous appelons *indice de la reconnaissance* et qui stocke une information de marquage permettant entre autres de ne faire subir qu'une seule opération à chaque jeton.

`MkdChronInst` est une instance de chronique couplée d'une liste de couples de listes d'entiers, que l'on appellera *marquage de l'instance*. Un entier permet d'identifier un unique évènement, donc une liste d'entiers identifie une reconnaissance. Le marquage de l'instance est donc une liste de couples de reconnaissances. Il permet de conserver en mémoire les couples de reconnaissances avec lesquels une certaine opération a déjà été effectuée. Ce marquage spécifique d'instances de chroniques est utilisé pour un besoin particulier de la conjonction où il est nécessaire de conserver un historique des opérations ayant été effectuées.

Un certain nombre de fonctions sont également utilisées :

$\mathcal{F} = \{+, \max, \text{checked}, \text{mark}, \text{markABS}, \text{complete}, \text{merge}, \text{mergeABS}, \text{chg\_win}, \text{test\_older}, \text{test\_olderMkd}, \text{compatible}, \text{compatibleABS}, \text{usable}, \text{equalt}\}$

Ces fonctions sont décrites dans le Tableau 4.1.

4.1.2 Structure globale des réseaux multi-jetons

Comme évoqué précédemment, nous souhaitons conserver le système de construction modulaire de nos réseaux. Pour pouvoir être combinés entre eux, ceux-ci ont donc toujours une structure commune, présentée Figure 4.2, qui reste très proche de celle de notre modèle précédent (Figure 3.4). Lors de l'étape de construction des réseaux, nous définissons donc toujours les fonctions $\texttt{Present}(C)$, $\texttt{Start}(C)$, $\texttt{Success}(C)$, $\texttt{WiniIn}(C)$, et $\texttt{WiniOut}(C)$.

Dans notre nouvelle structure, le type des places est modifié pour s'adapter à la nouvelle représentation des reconnaissances. Les places `Start` et `Success` sont de type `IndChronInst` et

Tableau 4.1 – Description des fonctions utilisées dans nos réseaux

Fonction	*Type (image de la fonction $\sigma : \mathcal{F} \rightarrow \mathcal{B}$)*	*Description*
+	`INT` × `INT` → `INT`	Addition usuelle.
max	`INT` × `INT` → `INT`	Donne l'entier maximum entre deux entiers.
checked	`IndChronInst` × `INT` → `IndChronInst`	Met à jour l'indice de la reconnaissance avec l'entier en argument.
mark	`MkdChronInst` × `IndChronInst` ×`IndChronInst` → `MkdChronInst`	Met à jour le marquage de l'instance marquée en y ajoutant le couple des indices des deux reconnaissances.
markABS	`MkdChronInst` × `IndChronInst` → `MkdChronInst`	Met à jour le marquage de l'instance marquée en y ajoutant les indices de la reconnaissance.
complete	`Event` × `INT` × `IndChronInst` → `IndChronInst`	Complète l'instance de reconnaissance partielle avec l'évènement indicé de l'entier.
merge	`MkdChronInst` × `IndChronInst` ×`IndChronInst` → `IndChronInst`	Récupère la première instance de reconnaissance et lui ajoute la seconde instance.
mergeABS	`MkdChronInst` × `IndChronInst` → `IndChronInst`	Récupère la première instance de reconnaissance et lui ajoute la combinaison des deux dernières instances.
chg_win	`INT` × `IndChronInst` → `INT`	Met à jour l'entier avec l'indice de la reconnaissance si celui-ci est plus grand.
test_older	`INT` × `IndChronInst` → *Bool*	Vérifie si l'entier est supérieur ou égal à l'indice de début de la reconnaissance.
test_olderMkd	`INT` × `MkdChronInst` → *Bool*	Vérifie si l'entier est supérieur ou égal à l'indice de début de la reconnaissance.
compatible	`MkdChronInst` × `IndChronInst` ×`IndChronInst` → *Bool*	Vérifie, sur le marquage de la reconnaissance marquée, si la combinaison avec les deux autres reconnaissances n'a pas encore été faite.
compatibleABS	`MkdChronInst`×`IndChronInst` → *Bool*	Vérifie, sur le marquage de la reconnaissance marquée, si la combinaison avec l'autre reconnaissance n'a pas encore été faite.
usable	`IndChronInst` × `INT` → *Bool*	Vérifie si l'indice de la reconnaissance est strictement inférieur à l'entier.
equalt	`IndChronInst` × `INT` → *Bool*	Vérifie si l'indice de la reconnaissance est égal à l'entier.

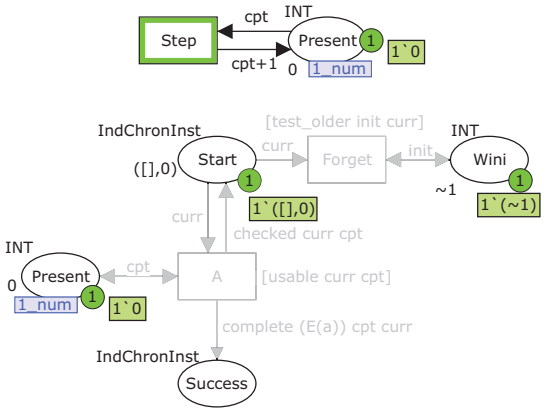


FIGURE 4.2 – Structure des réseaux multi-jetons

contiennent un jeton par reconnaissance.

Le compteur d'évènements est également légèrement modifié, comme on le verra dans la Section 4.1.3.

Le sens de lecture des réseaux est modifié : ceux-ci doivent maintenant être lus de haut en bas et non plus de gauche à droite.

Comme dans le modèle du Chapitre 3, les jetons des reconnaissances circulent dans le réseau pour être complétés au fur et à mesure. En revanche, comme il y a maintenant un jeton par reconnaissance, il faut tirer les transitions complétant les reconnaissances autant de fois qu'il y a de jetons à compléter.

Remarque 20 (marquage initial des places `Start`). Nous évoquions dans la Remarque 18 (p.84) les différents marquages initiaux des places `Start` contrôlant l'activation ou non des réseaux. Le marquage [[ ]] activait un réseau en offrant une liste vide [ ] à compléter alors que le marquage [ ] bloquait provisoirement le réseau.

Les places contiennent maintenant un jeton par reconnaissance donc les marquages initiaux possibles deviennent les suivants :

— ([ ], n), avec l'indice de reconnaissance $n \in \mathbb{N}$, ce qui correspond à une reconnaissance partielle vide en attente de complétion et qui active donc le réseau qui suit [1] ;

— le marquage vide (*i.e.* aucun jeton dans la place) qui indique qu'il n'y a aucune reconnaissance à compléter pour le moment et qui bloque donc le réseau qui suit.

4.1.3 Briques de base

Avant de pouvoir effectuer la nouvelle construction par induction, nous reprenons les briques de base définies dans la Section 3.2.3 pour les adapter à notre nouvelle représentation des recon-

1. Attention, dans le modèle précédent, [ ] désignait une liste vide de reconnaissances donc l'absence de reconnaissances à compléter. Dans le modèle présenté dans cette section, chaque jeton représente une reconnaissance, donc [ ] représente l'existence d'une reconnaissance vide dans l'attente d'être complétée.

naissances.

Compteur Ce sous-réseau noté `CPT` fait toujours office de compteur d'évènements. Il est composé d'une place `Present` dans laquelle est stockée la valeur du compteur, et d'une transition `Step` qui incrémente le compteur à chaque fois qu'elle est tirée. Il s'agit de tirer la transition `Step` avant de traiter un évènement et non après, contrairement au fonctionnement des anciens réseaux. Ceci permet de faciliter la manipulation des indices. À tout évènement est associée une valeur du compteur ce qui permet de distinguer, dans une reconnaissance, deux évènements de même nom étant survenus à des instants différents.

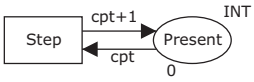


Figure 4.3 – Compteur

Opérateur `OR` Contrairement à la première modélisation et comme on le détaillera dans la Section 4.1.4, un opérateur de disjonction est nécessaire : il faut dupliquer à l'aide de la transition `startOR` les jetons de la place `Start` (qui correspondent aux reconnaissances partielles à être complétées par la disjonction) de façon à ce qu'ils puissent être utilisés pareillement par les réseaux correspondants à chacune des deux chroniques de la disjonction. On notera cet opérateur $\texttt{OP}_{\texttt{OR}}$.

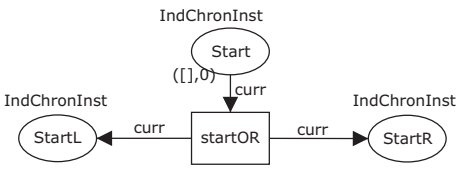


Figure 4.4 – Opérateur `OR`

La place `Start` de cette brique joue un rôle central en déclenchant la possibilité de reconnaître chacun des membres de la disjonction. Pour pouvoir s'y référer plus clairement par la suite, on la nomme `Start(OR)`.

Opérateur `AND` De même que dans notre précédente construction, ce sous-réseau noté $\texttt{OP}_{\texttt{AND}}$ sert à calculer la reconnaissance de la conjonction de deux chroniques C_1 et C_2. Il s'agit donc de réaliser toutes les combinaisons possibles d'une reconnaissance de C_1 avec une reconnaissance de C_2 et de les utiliser pour compléter une reconnaissance partielle éventuellement vide selon la nature de la chronique globale considérée. Réaliser ces combinaisons est un problème complexe, et le passage à un modèle multi-jeton ne permet plus d'intégrer cette complexité dans les fonctions en ML.

En effet, le problème est le suivant : on dispose de deux places contenant chacune respectivement les reconnaissances de C_1 et celles de C_2 (rappelons qu'à chaque reconnaissance correspond un unique jeton), ainsi que d'une place contenant les reconnaissances partielles à compléter par la

conjonction (ces reconnaissances partielles peuvent être vides selon la chronique globale considérée). Pour plusieurs raisons, il n'est pas correct de combiner aveuglément les éléments des deux places contenant les reconnaissances de C_1 et de C_2.

Une difficulté réside dans le fait que les combinaisons créées sont ensuite utilisées pour compléter les reconnaissances partielles (éventuellement vides). Il est donc nécessaire d'intégrer une structure assurant que toutes les possibilités de complétion sont réalisées (a priori chaque reconnaissance de C_1, par exemple, participe à plusieurs reconnaissances globales de la chronique considérée). Rappelons que, lorsqu'un jeton est utilisé par une transition, il est ôté de la place, à moins qu'un arc en sortie de la transition ne l'y remette. Si l'on consomme les jetons de reconnaissance de C_1 et de C_2 au fur et à mesure que l'on fait des combinaisons, il faut au préalable les avoir dupliqués un nombre exactement suffisant de fois pour pouvoir effectuer toutes les combinaisons possibles, et il faut ensuite s'assurer que toutes ces combinaisons sont effectivement réalisées (*i.e.* que les jetons dupliqués ne sont pas utilisés pour engendrer plusieurs fois une même combinaison). Ce problème est complexe entre autres car il n'est pas trivialement possible de savoir à combien de reconnaissances chaque jeton va participer, et donc combien de fois il faut le dupliquer.

Si, au contraire, on choisit de remettre dans leur place les jetons utilisés, ceci permet de s'affranchir du problème d'avoir suffisamment de jetons par rapport au nombre de combinaisons possibles à réaliser. La situation reste cependant complexe :

— il faut s'assurer de ne pas produire plusieurs fois une même reconnaissance avec les mêmes jetons ;
— il faut également veiller à ce que l'ensemble des combinaisons possibles soient intégralement parcouru ;
— si la conjonction est le second membre d'une séquence (et donc que les reconnaissances partielles à compléter ne sont pas vides), il faut combiner chaque couple de reconnaissances de C_1 et de C_2 avec les bonnes reconnaissances du premier membre de la séquence (celles qui les précèdent).

Pour remplir l'ensemble de ces conditions, il est donc nécessaire de conserver deux informations : quelles reconnaissances ont été complétées (dans le cadre d'une séquence), et quelles combinaisons de reconnaissances de C_1 et de C_2 ont été créées.

Notre réseau fonctionne donc comme suit. Les places `Operand1` et `Operand2` contiennent respectivement les reconnaissances de C_1 et de C_2, et la place `Start`, initialement marquée $([\,], 0)$ peut contenir des reconnaissances à compléter dans le cas d'une séquence. Ce sont les jetons de reconnaissance éventuellement vide de cette place `Start` qui portent les informations que l'on souhaite conserver : le contenu de la place `Start` est d'abord marqué d'une liste vide à l'aide de la transition `Mark0` (on passe donc d'un jeton de type `IndChronInst` à un jeton de type `MarkedChronInst`), cette liste vide est ensuite mise à jour pour contenir des couples de listes d'entiers : chaque liste d'entiers correspond à une reconnaissance (de C_1 ou de C_2 selon s'il s'agit du premier ou du second membre du couple), et donc chaque couple correspond à un couple de deux reconnaissances, l'une de C_1 et l'autre de C_2. On conserve ainsi non seulement la liste des combinaisons de C_1 et de C_2 ayant déjà été créées, mais aussi quelles reconnaissances partielles de la place `Start` elles ont complété. La garde `compatible` sur la transition `Combine` permet d'engendrer exactement les combinaisons recherchées, en vérifiant dans la liste les combinaisons ayant déjà été engendrées.

Ainsi, lorsque toutes les combinaisons possibles ont été réalisées, la transition `Combine` n'est plus tirable.

Les places `StartSub1`, et `StartSub2` sont fusionnées avec les places `Start` des réseaux correspondant à C_1 et C_2, et la transition `Activate` permet d'activer ces places lorsque la place `StartBis` contient au moins un jeton. En effet, dans le cas où la conjonction est la seconde partie d'une séquence, on ne souhaite pas que le réseau commence à reconnaitre la conjonction tant que le premier membre de la séquence n'a pas été reconnu, et ceci se traduit par l'arrivée d'un jeton dans la place `Start` puis dans la place `StartBis`.

Les places `Wini` (qui en fait ne sont qu'une place car elles sont fusionnées — ce qui n'est ici qu'une représentation graphique facilitant la lecture) ainsi que les transitions `Forget`, `Forget1` et `Forget2` servent dans le cas d'une combinaison avec une absence.

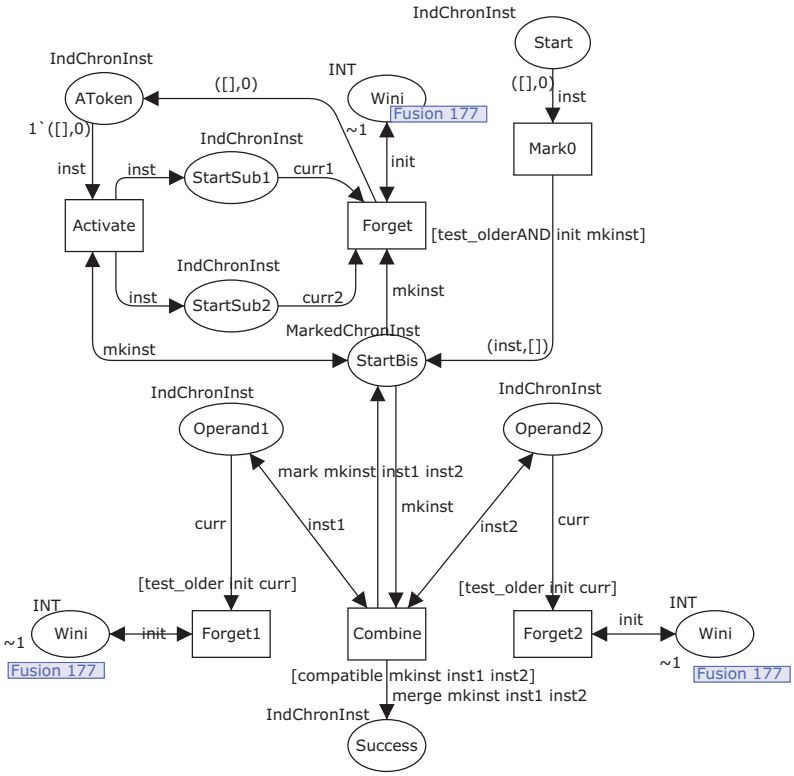


Figure 4.5 – Opérateur `AND`

Les places `Start` et `Success` de ce réseau jouent un rôle central car la première, comme pour la brique du `OR`, déclenche la possibilité de reconnaitre la chronique, et la seconde regroupe les reconnaissances de C_1&C_2. D'autre part, la place `Wini` ne pourra fonctionner que si elle est fusionnée avec les places correspondantes situées à l'extérieur du réseau et que son contenu est donc mis à jour. On aura donc besoin de se référer à ces places. Pour plus de clarté, on note `Success(AND)`

pour `Success`, `Wini`(`AND`) pour `Wini` et `Start`(`AND`) pour `Start`.

Opérateur `ABS` Comme dans notre précédente construction, ce sous-réseau noté $\texttt{OP}_{\texttt{ABS}}$ sert à composer deux réseaux de Petri correspondant aux chroniques C_1 et C_2 pour obtenir les reconnaissances de $(C_1) - [C_2[$ qui seront conservées dans la place `Success`(C_1).

Cet opérateur fonctionne autour du même principe que l'opérateur `ABS` du modèle à un seul jeton, à savoir qu'il faut gérer les bornes de l'absence :

— d'une part, dans le cas de plusieurs absences imbriquées, à travers la distinction entre les places `WiniBe` et `WiniAf` (*cf.* p.100) ;
— d'autre part, dans le cas de l'imbrication dans une séquence, il faut isoler les reconnaissances de l'absence pour les recombiner correctement (il s'agit donc de recombiner séquentiellement les reconnaissances éventuelles de la premiere partie de la séquence sui sont dans la place `Start` avec les reconnaissances de l'absence qui sont dans la place `Oper`).

L'introduction de plusieurs jetons n'influe pas sur le fonctionnement des places `WiniBe` et `WiniAf` mais complique la combinaison des reconnaissances des places `Start` et `Oper`. En effet, on retrouve la même problématique que dans l'opérateur de conjonction. Il faut effectuer toutes les combinaisons séquentielles possibles et, pour ce faire, nous ajoutons une structure qui marque les jetons de la place `Start` afin de garder la trace des combinaisons ayant été réalisées. Cette situation est analogue aux problèmes rencontrés autour de l'opérateur de conjonction.

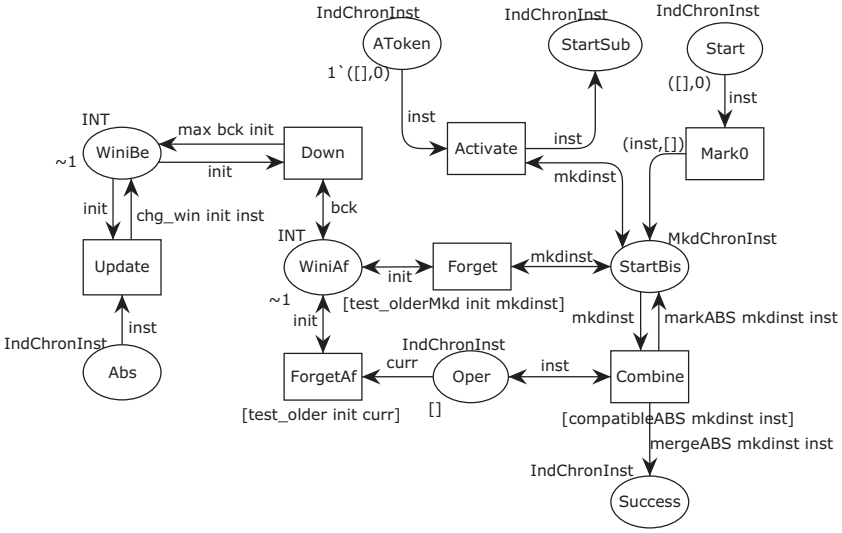


FIGURE 4.6 – Opérateur `ABS`

4.1.4 Construction par induction

Nous pouvons maintenant construire notre nouveau modèle. Définissons par induction sur la chronique C le nouveau réseau de Petri coloré $N(C)$ lui correspondant.

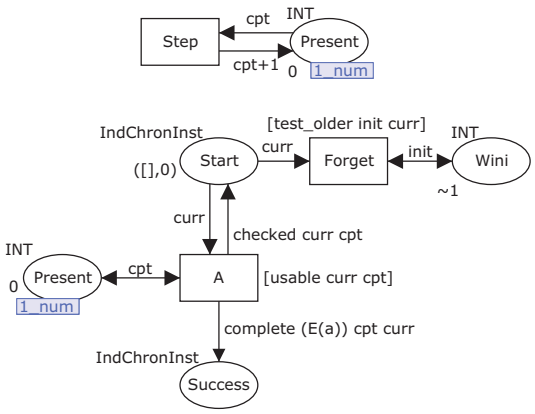


Figure 4.7 – Réseau correspondant à la chronique A

Si $C = A \in \mathfrak{N}$ Le fonctionnement global du réseau reconnaissant un évènement simple est sensiblement le même que celui de l'ancien modèle présenté dans la Section 3.2.4 (p.82). Le passage à un jeton par reconnaissance permet cependant de simplifier le réseau, qui est présenté Figure 4.7 :

- Une garde est apposée à la transition `Forget` qui permet de sélectionner les reconnaissances à supprimer dans le cas d'une absence. Cette suppression se fait alors simplement par le tirage de la transition qui les consomme. La transition `Forget` n'est donc tirable que lorsqu'elle doit supprimer des reconnaissances, ce qui constitue un début de « contrôle » des transitions.
- Une garde sur la transition `A` permet également d'instaurer un début de contrôle. L'indice marquant les reconnaissances [2] est utilisé ici pour garder une trace des jetons de reconnaissance ayant été complétés par l'occurrence de A en cours de traitement. L'indice n d'une reconnaissance l correspond à la valeur du compteur la dernière fois que l a été complétée par la transition `A`. Il suffit donc de comparer l'indice à la valeur courante du compteur : si le compteur a un indice supérieur, il faut tirer la transition. Un premier niveau de contrôle est implémenté : lors de l'occurrence d'un évènement A, la transition `A` est tirable exactement le bon nombre de fois, ce qui est intéressant car on ne peut donc pas la tirer une infinité de fois. En revanche, la gestion du flux n'est pas encore implémentée et la transition peut être tirée à l'occasion de l'occurrence d'un évènement autre que A.

2. On rappelle que les jetons de reconnaissance sont de type `IndChronInst`, à savoir des couples (l, n) où l est une reconnaissance et n est un entier.

Nous définissons :

$$\begin{aligned} \texttt{Present}(C) &= \texttt{Present} \\ \texttt{Start}(C) &= \texttt{Start} \\ \texttt{Success}(C) &= \texttt{Success} \\ \texttt{WiniOut}(C) &= \texttt{Wini} \\ \texttt{WiniIn}(C) &= \emptyset \end{aligned}$$

Puis nous fusionnons avec le compteur d'évènements comme dans le modèle précédent dans (3.1) p.84.

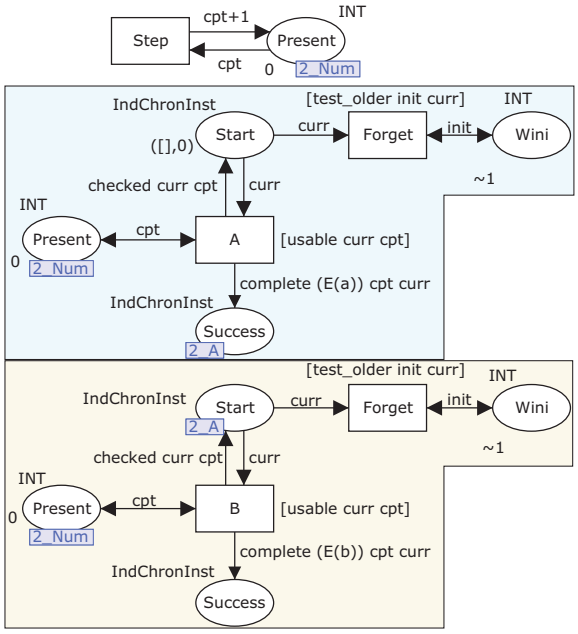


FIGURE 4.8 – Réseau correspondant à la chronique $A\ B$

Si $C = C_1\ C_2$ Afin de modéliser la séquence $C_1\ C_2$, nous fusionnons, comme dans le modèle précédent du Chapitre 3, la place `Success` du réseau $N(C_1)$ avec la place `Start` du réseau $N(C_2)$. La Figure 4.8 présente un exemple de séquence sur la chronique $A\ B$. Cette fusion fait fonctionner les deux réseaux en série et modifie le marquage initial de $\texttt{Start}(C_2)$. Comme évoqué dans la Remarque 20, pour désactiver le réseau relatif à C_2 tant qu'aucune reconnaissance de C_1 n'est formée, $\texttt{Start}(C_2)$ est vide.

Nous posons :

$$N'(C) = Fusion(\{N(C_1), N(C_2)\},$$
$$\{(\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2)\}),$$

$$(\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Start}(C_2)\})^{3},$$
$$(\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2), \texttt{WiniOut}(C_2)\})$$
$$\})$$

Nous définissons :

$$\begin{aligned}
\texttt{Present}(C) &= \texttt{Present}(C_1) \\
\texttt{Start}(C) &= \texttt{Start}(C_1) \\
\texttt{Success}(C) &= \texttt{Success}(C_2) \\
\texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1) \\
\texttt{WiniIn}(C) &= \texttt{WiniOut}(C_2)
\end{aligned}$$

Puis nous fusionnons avec le compteur d'évènements comme dans (3.1).

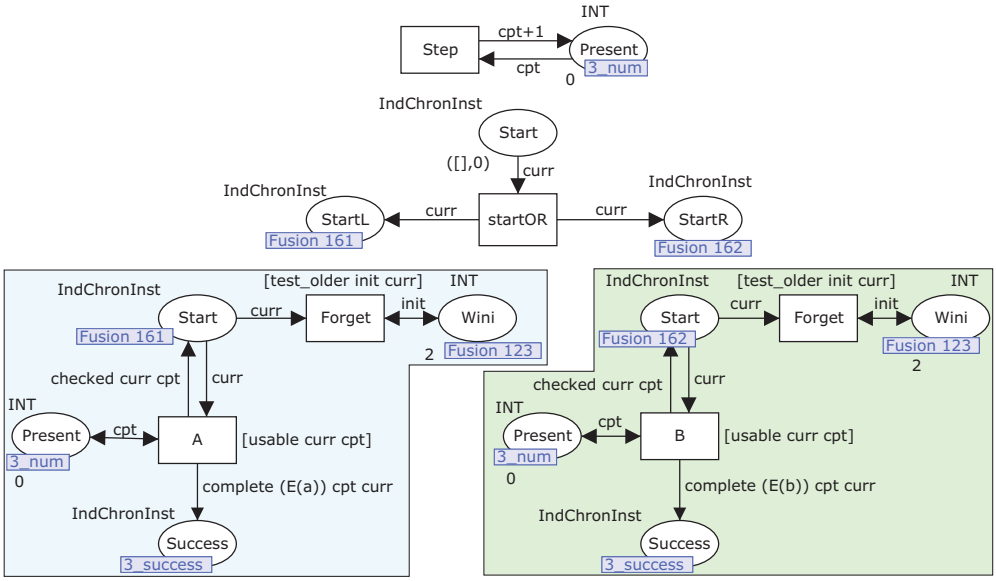


Figure 4.9 – Réseau correspondant à la chronique $A \,||\, B$

Si $C = C_1 \,||\, C_2$ Afin de modéliser la disjonction, comme dans le modèle précédent, les deux réseaux $N(C_1)$ et $N(C_2)$ fonctionnent en parallèle. Contrairement au réseau de disjonction de la première modélisation, le passage à un modèle multi-jeton nous empêche de simplement fusionner les places `Start` des réseaux $N(C_1)$ et $N(C_2)$. En effet, lorsque l'on tire une transition relative à un évènement celle-ci modifie le contenu de la place `Start` associée en changeant l'indice des reconnaissances y figurant, donc, si l'on tire une transition relative au membre de gauche de la

3. Le marquage initial de $\texttt{Start}(C_2)$ devient donc vide.

disjonction, les jetons de la place `Start` sont marqués et ne sont plus utilisables tant que l'on ne tire pas la transition `Step` du compteur d'évènements (c'est-à-dire tant qu'on ne passe pas à l'évènement suivant dans le flux). Ceci pose problème par exemple pour la chronique $A \parallel (A\ B)$ car l'on souhaite pouvoir tirer les transitions `A` des membres de gauche *et* de droite pour chaque évènement de nom A.

La transition `StartOr` duplique donc les jetons de reconnaissances de la place $\texttt{Start}(C_1 \parallel C_2)$ pour en mettre l'un dans $\texttt{Start}(C_1)$, l'autre dans $\texttt{Start}(C_2)$. Ces deux jetons identiques peuvent alors être complétés par chacune des deux parties du réseau.

Notons que, pour la chronique $A \parallel A$, on obtient donc deux reconnaissances de la chronique à chaque occurrence d'un évènement de nom A, ce qui correspond effectivement à la sémantique ensembliste.

Nous posons :

$$\begin{aligned} N'(C) = Fusion(\{&N(C_1), N(C_2), \texttt{OP}_{\texttt{OR}}\}, \\ &\{(\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2)\}), \\ &\ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Success}(C_2)\}), \\ &\ (\texttt{WiniOut}(C_1), \{\texttt{WiniOut}(C_1), \texttt{WiniOut}(C_2)\}), \\ &\ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2)\}), \\ &\ (\texttt{StartL}, \{\texttt{StartL}, \texttt{Start}(C_1)\}), \\ &\ (\texttt{StartR}, \{\texttt{StartR}, \texttt{Start}(C_2)\})\ ^4 \\ &\}) \end{aligned}$$

Nous définissons :

$$\begin{aligned} \texttt{Present}(C) &= \texttt{Present}(C_1) \\ \texttt{Start}(C) &= \texttt{Start}(\texttt{OR}) \\ \texttt{Success}(C) &= \texttt{Success}(C_1) \\ \texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1) \\ \texttt{WiniIn}(C) &= \begin{cases} \texttt{WiniIn}(C_1) & \text{si } \texttt{WiniIn}(C_1) \neq \emptyset \\ \texttt{WiniIn}(C_2) & \text{sinon} \end{cases} \end{aligned}$$

Puis nous fusionnons avec le compteur d'évènements comme dans (3.1).

Si $C = C_1 \& C_2$ Pour modéliser la conjonction $C_1 \& C_2$, les réseaux $N(C_1)$ et $N(C_2)$ doivent également fonctionner en parallèle. La brique $OP_{\texttt{AND}}$, dont le fonctionnement a été détaillé dans la Section 4.1.3, duplique les jetons de la place `Start` et active la possibilité de reconnaître les membres de la chronique, et ce en fusionnant les places `Start` de $N(C_1)$ et $N(C_2)$ avec les places `StartSub1` et `StartSub2`. Les reconnaissances de C_1 et C_2 sont récupérées par l'opérateur `AND` grâce à la fusion des places $\texttt{Success}(C_1)$ et $\texttt{Success}(C_2)$ avec les places `Operand1` et `Operand2`.

Nous posons : $N'(C) = Fusion(\{N(C_1), N(C_2), \texttt{OP}_{\texttt{AND}}\},$
$\{(\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2)\}),$
$(\texttt{Startsub1}, \{\texttt{Startsub1}, \texttt{Start}(C_1)\}),$

4. Les marquages initiaux des places $\texttt{Start}(C_1)$ et $\texttt{Start}(C_2)$ deviennent donc vides.

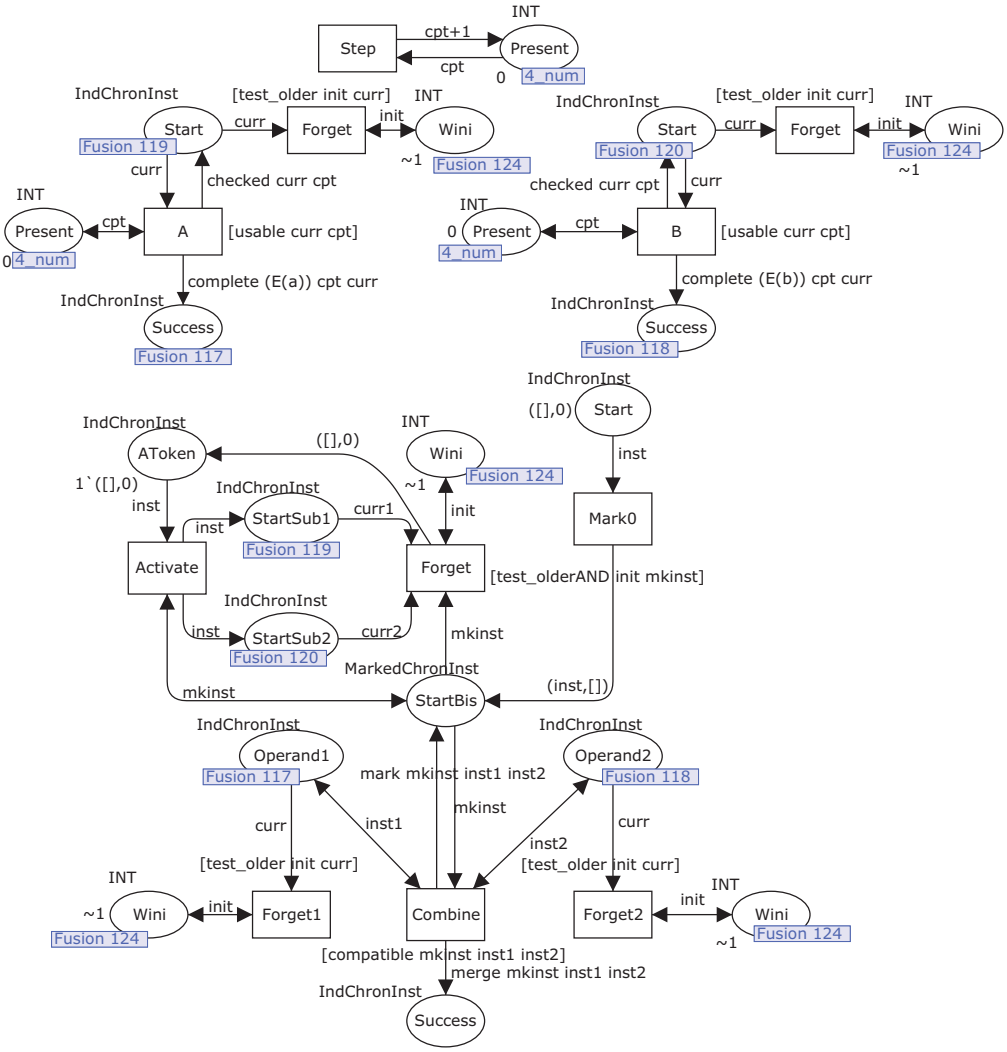


Figure 4.10 – Réseau correspondant à la chronique $A\&B$

$(\texttt{Startsub2}, \{\texttt{Startsub2}, \texttt{Start}(C_2)\}),$

$(\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Operand1}\}),$

$(\texttt{Success}(C_2), \{\texttt{Success}(C_2), \texttt{Operand2}\}),$

$(\texttt{WiniOut}(C_1), \{\texttt{WiniOut}(C_1), \texttt{WiniOut}(C_2), \texttt{Wini}(\texttt{AND})\}),$

$(\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2)\}) \quad \})$

Nous définissons :

$$\begin{aligned}
\texttt{Present}(C) &= \texttt{Present}(C_1) \\
\texttt{Start}(C) &= \texttt{Start}(\texttt{AND}) \\
\texttt{Success}(C) &= \texttt{Success}(\texttt{AND}) \\
\texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1) \\
\texttt{WiniIn}(C) &= \begin{cases} \texttt{WiniIn}(C_1) & \text{si } \texttt{WiniIn}(C_1) \neq \emptyset \\ \texttt{WiniIn}(C_2) & \text{sinon} \end{cases}
\end{aligned}$$

Puis nous fusionnons avec le compteur d'évènements comme dans (3.1).

Si $C = (C_1) - [C_2[$ Afin de modéliser l'absence $(C_1) - [C_2[$, il faut détruire les reconnaissances partielles de C_1 qui sont invalidées par une reconnaissance de C_2. Pour ce faire, comme dans le modèle précédent, nous fusionnons la place $\texttt{Success}(C_2)$ avec la place `Abs` de l'opérateur $\texttt{OP}_{\texttt{ABS}}$ pour mettre à jour le contenu de la place `WiniBe` qui est fusionnée avec $\texttt{Wini}(C_1)$. Le marquage de cette dernière, permet, à l'aide des transitions `Forget`, de supprimer les reconnaissances invalidées de C_1 en les consommant.

D'autre part, les places $\texttt{Present}(C_1)$ et $\texttt{Present}(C_2)$ sont toujours fusionnées pour propager la valeur du compteur. Et l'on fusionne la place $\texttt{Wini}(C_2)$ avec la place `WiniAf` qui va devenir la place `Wini` du réseau. Ainsi, si une seconde absence vient s'imbriquer sur le réseau de $(C_1) - [C_2[$, les reconnaissances de la nouvelle chronique interdite viendront également nettoyer le réseau $N(C_2)$.

La place `Start` du réseau est la place $\texttt{Start}(\texttt{ABS})$ ce qui permet, dans le cas d'une imbrication de l'absence dans une séquence, d'isoler les reconnaissances de l'absence pour ensuite les combiner correctement avec les reconnaissances de la place `Start`.

Nous posons :

$$\begin{aligned}
N'(C) = Fusion(&\{N(C_1), N(C_2), \texttt{OP}_{\texttt{ABS}}\}, \\
&\{(\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2), \texttt{Present}(\texttt{ABS})\}), \\
&\ (\texttt{Start}(C_1), \{\texttt{Start}(C_1), \texttt{Start}(C_2), \texttt{StartSub}\}), \\
&\ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Oper}\}), \\
&\ (\texttt{Success}(C_2), \{\texttt{Success}(C_2), \texttt{Abs}\}), \\
&\ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniBe}\})\ \})
\end{aligned}$$

Nous définissons :

$$\begin{aligned}
\texttt{Present}(C) &= \texttt{Present}(C_1) \\
\texttt{Start}(C) &= \texttt{Start}(\texttt{ABS}) \\
\texttt{Success}(C) &= \texttt{Success}(\texttt{ABS}) \\
\texttt{WiniOut}(C) &= \texttt{WiniAf} \\
\texttt{WiniIn}(C) &= \emptyset
\end{aligned}$$

Puis nous fusionnons avec le compteur d'évènements comme dans (3.1), ce qui achève la construction formelle de nos premiers réseaux multi-jetons.

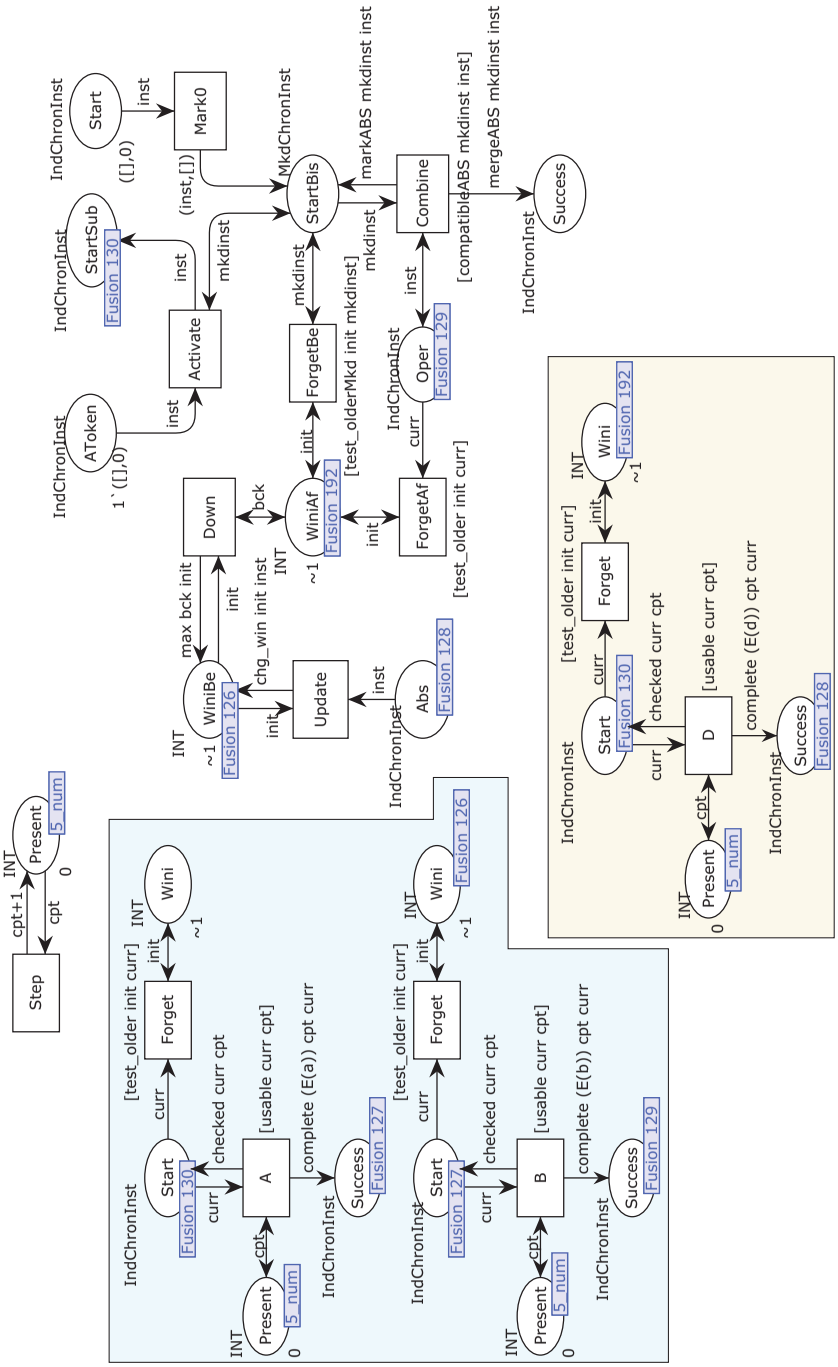


Figure 4.11 – Réseau correspondant à la chronique $(A\ B) - [D[$

4.1.5 Bilan sur le degré de contrôle acquis et stratégie de tirage

Le passage à un modèle multi-jeton où un jeton est associé à chaque reconnaissance partielle a permis d'implémenter un début de structure de contrôle sur le tirage des transitions. En effet, mis à part les transitions `Step` et `Down` qui sont en permanence tirables, les transitions ne sont tirables qu'un nombre limité de fois à la suite. Ainsi, à l'occurrence d'un évènement A, les transitions `A` sont tirables exactement le bon nombre fois, puis ne sont plus tirables jusqu'au tirage de la transition `Step` pour le traitement d'un nouvel évènement. Il en est de même pour les autres transitions.

En revanche, il n'y a pas encore d'implémentation de la gestion du flux dans les réseaux donc certaines transitions sont tirables, certes un nombre limité de fois, mais alors qu'elles ne devraient pas être tirées pour la gestion de l'évènement en cours de traitement. Par exemple, à l'occurrence d'un évènement `B`, une transition `A` peut être tirée mais ne doit pas l'être. Les réseaux que nous avons construits ne sont donc pas encore indépendants pour atteindre les marquages correspondant au traitement correct de chaque évènement, marquages où l'on peut lire les ensembles de reconnaissances correspondants. Il faut donc de nouveau définir une stratégie de tirage.

En d'autres termes, les deux problèmes majeurs rendant la définition d'une stratégie de tirage nécessaire sont les suivants :

- deux transitions sont en permanence tirables :
  - la transition `Step`, dont le tirage au mauvais moment a un effet critique sur le réseau puisqu'il agit sur le compteur d'évènements qui est directement lié à la gestion du flux d'évènements,
  - la transition `Down`, dont le tirage intempestif n'a aucun effet sur le réseau, mais qui pourrait boucler indéfiniment ;
- les transitions liées aux évènements (`A`, `B`, ...) sont tirables dès que la place `Start` qui leur est associée est activée, et cela sans considération de l'évènement qui est en cours de traitement, ceci est donc directement lié à la gestion du flux.

Définition formelle d'une stratégie de tirage

Il nous faut donc de nouveau définir formellement une stratégie de tirage qui, associée aux réseaux précédents, permet de lire dans leur marquage, après le traitement d'un évènement, les reconnaissances correctes de la chronique concernée.

Pour ce faire, nous devons tout d'abord définir, comme pour le modèle précédent, une fonction auxiliaire $Forget_C$ qui correspond à la liste des transitions `Forget` de $N(C)$ convenablement ordonnée qui sont tirées lors d'une absence.

Nous serons amenés à plusieurs reprises à vouloir tirer une transition jusqu'à ce que celle-ci ne soit plus tirable. Nous notons donc $\texttt{Trans}^*$ le tirage de la transition `Trans` jusqu'à ce qu'elle ne soit plus tirable.

Définition 35 (stratégie de tirage des transitions `Forget` pour les réseaux multi-jetons). On définit par induction sur la chronique C la fonction auxiliaire $Forget_C$:

- Si $C = A \in \mathfrak{N}$, alors $Forget_C = [\texttt{Forget}^*]$
- Si $C = C_1\, C_2$, alors $Forget_C = Forget_{C_2} :: Forget_{C_1}$

— Si $C = C_1 \,||\, C_2$, alors $Forget_C = Forget_{C_2} :: Forget_{C_1}$
— Si $C = C_1 \& C_2$, alors $Forget_C = [\texttt{Forget}^*, \texttt{Forget1}^*, \texttt{Forget2}^*] :: Forget_{C_2} :: Forget_{C_1}$
— Si $C = (C_1) - [C_2[$, alors $Forget_C = Forget_{C_2} :: [\texttt{Down}] :: Forget_{C_1} :: [\texttt{ForgetBe}, \texttt{ForgetAf}]$

À partir de cette définition, nous pouvons maintenant définir la stratégie de tirage générale :

Définition 36 (stratégie de tirage pour les réseaux multi-jetons). On définit par induction sur la chronique C la *stratégie de tirage* S_C où, pour tout évènement $e \in \mathfrak{N}$, $S_C(e)$ correspond à la suite des transitions à tirer dans le réseau $N(C)$ pour le traitement de l'évènement e.

Pour ce faire, nous définissons une fonction auxiliaire S'_C qui donne toutes les transitions à tirer, exceptée la transition `Step`. En effet, comme dans le modèle précédent, celle-ci doit être tirée une seule fois par évènement, au début, et est donc ajoutée a posteriori.

Pour tout $e \in \mathfrak{N}$:
— Si $C = A \in \mathfrak{N}$, alors $S'_C(e) = \begin{cases} [\texttt{A}^*] & \text{si } e = A \\ [\,] & \text{sinon} \end{cases}$
— Si $C = C_1\, C_2$, alors $S'_C(e) = S'_{C_1}(e) :: S'_{C_2}(e)$
— Si $C = C_1 \,||\, C_2$, alors $S'_C(e) = [\texttt{Start}(\texttt{OR})^*] :: S'_{C_1}(e) :: S'_{C_2}(e)$
— Si $C = C_1 \& C_2$, alors $S'_C(e) = [\texttt{Mark0}^*, \texttt{Activate}^*] :: S'_{C_1}(e) :: S'_{C_2}(e) :: [\texttt{Combine}^*]$
— Si $C = (C_1) - [C_2[$, alors $S'_C(e) = [\texttt{Mark0}^*, \texttt{Activate}^*] :: S'_{C_1}(e) :: S'_{C_2}(e) :: [\texttt{Update}, \texttt{Down}] :: Forget_{(C_1)-[C_2[} :: [\texttt{Combine}^*]$

On pose maintenant, pour tout $e \in \mathfrak{N}$, $S_C(e) = [\texttt{Step}] :: S'_C(e)$.

Ceci achève la construction du modèle dit « multi-jetons » qui intègre un début de structure de contrôle du tirage des transitions grâce à l'éclatement des jetons de listes de reconnaissances.

4.2 Construction et fonctionnement des réseaux « contrôlés »

Rappelons que l'objectif de ce chapitre est de construire un modèle de reconnaissance qui soit à la fois :
— modulaire ;
— concurrent ;
— autonome dans le sens où tout tirage de transitions correspondant au traitement d'un évènement du flux doit mener au marquage dans lequel les reconnaissances correctes associées peuvent être lues.

Dans la Section 4.1, une première étape de contrôle a été implémentée grâce au passage à une représentation multi-jeton des ensembles de reconnaissances, avec un jeton par reconnaissance. Cependant, les réseaux obtenus ne sont pas encore autonomes et il reste à implémenter une structure de gestion du flux d'évènements.

Dans cette section, nous achevons donc d'implémenter l'autonomie de nos réseaux en introduisant une telle structure [CCKP13a]. Le problème est double, il s'agit de :

1. gérer les transitions relatives aux évènements (`A`, `B`...) pour qu'elles ne soient tirables que lorsque l'évènement du flux en cours de traitement possède le bon nom ;

2. gérer la transition `Step` qui est pour le moment en permanence tirable : c'est un problème non trivial car il faut la tirer une fois que toutes les autres transitions du réseau ont été tirées, donc une fois qu'aucune autre transition du réseau n'est plus tirable.

Pour répondre à ces deux problèmes, nous procédons de la façon suivante, en implémentant la structure de gestion du flux :

1. Nous ajoutons aux réseaux une place contenant les événements du flux. La valeur du compteur permet d'identifier l'évènement qui est en cours de traitement et il est alors possible de mettre une garde sur les transitions relatives aux évènements pour qu'elles ne soient tirables que lorsqu'elles sont concernées.
2. Nous introduisons des jetons d'activation de transitions. Ces jetons parcourent les réseaux, en activant successivement diverses transitions dans un certain ordre, pour ensuite se retrouver autour de la transition `Step`. Lorsque tous les jetons de contrôle sont de nouveau parvenus à la transition `Step`, c'est que l'ensemble des autres transitions du réseau ont été correctement tirées et l'on peut donc tirer `Step` qui réinjecte les jetons de contrôle dans le réseau tout en incrémentant le compteur.

Dans cette section, nous allons faire évoluer notre modèle multi-jetons pour répondre aux objectifs que nous nous sommes fixés en début de chapitre, et construire un modèle dit « contrôlé ». Nous présentons en 4.2.1 les types et expressions utilisés puis en 4.2.2 la structure globale des réseaux contrôlés. Dans la Section 4.2.3, nous définissons les briques de bases jouant le rôle des opérateurs pour les réseaux contrôlés et, dans la Section 4.2.4 nous introduisons une structure en réseaux de Petri colorés que nous appelons « séparateur de jetons » et qui est utilisée pour la gestion des jetons de contrôle. La Section 4.2.5 est consacrée à la construction par induction des réseaux contrôlés. L'aspect « contrôlé » des réseaux nous permet ensuite d'utiliser les outils d'analyse de CPN Tools et de développer dans la Section 4.2.6 les graphes d'espace d'états des réseaux qui mettent en avant les caractéristiques recherchées des réseaux, à savoir la modularité, la concurrence, et l'autonomie.

4.2.1 Types et expressions utilisés

Dans nos réseaux « contrôlés » nous utilisons la même représentation des reconnaissances que dans la Section 4.1, à savoir que chaque reconnaissance partielle est représentée par un jeton. Nous conservons donc sensiblement les même types que ceux définis dans la Section 4.1.1.

Nous ajoutons :

— le type `JETON` qui est utilisé pour la structure d'activation des transitions ;
— le type `MkdINT` qui correspond à un entier marqué d'un autre entier et qui est utilisé pour le marquage des places `Wini` ;
— le type `IndChronInstList` qui permet de stocker des listes de reconnaissances.

On pose $\mathcal{B} = \{\texttt{JETON}, \texttt{INT}, \texttt{MkdINT}, \texttt{Event}, \texttt{IndChronInst}, \texttt{IndChronInstList}\}$, toujours avec $\texttt{NEvent} = \texttt{Event} \times \texttt{INT}$,

et $\mathcal{F} = \{+$, maxi, addChecked, complete, mergeAll, chgWin, test_old, test_oldAND, remobs, rightTime, decr2, unfold, Guard, negGuard $\}$. Ces fonctions sont décrites dans le Tableau 4.12.

Tableau 4.12 – Description des fonctions utilisées dans nos réseaux

Fonction	*Type (image de la fonction $\sigma : \mathcal{F} \rightarrow \mathcal{B}$)*	*Description*
+	`INT` × `INT` → `INT`	Addition usuelle.
maxi	`MkdINT` × `MkdINT` → `MkdINT`	Donne l'entier maximum entre les deux premières composantes des deux couples.
addChecked	`IndChronInstList` × `IndChronInst` × `INT` → `IndChronInstList`	Met à jour l'indice de la reconnaissance avec l'entier et ajoute la reconnaissance à la liste.
complete	`Event` × `INT` × `IndChronInst` → `IndChronInst`	Complète l'instance de reconnaissance partielle avec l'évènement indicé de l'entier.
mergeAll	`IndChronInstList` × `IndChronInstList` ×`IndChronInstList` × `INT` → `IndChronInst`	Réalise toutes les combinaisons possibles de reconnaissances des trois listes, telles que la reconnaissances de la troisième liste s'enchaine séquentiellement avec les reconnaissances des deux premières listes. La fonction renvoie autant de jetons que de reconnaissances créées.
chgWin	`MkdINT` × `IndChronInst` → `INT`	Met à jour l'entier marqué avec l'indice de la reconnaissance si celui-ci est plus grand.
test_old	`MkdINT` × `IndChronInst` → *Bool*	Vérifie si l'entier marqué est supérieur ou égal à l'indice de début de la reconnaissance.
test_oldAND	`MkdINT` × `IndChronInstList` → *Bool*	Vérifie si l'entier marqué est supérieur ou égal à l'indice de début de l'une des reconnaissances.
remobs	`MkdINT` × `IndChronInstList` → *IndChronInstList*	Supprime de la liste des reconnaissances les reconnaissances dont l'indice de début est strictement inférieur à l'entier marqué.
rightTime	`IndChronInst` × `INT` → *Bool*	Vérifie si l'indice de la reconnaissance est égal à l'entier en argument.
decr2	`MKdINT` × `INT` → *MkdINT*	Modifie le marquage de l'entier marqué en le marquant de l'entier en argument décrémenté de 1.
unfold	`IndChronInstList` → *IndChronInst*	Éclate la liste de reconnaissances en autant de jetons que de reconnaissances.
Guard	`IndChronInst` × `INT` × `Event` × `NEvent` → *Bool*	Vérifie que l'évènement indicé porte le nom recherché et que son indice est égal à l'entier en argument, et vérifie également que l'indice de la reconnaissance est strictement inférieur à l'entier en argument.
negGuard	`IndChronInst` × `INT` × `Event` × `NEvent` → *Bool*	Vérifie que soit l'évènement indicé porte le nom recherché et son indice est égal à l'entier en argument et également l'indice de la reconnaissance est supérieur ou égal à l'entier en argument, soit l'évènement ne porte par le bon nom mais son indice est égal à l'entier en argument.

4.2.2 Structure globale des réseaux

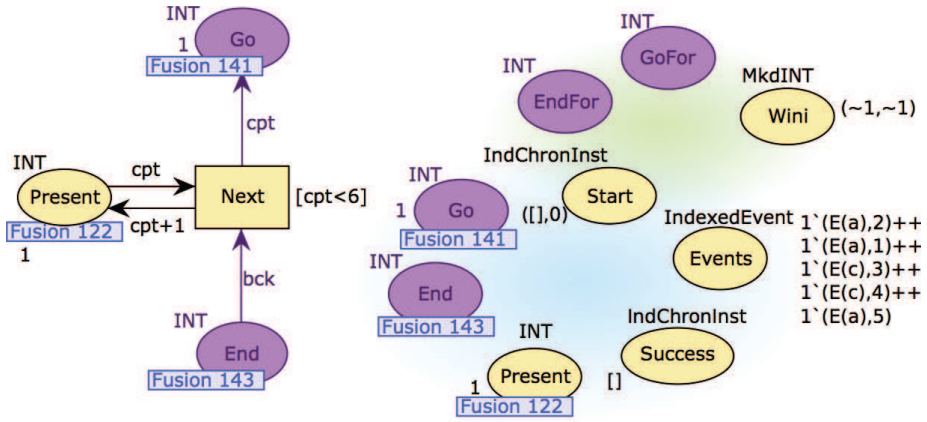


FIGURE 4.13 – Structure globale des réseaux contrôlés

Nous conservons toujours une structure modulaire pour pouvoir définir nos réseaux par induction. La structure globale des réseaux contrôlés que nous allons construire dans cette section est présentée dans la Figure 4.13. Elle est fondée sur celle de la Section 4.1.2 à laquelle nous avons ajouté les places suivantes :

— la place `Events` est de type `IndexedEvent`, elle contient le flux d'évènements à traiter, avec un jeton par évènement ;
— les places `Go` et `End` contiennent des jetons de contrôle qui sont de type `INT`, ils sont en charge de l'activation et de la désactivation des transitions associées au traitement des évènements ;
— les places `GoFor` et `EndFor`, également de type `INT`, ont le même rôle que `Go` et `End` mais pour la transition `Forget`.

Notons que, dans nos réseaux, la place `Events` contient directement l'ensemble des évènements du flux à traiter, mais qu'il est tout à fait envisageable de relier la place `Events` à une structure extérieure fournissant au fur et à mesure les évènements à traiter. Il est ainsi possible de connecter nos réseaux de reconnaissance de chronique contrôlés à des simulations réalisées également en réseaux de Petri colorés, par exemple.

Le compteur d'évènements est également modifié pour prendre en compte les jetons de contrôle, ce qui est détaillé dans la Section 4.2.3.

Comme dans le précédent modèle, des jetons représentant des reconnaissances partielles évoluent dans le réseau, depuis la place `Start` jusqu'à la place `Success`, étant peu à peu complétés par des évènements du flux. Les transitions sont contrôlées par les jetons de contrôle qui parcourent le réseau intégralement au traitement de chaque évènement du flux. Le jeton de contrôle parcourt la partie du réseau liée à la reconnaissance en profondeur (places `Go` à `End`), puis remonte en sens inverse en passant par les zones liées à l'absence (places `GoFor` à `EndFor`).

4.2.3 Briques de base

Notre construction par induction est toujours fondée sur un ensemble de briques de base que nous allons présenter ici. Ces briques sont celles de la Section 4.1.3 que nous avons modifiées pour intégrer les jetons de contrôle.

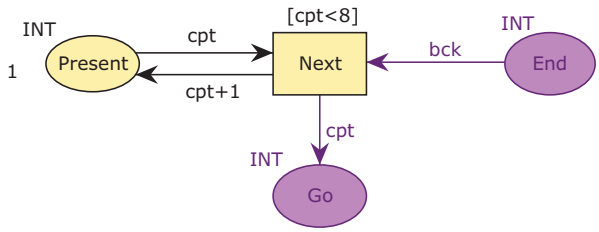


Figure 4.14 – Compteur

Compteur Notre modèle possède toujours un compteur d'évènements qui est présenté dans la Figure 4.14 et noté `CPT`. La transition `Step` s'est transformée en transition `Next`. Son tirage incrémente non seulement la valeur du compteur d'évènements qui est stockée dans la place `Present` pour passer à l'évènement suivant du flux, mais il transfère également le jeton de contrôle de la place `End` à la place `Go`.

Ainsi, la transition `Next` n'est tirable que lorsqu'il y a un jeton de contrôle dans la place `End`. Comme évoqué précédemment, ce jeton de contrôle parcourt l'ensemble du réseau, en étant parfois dupliqué puis ensuite unifié, et en activant ainsi successivement les différentes transitions à tirer dans le réseau. Lorsque toutes les transitions ont été correctement tirées, le jeton se trouve à nouveau dans la place `End`, ce qui permet de tirer `Next` et de passer au traitement de l'évènement suivant par le biais de la place `Go`.

Une garde [`cpt`$< d$], avec $d \in \mathbb{N}$ est apposée à la transition `Next`. Le réseau doit traiter tous les évènements du flux d'indice d'occurrence strictement inférieur à d. L'entier d de la garde correspond donc à l'entier d figurant dans l'ensemble de reconnaissances $R_C(\varphi, d)$ de la Définition 16. Du point de vue de nos réseaux, une fois la valeur de cet entier atteint par le compteur d'évènements, la transition `Next` n'est plus tirable et plus aucun autre évènement ne peut donc être traité. La garde contribue donc à assurer que toute séquence de transitions tirable est finie. Si l'on souhaite faire évoluer cette valeur dans le temps, on peut imaginer l'ajout d'une place de type `INT` en entrée de la transition `Next` et fournissant la valeur de d.

Opérateur `OR` La brique correspondant à l'opérateur de disjonction est présentée Figure 4.15 et notée $\texttt{OP}_{\texttt{OR}}$. Elle se compose maintenant de deux parties. Comme nous souhaitons maintenir un maximum de concurrence au sein de nos réseaux, et comme les traitements des branches de droite et de gauche d'une conjonction sont indépendants, le jeton de contrôle est dupliqué par l'opérateur de disjonction. Chaque jeton peut alors parcourir indépendamment sa branche de la disjonction. Lorsque les deux jetons ont achevé leurs parcours, cela signifie que l'ensemble des transitions liées

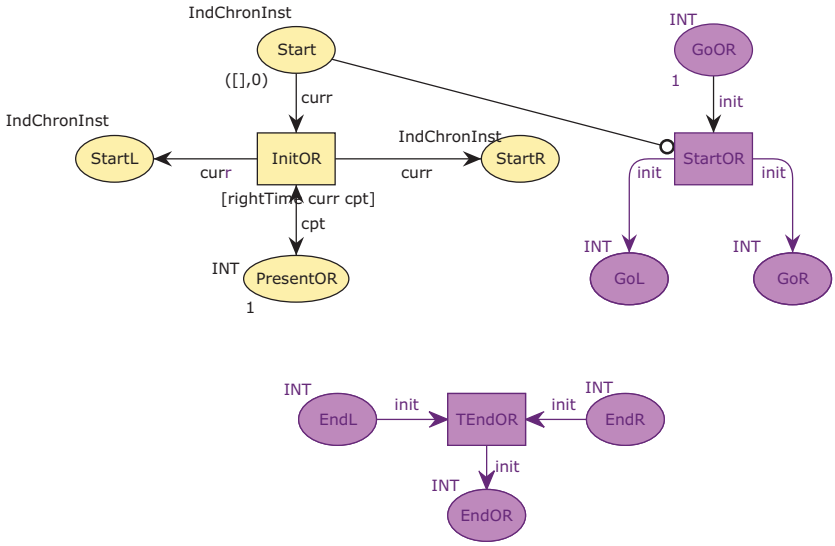


FIGURE 4.15 – Opérateur `OR`

à la disjonction ont été correctement tirées. On peut alors réunir les deux jetons de contrôle en un seul et procéder au traitement de la suite du réseau, selon la chronique étudiée.

Ce comportement est dirigé par les transitions `StartOR` et `TEndOR`. Pour activer le réseau de la disjonction, il faut que :

— toutes les reconnaissances partielles de la place `Start` aient été dupliquées pour traitement dans les places `StartL` et `StartR`, ce qui se matérialise par l'arc inhibiteur reliant la place `Start` à la transition `StartOR` ;

— un jeton de contrôle soit parvenu à la place `GoOR`.

Alors, le tirage de la transition `StartOR` duplique le jeton de contrôle dans les places `GoL` et `GoR` qui vont activer chacune des deux branches de la disjonction.

Lorsque les transitions d'une des branches ont toutes été correctement tirées, le jeton de contrôle parvient à la place `EndL` ou `EndR` selon la branche qu'il a explorée. Une fois que les deux jetons sont réunis dans les places `EndL` ou `EndR`, cela signifie que toute la disjonction a été correctement traitée, et la transition `TEndOR` est activée. Son tirage permet de procéder au traitement de la suite du réseau.

La transition `InitOR` fonctionne comme dans le modèle précédent (*cf.* 4.1.3), à savoir qu'elle duplique les reconnaissances partielles en attente de complétion dans `Start` pour qu'elles puissent être complétées par chacune des deux branches de la disjonction, à partir des places `StartL` et `StartR`. Une garde est ajoutée à la transition `InitOR` pour assurer que les reconnaissances dupliquées le soient au bon moment.

Opérateur `AND` L'opérateur de conjonction $\texttt{OP}_{\texttt{AND}}$ qui est représenté dans la Figure 4.16 fonctionne sur le même principe que celui du modèle multi-jetons (Figure 4.4, p.122) : les reconnais-

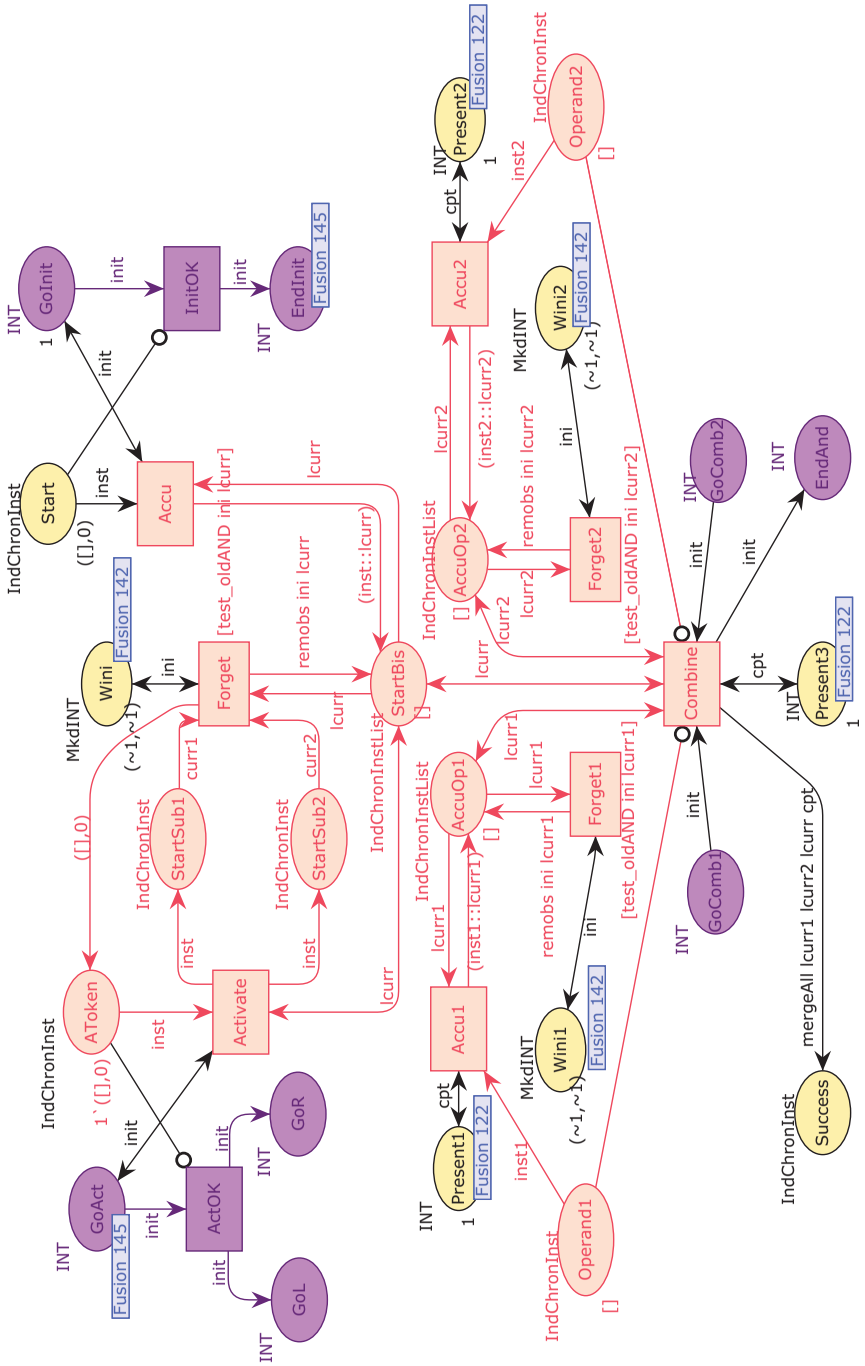


Figure 4.16 – Opérateur AND

sances des deux membres de la conjonction (places `Operand1` et `Operand2`) et de l'éventuel premier membre de la séquence (place `Start`), dans le cas d'une imbrication dans une séquence, sont réalisées séparément puis combinées (transition `Combine`). Comme dans les modèles précédents, on se référera à `Start(AND)` et `Success(AND)` pour les places `Start` et `Success`.

Il s'agit ici de contrôler, à l'aide des jetons de contrôle, le tirage des transitions. L'opérateur doit effectuer quatre tâches principales qui doivent être réalisées successivement dans cet ordre :

(i) pour répondre au problème soulevé dans la section précédente (p.122) pour la construction du modèle multi-jetons, les reconnaissances de la place `Start` doivent être accumulées dans la place `StartBis` ;

(ii) les places `Start` des deux membres de la conjonction doivent être activées avec les places `StartSub1` et `StartSub2` ;

(iii) les réseaux relatifs aux deux membres de la conjonction doivent être parcourus ;

(iv) les reconnaissances de `Start`, et des deux membres de la conjonction (places `Operand1` et `Operand2`) doivent être combinées.

Pour ce faire, le jeton de contrôle parcourt l'opérateur de conjonction comme suit :

(i) Le jeton de contrôle intègre le réseau par la place `GoInit`. Lorsque toutes les reconnaissances de la place `Start` ont été accumulées dans `StartBis` (et donc lorsque `Start` est vide), la transition `InitOK` peut être tirée (ce qui est assuré par un arc inhibiteur). Ceci indique que la première tâche est accomplie.

(ii) Le jeton de contrôle est alors transféré à la place `GoAct`. Alors :
— si la place `AToken` est vide (*i.e.* si les places `Start` des réseaux relatifs aux deux membres de la conjonction sont déjà activées), la transition `ActOK`, indiquant que l'activation est bien effectuée, est tirable ;
— sinon, il faut tirer la transition `Activate` pour activer les réseaux relatifs aux deux membres de la conjonction et vider la place `AToken`, rendant ainsi `ActOK` tirable.
Ceci assure que l'on ne procède pas à la suite des tâches (*i.e.* qu'on ne tire pas `ActOK`) tant que les réseaux des deux membres de la conjonction n'ont pas été activés.

(iii) Au tirage de `ActOK`, le jeton de contrôle est dédoublé pour aller parcourir les deux réseaux relatifs aux membres de la conjonction.

(iv) Il reste alors à combiner les reconnaissances des places `Start`, `Operand1` et `Operand2`. Comme dans le modèle précédent, les reconnaissances de `Start` ont été accumulées dans `StartBis` sous la forme d'une liste de reconnaissances marquées, ce qui permet de garder la trace des combinaisons ayant déjà été effectuées. Cependant, ceci ne suffit pas dans cette situation où nous avons besoin de savoir quand toutes les combinaisons ont été effectuées, c'est-à-dire quand la transition `Combine` n'est plus tirable. Pour répondre à ce problème, on accumule également sous forme de listes les reconnaissances de `Operand1` et `Operand2` dans `AccuOp1` et `AccuOp2` à l'aide de `Accu1` et `Accu2`. Ainsi, la combinaison des reconnaissances peut se faire avec un unique tirage de `Combine` [5], ce qui permet de transférer en même temps le jeton de

5. Notons que l'on retrouve ici le mécanisme des réseaux à un seul jeton.

contrôle dans la place `EndAnd`, indiquant que toutes les tâches ont été effectuées. La transition `Combine` n'est tirable que lorsque :

— les places `GoComb1` et `GoComb2` contiennent chacune un jeton de contrôle, *i.e.* les deux réseaux relatifs aux membres de la conjonction ont été parcourus ;

— les places `Operand1` et `Operand2` sont vides, *i.e.* l'intégralité de leur contenu a bien été accumulé dans les places `AccuOp1` et `AccuOp2`.

On obtient ainsi un opérateur fonctionnant sur le même principe que l'opérateur du modèle multi-jetons mais dans lequel le contrôle du tirage des transitions est implémenté.

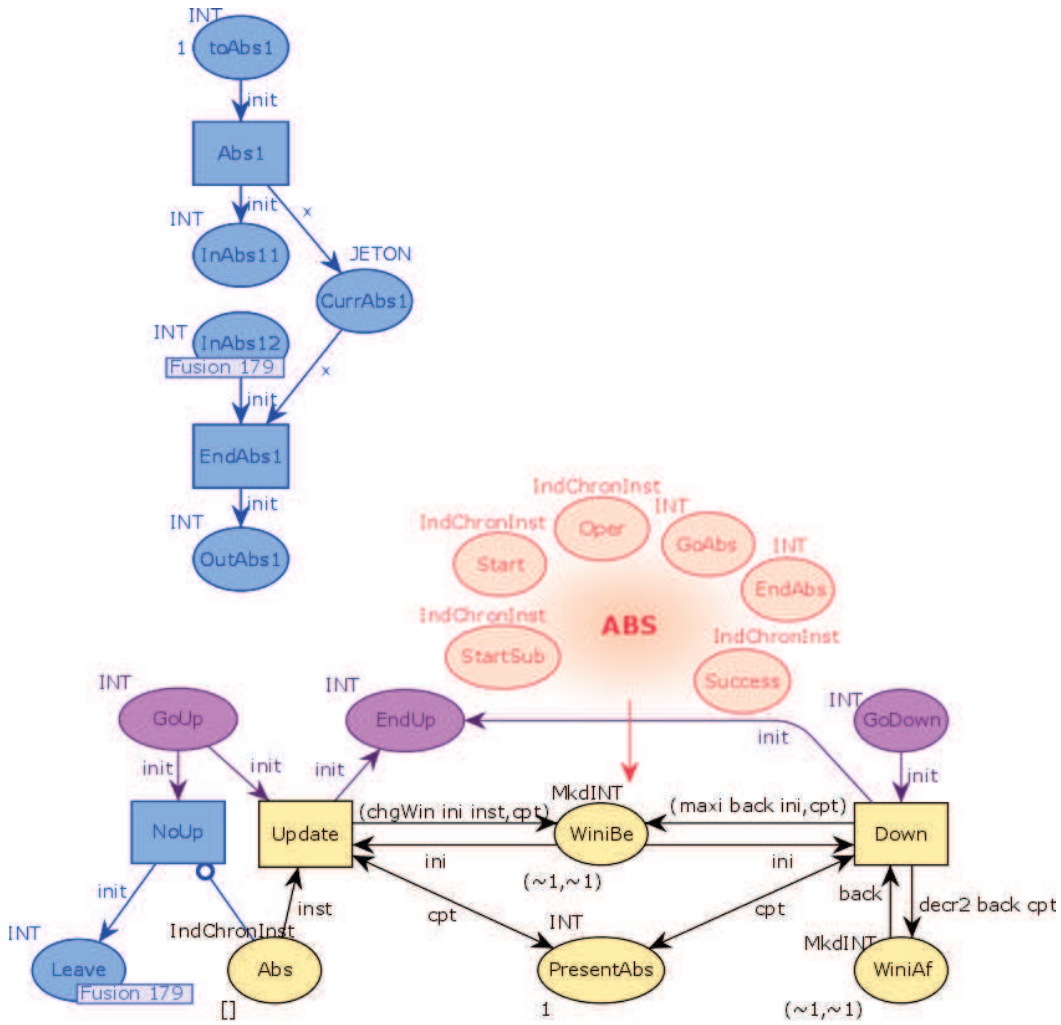


Figure 4.17 – Opérateur `ABS` de l'absence

Opérateur `ABS` De même que pour $\texttt{OP}_{\texttt{AND}}$, l'opérateur `ABS` noté $\texttt{OP}_{\texttt{ABS}}$ et représenté dans la Figure 4.17 fonctionne sur le même principe que l'opérateur `ABS` du modèle multi-jetons (Figure 4.6, p.125) mais implémente en prime le contrôle du tirage des transitions. Comme pour les modèles précédents, on se référera à `Start`(`ABS`) et `Success`(`ABS`) pour `Start` et `Success`.

La problématique est quadruple, il faut :

(i) mettre à jour le contenu de la place `WiniBe` qui correspond à la valeur de `Wini` liée à l'absence en cours (comme détaillé dans les modèles précédents) ;

(ii) combiner les reconnaissances du premier membre de l'absence avec les reconnaissances du premier membre d'une séquence dans le cas d'une imbrication dans une séquence ;

(iii) dans le cas d'une imbrication dans une autre absence, mettre à jour le contenu de la places `WiniAf` qui correspond à la valeur de `Wini` liée à un absence à un niveau supérieur ;

(iv) gérer, selon les bornes de l'absence, le tirage des transitions `Forget` du réseau autant de fois que nécessaire.

Dans le cas d'une simple absence, le jeton de contrôle effectue le parcours suivant :

— il intègre le réseau dans la place `toAbs1`, ce qui engendre le tirage de la transition `Abs1` et alors :
  — la place `CurrAbs1` contient un jeton qui indique que c'est cette absence qui est en cours de traitement (cette place est utile dans le cas de plusieurs absences imbriquées, cas que l'on étudiera par la suite),
  — le jeton de contrôle est transféré dans la place `InAbs11` ;

— le jeton de contrôle parcourt alors successivement les réseaux relatifs aux deux membres de l'absence ;

— à la suite de ce parcours, le jeton se retrouve dans la place `GoUp`, et deux cas se présentent alors :

1. s'il y a eu une nouvelle reconnaissance de la chronique interdite, alors la transition `Update` est tirable ce qui permet la mise à jour de la place `WiniBe`, et, par la place `EndUp`, le jeton de contrôle va remonter dans le premier membre de l'absence, par les places `EndFor` et `GoFor`, activant les transitions `Forget` correspondantes, puis le jeton atterri dans la place `InAbs12`,
2. sinon, le jeton de contrôle est directement transféré par `NoUp` dans la place `InAbs12`,

Notons que ces deux cas sont exclusifs : la transition `Update` est tirable si la place `Abs` contient une reconnaissance, alors que la transition `NoUp` n'est tirable que si la place `Abs` est vide ;

— la transition `EndAbs1` est ensuite tirable, ce qui achève le traitement de l'absence en activant la zone rouge `ABS` qui réalise les combinaisons des reconnaissances du premier membre de l'absence avec les reconnaissances du premier membre d'une séquence dans le cas d'une imbrication dans une séquence [6].

6. Le mécanisme de cette zone n'est pas détaillé ici pour rendre le réseau plus aisément compréhensible. Il répond cependant à la même problématique que celle rencontrée dans le cas des combinaisons à effectuer pour une conjonction, et a donc la même structure.

Explicitons maintenant le cas de deux absences imbriquées. On considère une absence numérotée 1 imbriquée dans une absence numérotée 2 pour distinguer les places des deux opérateurs $\texttt{OP}_{\texttt{UP}}$. Alors la place `GoDown1` (*i.e.* la place `GoDown` de l'absence « interne ») est fusionnée avec la place `EndFor2` (*i.e.* la place `EndFor` de l'absence externe). Le traitement de la première absence n'est pas influencé par la présence de la seconde absence. Le traitement de l'absence 2 s'effectue comme suit :

— le jeton de contrôle suit le parcours lié à une absence quelconque indiqué ci-dessus ;
— comme `EndFor2` est fusionné avec `GoDown1`, au lieu d'achever le traitement de l'absence, le jeton de contrôle, en remontant dans le réseau à travers les places `EndFor` et `GoFor` de l'absence externe, va activer la transition `Down1` de l'absence interne, qui met à jour la place `WiniBe` avec la valeur de `WiniAf` ;
— la transition `Down` transfère en même temps que cette mise à jour le jeton de contrôle dans la place `EndUp`, ce qui permet de parcourir les places `GoFor` et `EndFor` de l'absence interne, et ainsi propager le tirage des transitions `Forget` à l'absence interne ;
— le jeton de contrôle arrive alors dans la place `InAbs12` qui est également fusionnée avec la place `InAbs22`, mais comme la seconde absence est en cours de traitement il y a un jeton de marquage dans la place `CurrAbs2` mais pas dans la place `CurrAbs1`, donc seule la transition `EndAbs2` est tirable ce qui achève le traitement de l'absence en activant la zone rouge `ABS2`.

4.2.4 Un séparateur de jetons générique

Avant de pouvoir finalement construire notre modèle contrôlé de reconnaissance de chroniques, nous allons définir une dernière structure qui fait partie du réseau chargé de la reconnaissance d'un évènement simple.

La problématique est centrée autour du contrôle de la transition relative à l'évènement simple à reconnaître, par exemple `A`. Il s'agit de trouver un moyen de détecter qu'une transition `A` n'est plus tirable pour faire évoluer le jeton de contrôle dans la suite du réseau. Or, la transition `A` a effet sur une place `Start` pouvant contenir plusieurs reconnaissances partielles en attente de complétion. Certaines de ces reconnaissances doivent être traitées par la transition `A` et d'autres, non (typiquement parce qu'elles viennent d'être produites et qu'il faut donc attendre l'exécution suivante). Il faut élaborer un moyen de savoir quand toutes les instances devant être traitées l'ont bien été, et ce afin de déclencher la suite du parcours du jeton de contrôle. Lorsque toutes les instances ont été traitées, la transition `A` n'est plus tirable. Le comportement recherché ressemble donc à celui d'un arc inhibiteur. Cependant, la transition n'est plus tirable non pas parce que l'une des places en entrée est dénuée de tout jeton, mais parce qu'il n'y a plus aucun jeton qui vérifie la garde apposée à la transition. Il s'agit donc de construire une structure qui remplit cette fonction, avec la contrainte supplémentaire qu'elle ne doit, à terme, pas avoir modifié le contenu de la place `Start` car les reconnaissances partielles en attente de complétion peuvent avoir à être utilisées ultérieurement lors du traitement d'un autre évènement. Nous appelons cette structure un *séparateur de jetons*. Elle est présentée dans la Figure 4.18 et repose notamment sur les arcs inhibiteurs.

En d'autres termes, la situation est la suivante. Nous disposons de deux places de contrôle `Go` et `End`, ainsi que d'une place `Start` contenant des jetons et reliée à une transition possédant une

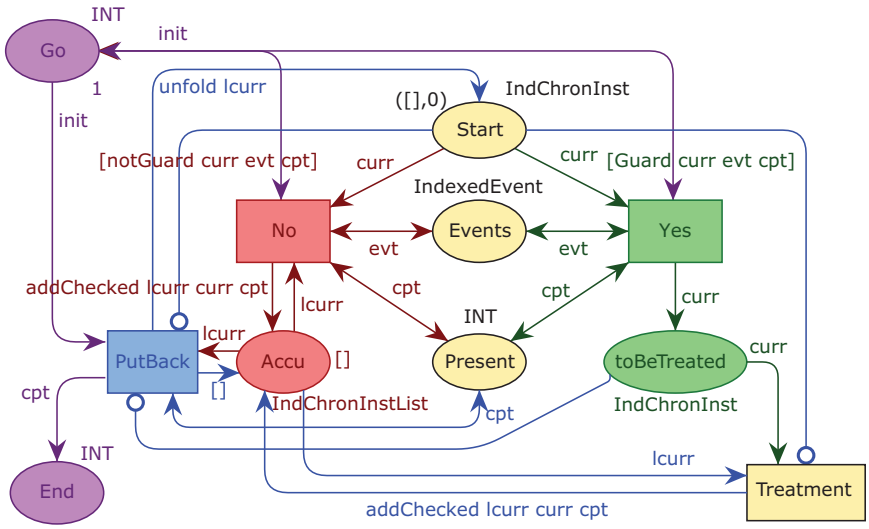


FIGURE 4.18 – Le réseau *séparateur de jetons*

garde. Nous souhaitons effectuer le contrôle suivant :

1. rien ne doit être effectué tant qu'aucun jeton de contrôle n'est arrivé dans la place `Go` ;
2. la transition doit être tirée pour chaque jeton de `Start` vérifiant la garde ;
3. une fois que ces tirages sont effectués, le jeton de contrôle doit être transféré de la place `Go` à la place `End` ;
4. lorsque le jeton de contrôle arrive à la place `End`, l'ensemble des jetons initialement présents dans la place `Start` doit toujours y être (mais ils peuvent avoir été manipulés entre temps).

Pour résoudre ce problème, nous introduisons donc le réseau *séparateur de jetons* présenté dans la Figure 4.18 et construit comme suit :

— la transition concernée est divisée en trois transitions : deux d'entre elles (`Yes` et `No`) représentent la garde, et la troisième (`Treatment`) correspond au traitement administré initialement par la transition divisée (il peut y avoir d'autres arcs en sortie de `Treatment`) ;
— la transition `Yes` conserve la garde initiale, alors qu'une garde complémentaire est apposée à la transition `No` – ceci permet de séparer les jetons de la place initiale `Start` en deux ensembles ;
— les deux transitions vident donc la place `Start`, et remplissent deux places : `Yes` transfère les jetons de `Start` vérifiant la garde dans la place `toBeTreated` pour qu'ils soient traités ultérieurement, alors que `No` accumule les jetons vérifiant la garde complémentaire dans une liste dans la place `Accu` ;
— un arc inhibiteur reliant la place `Start` à la transition `Treatment` empêche tout traitement tant que l'*intégralité* des jetons n'a pas été répartie entre les places `toBeTreated` et `Accu` ;

— lorsque cette séparation est terminée, le traitement peut donc être effectué, et les jetons traités sont ajoutés à la liste présente dans `Accu` ;
— le rôle de la transition `putBack` est de transférer le jeton de contrôle de la place `Go` à la place `End` tout en restaurant le contenu de la place `Start` : comme nous avons accumulé tous les jetons dans la liste de la place `Accu` (aussi bien ceux ayant transité par `No` que ceux qui sont passés par `Yes`), il suffit de déployer la liste ;
— deux arcs inhibiteurs reliant la transition `putBack` respectivement aux places `Start` et `toBeTreated` assurent que `putBack` est tirée en dernier.

Notons qu'il est important d'accumuler les jetons sous forme d'une *unique liste* dans la place `Accu`. En effet, ceci permet de restaurer le contenu de la place `Start` avec un seul tirage de la transition `PutBack`. Ceci est nécessaire pour savoir à quel moment le processus est terminé et donc quand il faut transférer le jeton de contrôle vers la place `End`.

4.2.5 Construction par induction des réseaux contrôlés

Avec les briques de base définies dans la Section 4.2.3 et la structure de séparation de jetons introduite dans la Section 4.2.4, nous pouvons maintenant procéder à la construction d'un modèle de reconnaissance de chronique que nous appelons « contrôlé ». Comme pour les modèles précédents, la construction se fait par induction autour d'une structure commune que nous avons détaillée dans la Section 4.2.2. Pour chaque chronique C nous construisons un réseau contrôlé $N(C)$ associé dans lequel circulent des jetons représentant des reconnaissances partielles qui sont progressivement complétées en transitant dans le réseau, selon le flux d'évènements étudiés.

Reconnaissance d'un évènement simple : $C = A$ avec $A \in \mathfrak{N}$

Commençons par construire le réseau charnière reconnaissant un évènement simple, comme par exemple A dont le réseau $N(A)$ est présenté dans la Figure 4.19, et dont la structure globale est représentée dans la Figure 4.20.

En haut à gauche, on reconnaît le compteur d'évènements défini Section 4.2.3. Comme dans les modèles précédents, on fusionne la brique reconnaissant A, notée $N'(A)$, au compteur d'évènements. Celui-ci est piloté par les places `Go` et `End` qui activent ou désactivent la transition `Next`. Dans le cas restreint de la reconnaissance d'un évènement simple, le jeton de contrôle de la place `Go` doit parcourir le réseau $N(A)$ pour activer diverses transitions appropriées puis achever son parcours dans la place `End` ce qui active de nouveau la transition `Next`.

Le reste du réseau, $N'(A)$, peut être divisé en deux parties qui sont identifiées dans la Figure 4.20 :

1. On reconnaît la structure de séparation de jetons présentée dans la section précédente dans la partie inférieure du réseau. En effet, $N'(A)$ est construit autour cette structure qui est appliquée une première fois à la transition relative à l'évènement simple recherché, ici `A`. Elle permet de contrôler son tirage, comme annoncé dans la Section 4.2.4. La place `toBeTreated` du séparateur de jeton est appelée dans $N'(A)$ `toComplete` car elle contient les reconnaissances à compléter par un évènement a. À cette structure, nous ajoutons la place `EventsA` qui

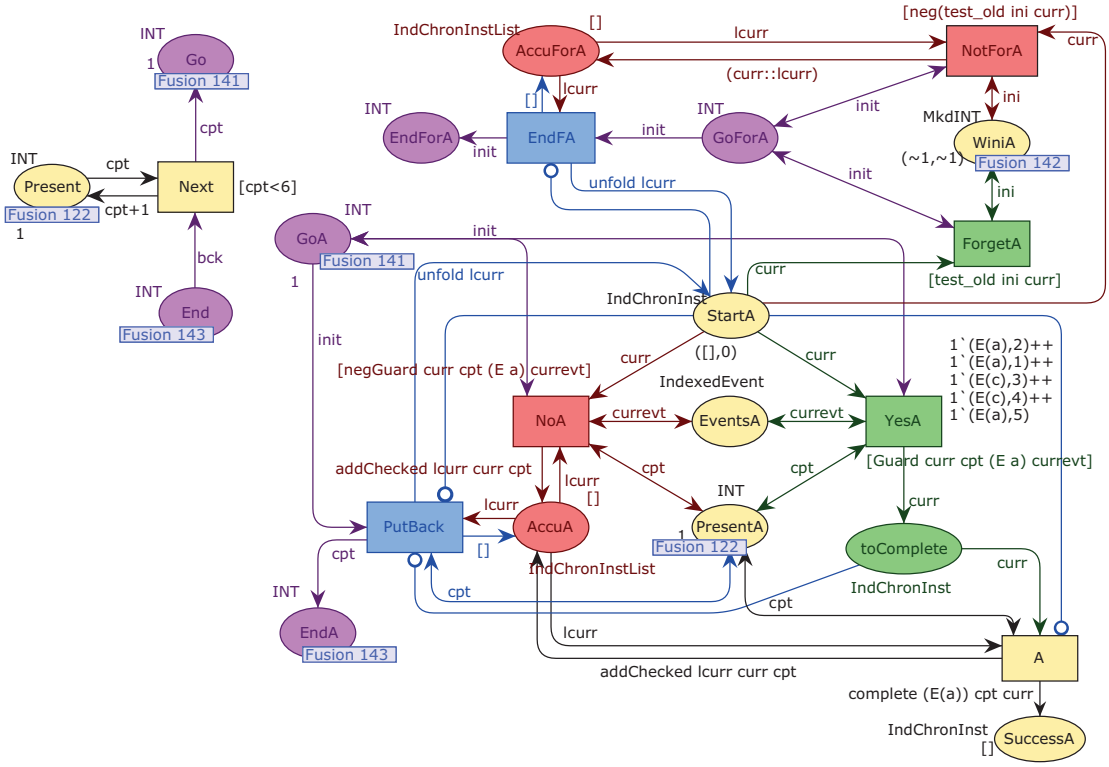


FIGURE 4.19 – Réseau reconnaissant la chronique A

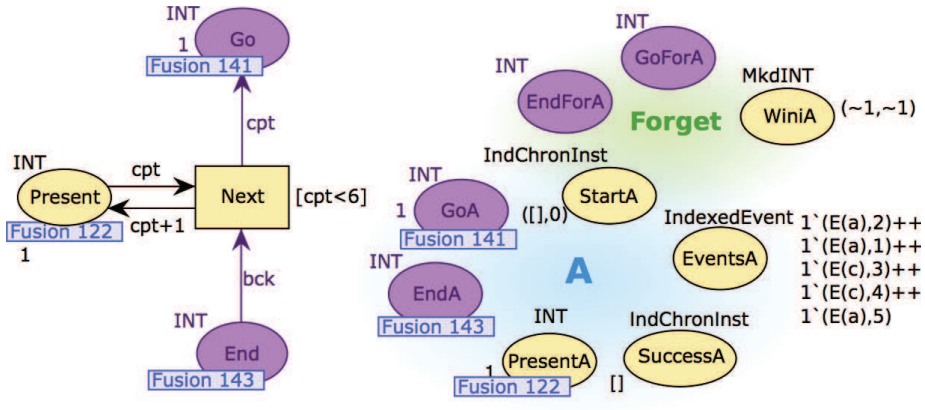


FIGURE 4.20 – Structure globale du réseau reconnaissant la chronique A

est reliée aux transitions `NoA` et `YesA` et qui contient le flux d'évènements à traiter. Dans le cas restreint de la reconnaissance d'un évènement simple, la place `StartA` contient la liste vide [ ] qui va être complétée par des évènements a du flux pour former des nouvelles reconnaissances de A qui sont regroupées dans la place `SuccessA`. Lors de l'occurrence d'évènements de noms différents de a, seule la transition `NoA` est activée, et le jeton parcourt la structure de séparation pour revenir dans la place `StartA` sans avoir été complété, donc sans qu'il se soit produit aucune modification du marquage du réseau.

2. La partie supérieure restante du réseau $N'(A)$ est une structure de séparation simplifiée appliquée à la transition `Forget` de nos anciens réseaux. Elle est chargée de supprimer certains jetons dans le cas d'une absence, et son fonctionnement sera détaillé par la suite lors de la construction du réseau reconnaissant une absence.

Nous définissons inductivement les places principales du réseau définissant sa structure globale :

$$\begin{aligned}
\texttt{Present}(C) &= \texttt{PresentA} \\
\texttt{Start}(C) &= \texttt{StartA} \\
\texttt{Success}(C) &= \texttt{SuccessA} \\
\texttt{WiniOut}(C) &= \texttt{WiniA} \\
\texttt{WiniIn}(C) &= \emptyset \\
\texttt{Events}(C) &= \texttt{EventsA} \\
\texttt{Go}(C) &= \texttt{GoA} \\
\texttt{End}(C) &= \texttt{EndA} \\
\texttt{GoFor}(C) &= \texttt{GoForA} \\
\texttt{EndFor}(C) &= \texttt{EndForA}
\end{aligned}$$

Formellement, le réseau est construit par la fusion du compteur d'évènements avec la brique de base reconnaissant l'évènement simple comme suit :

$$\begin{aligned}
N(C) = Fusion(\{N'(C), \texttt{CPT}\}, \{&(\texttt{Present}(\texttt{CPT}), \{\texttt{Present}(\texttt{CPT}), \texttt{Present}(C)\}), \\
&(\texttt{Go}(\texttt{CPT}), \{\texttt{Go}(\texttt{CPT}), \texttt{Go}(C)\}), \\
&(\texttt{End}(\texttt{CPT}), \{\texttt{End}(\texttt{CPT}), \texttt{End}(C)\})\})
\end{aligned} \tag{4.1}$$

Exemple 15. Lorsque le processus de reconnaissance du réseau $N(A)$ est mis en route sur le flux $\varphi = ((a,1),(a,2),(c,3),(c,4),(a,5))$, c'est-à-dire lorsque le marquage de la place `EventsA` est :

$$1`(E(a),1) + +1`(E(a),2) + +1`(E(c),3) + +1`(E(c),4) + +1`(E(a),5)$$

alors le marquage final de la place $\texttt{Success}(A)$ est le suivant :

$$1`([(E(a),1)],1) + +1`([(E(a),2)],2) + +1`([(E(a),5)],5)$$

ce qui indique trois reconnaissances de A.

Cela correspond bien à la sémantique ensembliste qui donne l'ensemble de reconnaissances suivant : $R_A(\varphi, 5) = \{(a, 1), (a, 2), (a, 5)\}$.

Reconnaissance d'une séquence : $C = C_1\ C_2$

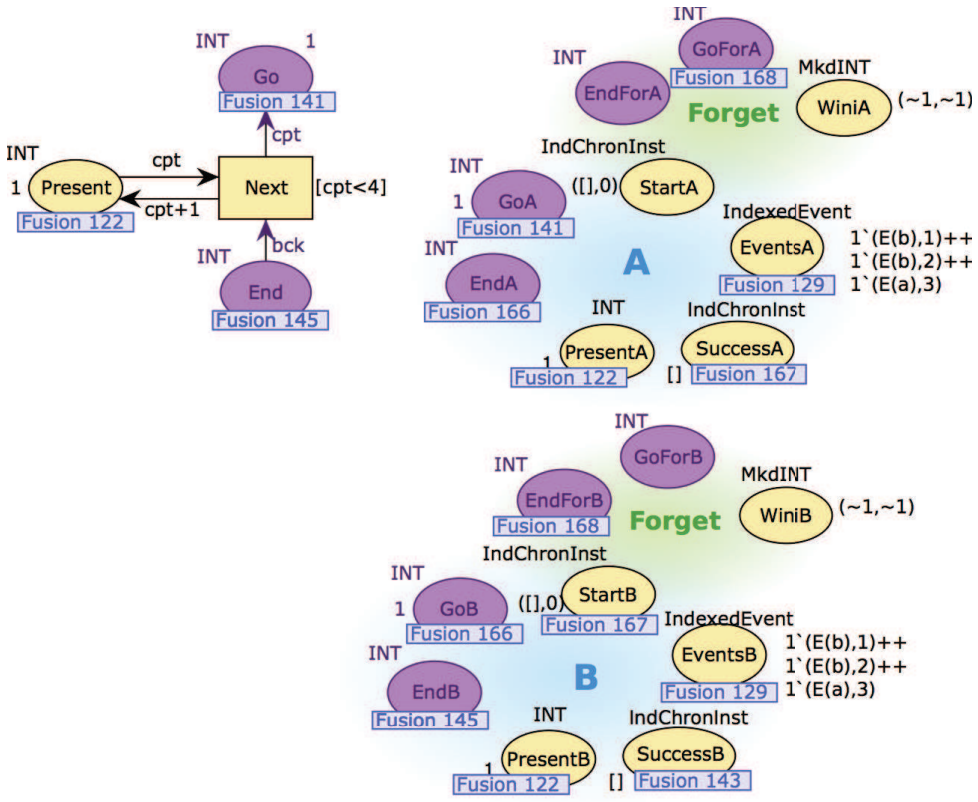


FIGURE 4.21 – Structure générale du réseau reconnaissant la chronique $A\ B$

La Figure 4.21 présente une exemple de réseau reconnaissant une séquence, sur la chronique $A\ B$.

Comme dans les modèles précédents, pour reconnaître une séquence, les réseaux fonctionnent en série donc nous fusionnons la place `Success`(C_1) avec `Start`(C_2). Comme précédemment, cette fusion modifie le marquage initial de la place `Start`(C_2) qui devient vide. Nous ne commençons donc pas à reconnaître C_2 tant que C_1 n'est pas reconnue.

Dans le cadre de l'implémentation du processus de contrôle du tirage des transitions, il faut aussi assurer la transmission du jeton de contrôle. Rappelons que le jeton de contrôle parcourt

la partie liée à la reconnaissance en profondeur puis remonte par les structures liées à l'absence. Comme les deux réseaux fonctionnent en série, le jeton de contrôle va parcourir le premier réseau puis le second : nous fusionnons donc simplement les places $\texttt{End}(C_1)$ et $\texttt{Go}(C_2)$ pour transmettre le jeton d'un réseau à l'autre. Il s'agit là du parcours en profondeur réalisant la reconnaissance. Le retour du jeton se fait en sens inverse par les structures liées à l'absence : il va parcourir d'abord la structure du réseau $N(C_2)$ puis celle de $N(C_1)$. Nous fusionnons donc les places $\texttt{EndFor}(C_2)$ et $\texttt{GoFor}(C_1)$.

Formellement, nous définissons le réseau comme suit :

$$\begin{aligned} N'(C_1\ C_2) = Fusion(\{N'(C_1), N'(C_2)\}, \{&(\texttt{End}(C_1), \{\texttt{End}(C_1), \texttt{Go}(C_2)\}),\\ &(\texttt{EndFor}(C_2), \{\texttt{EndFor}(C_2), \texttt{GoFor}(C_1)\}),\\ &(\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2)\}),\\ &(\texttt{Events}(C_1), \{\texttt{Events}(C_1), \texttt{Events}(C_2)\}),\\ &(\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Start}(C_2)\}),\\ &(\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2), \texttt{WiniOut}(C_2)\})\}) \end{aligned}$$

Nous définissons également inductivement les places principales du réseau :

$$\begin{aligned} \texttt{Present}(C) &= \texttt{Present}(C_1)\\ \texttt{Start}(C) &= \texttt{Start}(C_1)\\ \texttt{Success}(C) &= \texttt{Success}(C_2)\\ \texttt{WiniOut}(C) &= \texttt{WiniOut}(C_2)\\ \texttt{WiniIn}(C) &= \texttt{WiniIn}(C_2)\\ \texttt{Events}(C) &= \texttt{Events}(C_1)\\ \texttt{Go}(C) &= \texttt{Go}(C_1)\\ \texttt{End}(C) &= \texttt{End}(C_2)\\ \texttt{GoFor}(C) &= \texttt{GoFor}(C_2)\\ \texttt{EndFor}(C) &= \texttt{EndFor}(C_1) \end{aligned}$$

Puis nous fusionnons le réseau $N'(C_1\ C_2)$ avec le compteur d'évènements pour définir $N(C_1\ C_2)$ comme dans (4.1).

Reconnaissance d'une disjonction : $C = C_1 \mid\mid C_2$

La Figure 4.22 présente une exemple de réseau reconnaissant une disjonction, sur la chronique $A \mid\mid (B\ A)$.

Comme dans notre modèle précédent, et contrairement à la séquence, les réseaux d'une disjonction fonctionnent en parallèle à l'aide de l'opérateur de disjonction défini dans la Section 4.2.3. Ceci se traduit par la duplication, réalisée par l'opérateur de disjonction, des reconnaissances partielles en attente de complétion et du jeton de contrôle. Nous fusionnons donc `StartL` et `StartR` de l'opérateur avec respectivement $\texttt{Start}(C_1)$ et $\texttt{Start}(C_2)$ pour dupliquer les reconnaissances partielles en attente et les insérer dans chacune des branches de la disjonction. Nous fusionnons

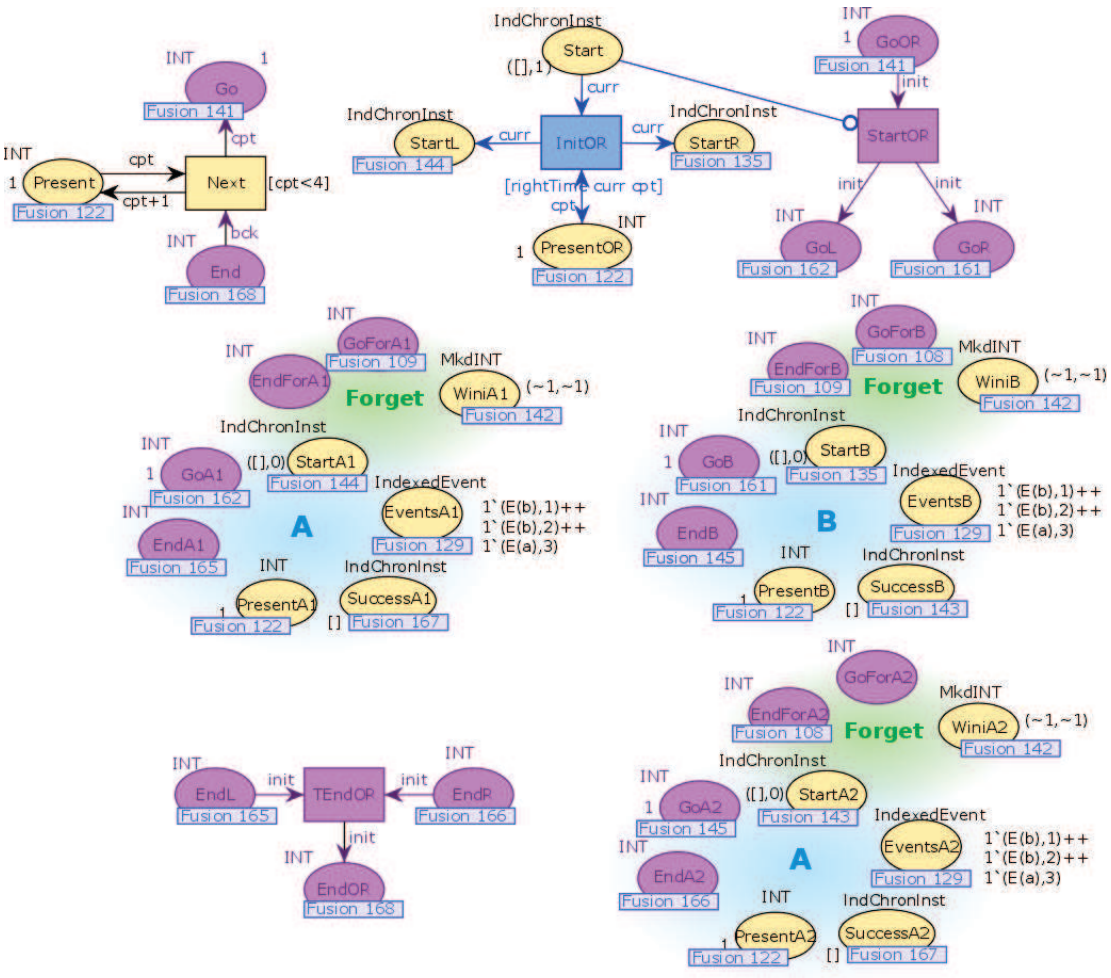


FIGURE 4.22 – Structure générale du réseau reconnaissant $A \,||\, (B\ A)$.

également GoL et GoR de l'opérateur avec respectivement $\texttt{Go}(C_1)$ et $\texttt{Go}(C_2)$, ce qui duplique le jeton de contrôle et permet d'activer les deux branches de la disjonction en parallèle. Lorsque chacun des deux jetons de contrôle a parcouru l'ensemble de sa branche, tout le réseau a correctement été parcouru. Nous fusionnons donc EndL et EndR de l'opérateur avec respectivement $\texttt{End}(C_1)$ et $\texttt{End}(C_2)$ pour réunir les deux jetons en un seul avec TEndOR. Les reconnaissances de la disjonction sont regroupées dans $\texttt{Success}(C_1)$ et $\texttt{Success}(C_2)$ qui sont fusionnées.

Formellement, nous définissons le réseau comme suit :

$$\begin{aligned}N'(C_1 \mid\mid C_2) = Fusion(&\{N'(C_1), N'(C_2), \texttt{OP}_{\texttt{OR}}\},\\ &\{(\texttt{GoL}, \{\texttt{GoL}, \texttt{Go}(C_1)\}), (\texttt{GoR}, \{\texttt{GoR}, \texttt{Go}(C_2)\}),\\ &\ (\texttt{EndFor}(C_2), \{\texttt{EndFor}(C_2), \texttt{GoFor}(C_1)\}),\\ &\ (\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2), \texttt{Present}(\texttt{OR})\}),\\ &\ (\texttt{StartL}, \{\texttt{StartL}, \texttt{Start}(C_1)\}), (\texttt{StartR}, \{\texttt{StartR}, \texttt{Start}(C_2)\}),\\ &\ (\texttt{Events}(C_1), \{\texttt{Events}(C_1), \texttt{Events}(C_2)\}),\\ &\ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Success}(C_2)\}),\\ &\ (\texttt{WiniOut}(C_1), \{\texttt{WiniOut}(C_1), \texttt{WiniOut}(C_2)\}),\\ &\ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2)\}),\\ &\ (\texttt{EndL}, \{\texttt{EndL}, \texttt{End}(C_1)\}), (\texttt{EndR}, \{\texttt{EndR}, \texttt{End}(C_2)\})\ \})\end{aligned}$$

Nous définissons également inductivement les places principales du réseau :

$$\begin{aligned}\texttt{Present}(C) &= \texttt{Present}(C_1)\\ \texttt{Start}(C) &= \texttt{Start}(\texttt{OR})\\ \texttt{Success}(C) &= \texttt{Success}(C_1)\\ \texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1)\\ \texttt{WiniIn}(C) &= \begin{cases}\texttt{WiniIn}(C_1) & \text{si } \texttt{WiniIn}(C_1) \neq \emptyset\\ \texttt{WiniIn}(C_2) & \text{sinon}\end{cases}\\ \texttt{Events}(C) &= \texttt{Events}(C_1)\\ \texttt{Go}(C) &= \texttt{Go}(\texttt{OR})\\ \texttt{End}(C) &= \texttt{End}(\texttt{OR})\\ \texttt{GoFor}(C) &= \texttt{GoFor}(C_2)\\ \texttt{EndFor}(C) &= \texttt{EndFor}(C_1)\end{aligned}$$

Puis nous fusionnons le réseau $N'(C_1 \mid\mid C_2)$ avec le compteur d'évènements pour définir le réseau $N(C_1 \mid\mid C_2)$ comme dans (4.1).

Exemple 16. Lorsque le processus de reconnaissance du réseau $N(A \mid\mid (B\ A))$ présenté Figure 4.22 est mis en route sur le flux $\varphi = ((b,1),(b,2),(a,3))$ alors le marquage final de la place $\texttt{Success}(A \mid\mid (B\ A))$ est le suivant :

$$1`([(E(a),3)],3) + +1`([(E(a),3),(E(b),1)],3) + +1`([(E(a),3),(E(b),2)],3)$$

ce qui indique trois reconnaissances de $A \mid\mid (B\ A)$, une provenant de la branche de gauche et les

deux autres, de la branche de droite de la disjonction.

Cela correspond bien à la sémantique ensembliste qui donne l'ensemble de reconnaissances suivant : $R_{A||(B\ A)}(\varphi, 3) = \{\langle\perp, \langle(b,1),(a,3)\rangle\rangle, \langle\perp, \langle(b,2),(a,3)\rangle\rangle, \langle(a,3),\perp\rangle\}$.

Reconnaissance d'une conjonction : $C = C_1 \& C_2$

Le réseau reconnaissant une conjonction $C_1\&C_2$ est construit à partir de la brique $\texttt{OP}_{\texttt{AND}}$, des deux sous-réseaux relatifs à C_1 et C_2, et d'un compteur d'évènements. À titre d'exemple, la Figure 4.23 présente le réseau reconnaissant la chronique $A\&B$.

La place `Start` du réseau est la place `Start(AND)`, dont le contenu est combiné par la transition `Combine` avec les reconnaissances de C_1 et de C_2. Pour ce faire, les places `Success`(C_1) et `Success`(C_2) sont fusionnées avec `Operand1` et `Operand2`, récupérant ainsi les reconnaissances de C_1 et de C_2 pour réaliser des reconnaissances de la conjonction.

Les réseaux relatifs à C_1 et C_2 sont activés par la transition `Activate` : les places `Start`(C_1) et `Start`(C_2) sont fusionnées respectivement avec `StartSub1` et `StartSub2` pour réaliser cette activation. Ainsi, dans le cas de l'imbrication dans une séquence, le réseau ne commence pas à reconnaître la conjonction tant qu'il n'y a pas de reconnaissance du premier membre de la séquence, dans la place `Start`, à compléter.

Les places `Present` et `Wini` sont fusionnées pour propager correctement les entiers stockés.

Le jeton de contrôle commence par activer l'initialisation du réseau qui se fait par la transition `Accu`, donc la place `Go` du réseau est la place `GoInit`. Il active ensuite l'activation (par la transition `Activate`) des places `Start` des réseaux relatifs à C_1 et C_2, donc `EndInit` est fusionnée avec `GoAct`. Lorsque l'activation est réalisée, le jeton de contrôle est dédoublé par la transition `ActOK` pour parcourir en parallèle les réseaux relatifs à C_1 et C_2 : pour ce faire, `GoL` et `GoR` sont fusionnées respectivement avec `Go`(C_1) et `Go`(C_2). Une fois que les deux réseaux ont été entièrement parcourus, les deux jetons de contrôle sont regroupés dans `GoComb1` et `GoComb2` car ces places sont fusionnées avec `End`(C_1) et `End`(C_2). Ceci permet de tirer la transition `Combine` qui achève le traitement de la conjonction, ce qui se traduit par le fait que le jeton de contrôle arrive dans la place `EndAnd` qui constitue la place `End` du réseau. Les reconnaissances de la chronique se trouvent alors dans la place `Success(AND)` qui représente donc la place `Success` du réseau.

Formellement, nous définissons le réseau comme suit :

$$
\begin{aligned}
N'(C_1\&C_2) = Fusion(&\{N'(C_1), N'(C_2), \texttt{OP}_{\texttt{AND}}\},\\
&\{(\texttt{GoL}, \{\texttt{GoL}, \texttt{Go}(C_1)\}), (\texttt{GoR}, \{\texttt{GoR}, \texttt{Go}(C_2)\}),\\
&\ (\texttt{EndFor}(C_2), \{\texttt{EndFor}(C_2), \texttt{GoFor}(C_1)\}),\\
&\ (\texttt{Present}(C_1), \{\texttt{Present}(C_1), \texttt{Present}(C_2), \texttt{Present}(\texttt{AND})\}),\\
&\ (\texttt{StartSub1}, \{\texttt{StartSub1}, \texttt{Start}(C_1)\}),\\
&\ (\texttt{StartSub2}, \{\texttt{StartSub2}, \texttt{Start}(C_2)\}),\\
&\ (\texttt{Events}(C_1), \{\texttt{Events}(C_1), \texttt{Events}(C_2)\}),\\
&\ (\texttt{WiniIn}(C_1), \{\texttt{WiniIn}(C_1), \texttt{WiniIn}(C_2)\}),\\
&\ (\texttt{WiniOut}(C_1), \{\texttt{WiniOut}(C_1), \texttt{WiniOut}(C_2), \texttt{Wini}(\texttt{AND})\}),\\
&\ (\texttt{Success}(C_1), \{\texttt{Success}(C_1), \texttt{Operand1}\}),\\
&\ (\texttt{Success}(C_2), \{\texttt{Success}(C_2), \texttt{Operand2}\}),
\end{aligned}
$$

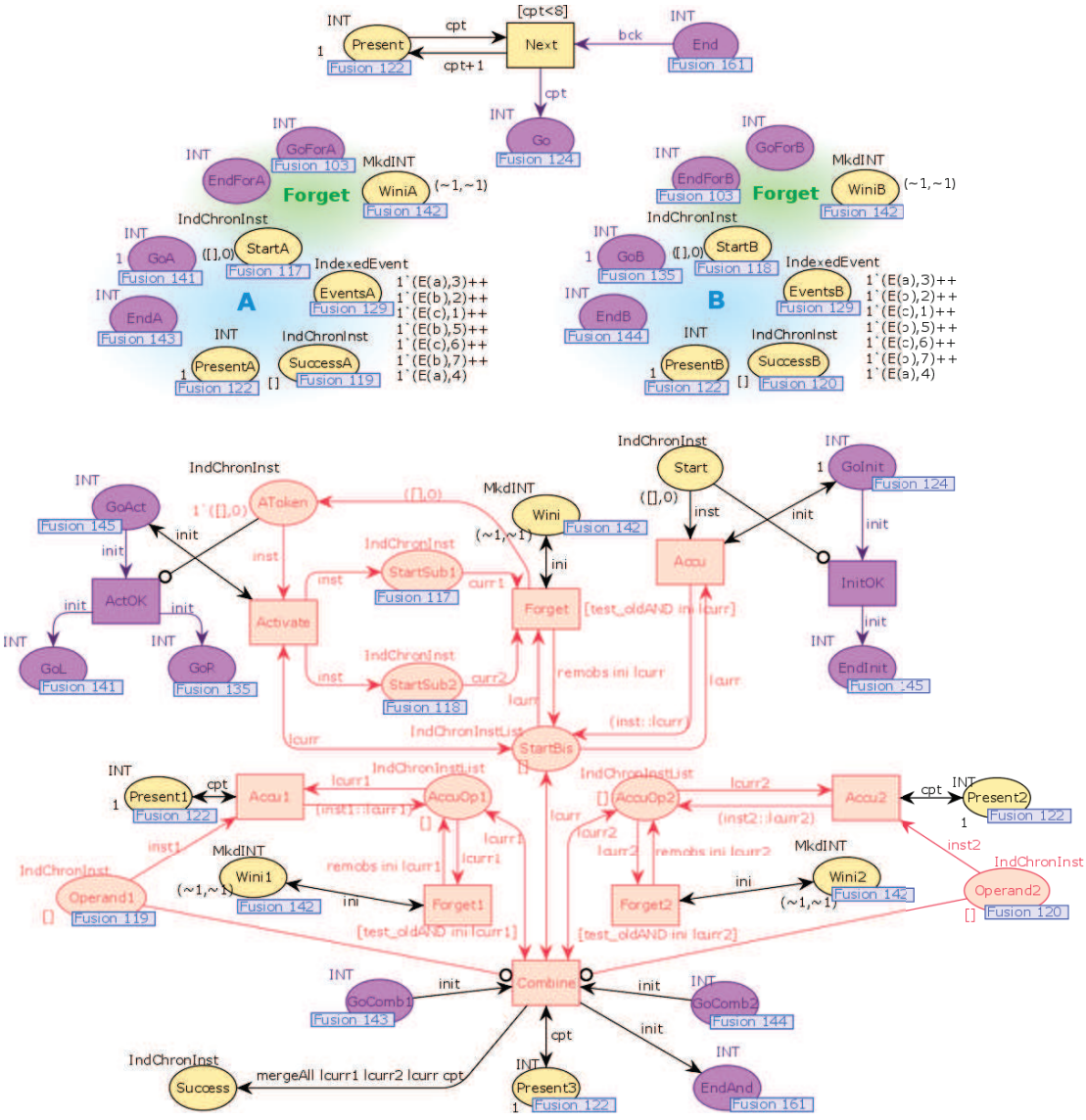


Figure 4.23 – Structure générale du réseau reconnaissant $A\&B$

$(\texttt{GoComb1}, \{\texttt{GoComb1}, \texttt{End}(C_1)\}), (\texttt{GoComb2}, \{\texttt{GoComb2}, \texttt{End}(C_2)\})\ \})$

Nous définissons également inductivement les places principales du réseau :

$$
\begin{aligned}
\texttt{Present}(C) &= \texttt{Present}(C_1) \\
\texttt{Start}(C) &= \texttt{Start}(\texttt{AND}) \\
\texttt{Success}(C) &= \texttt{Success}(\texttt{AND}) \\
\texttt{WiniOut}(C) &= \texttt{WiniOut}(C_1) \\
\texttt{WiniIn}(C) &= \begin{cases} \texttt{WiniIn}(C_1) & \text{si } \texttt{WiniIn}(C_1) \neq \emptyset \\ \texttt{WiniIn}(C_2) & \text{sinon} \end{cases} \\
\texttt{Events}(C) &= \texttt{Events}(C_1) \\
\texttt{Go}(C) &= \texttt{GoInit} \\
\texttt{End}(C) &= \texttt{EndAnd} \\
\texttt{GoFor}(C) &= \texttt{GoFor}(C_2) \\
\texttt{EndFor}(C) &= \texttt{EndFor}(C_1)
\end{aligned}
$$

Puis nous fusionnons le réseau $N'(C_1 \& C_2)$ avec le compteur d'évènements pour définir le réseau $N(C_1 \& C_2)$ comme dans (4.1).

Reconnaissance d'une absence : $C = (C_1) - [C_2[$

La structure générale du réseau reconnaissant $(A\ B) - [D[$ est présentée dans la Figure 4.24. La problématique de la combinaison des reconnaissances (d'une part les reconnaissances de l'absence et d'autre part les reconnaissances de la première partie de la séquence dans le cas d'une combinaison avec une séquence) dans le cas d'une absence est similaire à celle rencontrée dans une conjonction. La place `Start` et la place `Success` du réseau sont donc les places $\texttt{Start}(\texttt{ABS})$ et $\texttt{Success}(\texttt{ABS})$.

La place `Go` du réseau est la place `toAbs1` qui débute donc le traitement de l'absence. Le jeton de contrôle est alors transféré à la place `InAbs11` qui est fusionnée avec $\texttt{Start}(C_1)$, le jeton va donc parcourir le réseau relatif à C_1. Les places $\texttt{End}(C_1)$ et $\texttt{Start}(C_2)$ sont fusionnées, donc le jeton parcourt ensuite le réseau relatif à C_2.

Ceci établit les reconnaissances de C_1 et de C_2, il s'agit maintenant de mettre à jour la valeur de l'entier de repère `Wini` et pour cela le jeton rentre dans le mécanisme de l'opérateur $\texttt{OP}_{\texttt{ABS}}$: les places $\texttt{End}(C_2)$ et `GoUp` sont fusionnées. Il y a alors deux cas de figure :

— soit la valeur de `WiniBe` doit être mise à jour car il y a eu une nouvelle reconnaissance de C_2, et alors le jeton est transféré dans la place `EndUp` qui est fusionnée avec $\texttt{GoFor}(C_1)$: le jeton de contrôle va parcourir le mécanisme de Forget du réseau relatif à C_1 pour supprimer les reconnaissances rendues inadéquates par la nouvelle reconnaissance de C_2, puis, comme $\texttt{EndFor}(C_1)$ est fusionnée avec `InAbs12`, le jeton de contrôle se retrouve dans `InAbs12` ;

— soit il n'y a aucune mise à jour à effectuer, et alors le jeton de contrôle est directement transféré dans la place `InAbs12` car celle-ci est fusionnée à `Leave`.

Alors, il ne reste plus qu'à combiner les reconnaissances de l'absence avec d'éventuelles reconnaissances d'une première partie d'une séquence imbriquant l'absence, et le jeton parcourt donc la

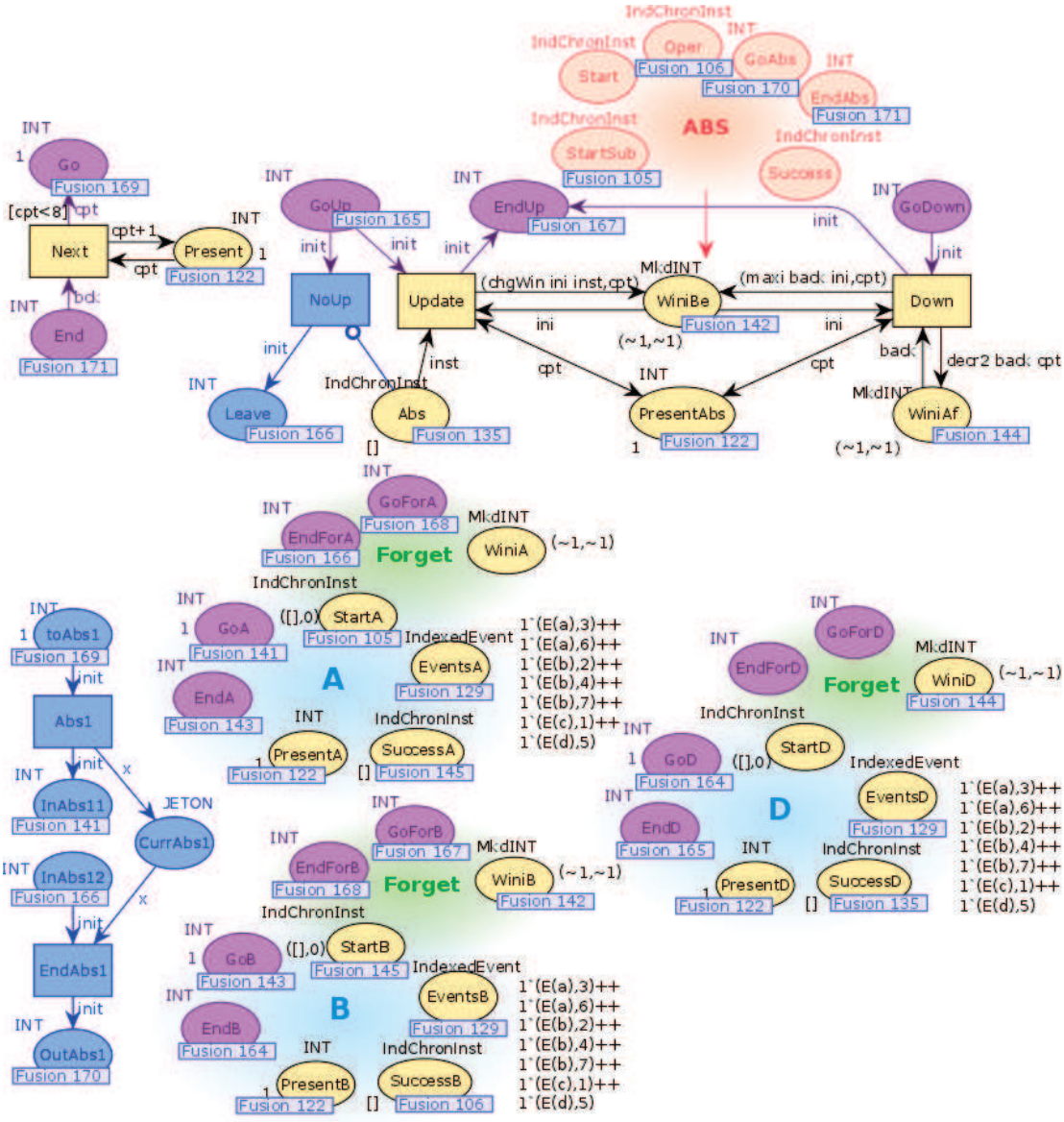


Figure 4.24 – Structure générale du réseau reconnaissant $(A\ B) - [D[$

structure rouge `ABS`, comme `OutAbs1` est fusionnée avec `GoAbs`. Á la fin de ce parcours, le jeton de contrôle se trouve dans la place `EndAbs` qui est la place `End` du réseau.

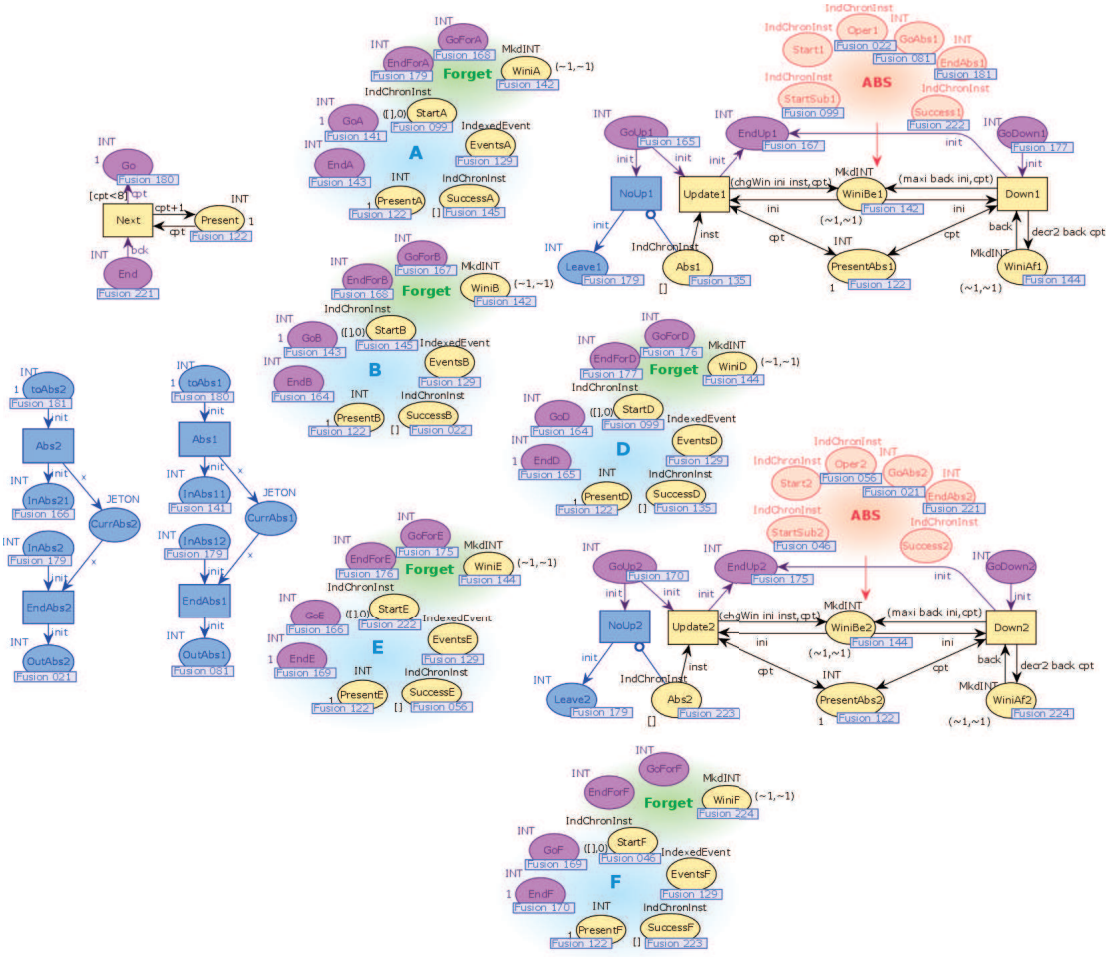


FIGURE 4.25 – Structure générale du réseau reconnaissant $((A\ B) - [D[\ E) - [F[$

Dans le cas d'une double absence, dont un exemple de réseau est présenté dans la Figure 4.25 avec la chronique $((A\ B) - [D[\ E) - [F[$, `EndFor`(E) est fusionnée avec `GoDown1`. Ceci permet de transférer le jeton de contrôle, avec la transition `Down`, dans la place `EndUp` tout en mettant à jour la valeur de `WiniBe` avec celle de `WiniAf`. Le jeton parcourt alors de nouveau le mécanisme Forget de l'absence interne (ici $(A\ B) - [D[$) puis est transféré dans la structure rouge `ABS` de l'absence externe (ici $-[F[$) pour achever le traitement de la chronique.

Formellement, nous définissons le réseau comme suit :

$$
\begin{aligned}
N'((C_1) - [C_2[) = Fusion(\{&N'(C_1), N'(C_2), \mathtt{OP_{ABS}}\},\\
&\{(\mathtt{Events}(C_1), \{\mathtt{Events}(C_1), \mathtt{Events}(C_2)\}),\\
&\ (\mathtt{Present}(C_1), \{\mathtt{Present}(C_1), \mathtt{Present}(C_2), \mathtt{Present(ABS)}\}),\\
&\ (\mathtt{Go}(C_1), \{\mathtt{Go}(C_1), \mathtt{InAbs1}\}), (\mathtt{EndUp}, \{\mathtt{EndUp}, \mathtt{GoFor}(C_1)\}),\\
&\ (\mathtt{End}(C_1), \{\mathtt{End}(C_1), \mathtt{Go}(C_2)\}), (\mathtt{End}(C_2), \{\mathtt{End}(C_2), \mathtt{GoUp}\}),\\
&\ (\mathtt{InAbs2}, \{\mathtt{InAbs2}, \mathtt{Leave}, \mathtt{EndFor}(C_1)\}),\\
&\ (\mathtt{WiniIn}(C_1), \{\mathtt{WiniIn}(C_1), \mathtt{WiniBe}\}),\\
&\ (\mathtt{Success}(C_1), \{\mathtt{Success}(C_1), \mathtt{Oper}\}),\\
&\ (\mathtt{Success}(C_2), \{\mathtt{Success}(C_2), \mathtt{Abs}\}),\\
&\ (\mathtt{Start}(C_1), \{\mathtt{Start}(C_1), \mathtt{Start}(C_2), \mathtt{StartSub}\})\})
\end{aligned}
$$

Nous définissons également inductivement les places principales du réseau :

$$
\begin{aligned}
\mathtt{Present}(C) &= \mathtt{Present}(C_1)\\
\mathtt{Start}(C) &= \mathtt{Start(ABS)}\\
\mathtt{Success}(C) &= \mathtt{Success(ABS)}\\
\mathtt{WiniOut}(C) &= \mathtt{WiniAf}\\
\mathtt{WiniIn}(C) &= \emptyset\\
\mathtt{Events}(C) &= \mathtt{Events}(C_1)\\
\mathtt{Go}(C) &= \mathtt{toAbs}\\
\mathtt{End}(C) &= \mathtt{OutAbs}\\
\mathtt{GoFor}(C) &= \mathtt{EndUp}\\
\mathtt{EndFor}(C) &= \mathtt{Leave}
\end{aligned}
$$

Puis nous fusionnons le réseau $N'((C_1) - [C_2[)$ avec le compteur d'évènements pour définir le réseau $N((C_1) - [C_2[)$ comme dans (4.1).

Exemple 17. Lorsque le processus de reconnaissance du réseau $N((A\ B) - [D[)$ présenté Figure 4.24 est mis en route sur le flux $\varphi = ((c,1),(b,2),(a,3),(b,4),(d,5),(a,6),(b,7))$ alors le marquage final de la place $\mathtt{Success}((A\ B) - [D[)$ indique deux reconnaissances de $(A\ B) - [D[$ avec les deux jetons suivants :

$$1`([(E(b),4),(E(a),3)],4) + +1`([(E(b),7),(E(a),6)],7)$$

Cela correspond bien à la sémantique ensembliste qui donne l'ensemble de reconnaissances suivant : $R_{(A\ B)-[D[}(\varphi,7) = \{\langle\langle(a,3),(b,4)\rangle\rangle, \langle\langle(a,6),(b,7)\rangle\rangle\}$.

Ceci achève la construction de notre modèle dit « contrôlé ».

4.2.6 Graphes d'espace d'états des réseaux contrôlés

Nous disposons maintenant d'un modèle de reconnaissance de chroniques « contrôlé » dans le sens où, pour un marquage initial lié à un flux d'évènements donné, toute séquence suffisamment longue de transitions tirées mène au marquage dans lequel peut être correctement lu l'ensemble des reconnaissances recherchées.

Nous nous étions fixés trois contraintes en début de section, à savoir :

1. avoir un modèle modulaire ;
2. maintenir de la concurrence dans les réseaux ;
3. tout en ayant un modèle « convergent » vers un même marquage à la suite du traitement d'un évènement du flux.

La construction de nos réseaux a bien respecté la première contrainte et, pour illustrer les deux autres contraintes, nous allons maintenant tirer parti des fonctionnalités du logiciel CPN Tools que nous utilisons pour modéliser nos réseaux. CPN Tools offre des outils d'analyse et notamment la génération du graphe d'espace d'états pour un réseau donné avec un marquage initial donné. Un graphe d'espace d'états présente l'ensemble des marquages atteignables du réseau, ainsi que les suites de transitions à tirer pour les atteindre. Ceci permet d'illustrer les contraintes que nous nous étions fixées car :

3. la confluence de nos réseaux se traduit par le fait que, périodiquement (à la suite du traitement complet de chaque évènement du flux), le graphe d'espace d'états converge vers un unique marquage qui correspond au marquage dans lequel peut être lu l'ensemble des reconnaissances courant associé à la chronique ;
2. la concurrence de nos réseaux se traduit par le fait qu'il y a plusieurs chemins possibles pour relier chaque paire successive de ces points de confluence.

La Figure 4.26 présente le graphe d'espace d'état du réseau reconnaissant la chronique $A \mid\mid (B\ A)$ sur le flux $((b,1),(a,2),(a,3))$. Dans CPN Tools, les différents états du marquage possibles du réseau sont représentés par des boîtes dans lesquelles apparaissent :

— un numéro identifiant l'état ;
— un couple $n : m$ indiquant le nombre n d'arcs en entrée et la quantité m d'arcs en sortie de l'état.

Le détail du marquage correspondant à chaque état peut être obtenu en cliquant sur la boîte.

Dans la Figure 4.26, il est clair que, après le traitement complet de chaque évènement, il y a un unique marquage atteint. En effet, on considère un flux de trois évènements, et les trois marquages correspondants sont les états 17, 37 et 63, ce qui montre l'aspect « convergent » de nos réseaux. Par ailleurs, il existe de nombreux chemins différents pour passer de l'un de ces états convergents au suivant, c'est-à-dire plusieurs suites des transitions, et ce, malgré la simplicité du flux d'évènements étudié à titre d'exemple. Notons que lorsque des chroniques plus complexes sont étudiées et lorsque des flux d'évènements plus longs sont considérés, la concurrence des réseaux est d'autant plus accrue.

À travers le graphe d'espace d'états, nous avons donc pu mettre en avant que les réseaux, quel que soit l'ordre des transitions tirées, produisent toujours les mêmes reconnaissances à la suite du

traitement d'un évènement, et que nous avons par ailleurs su conserver de la concurrence dans les réseaux malgré la structure de contrôle que nous avons implémentée.

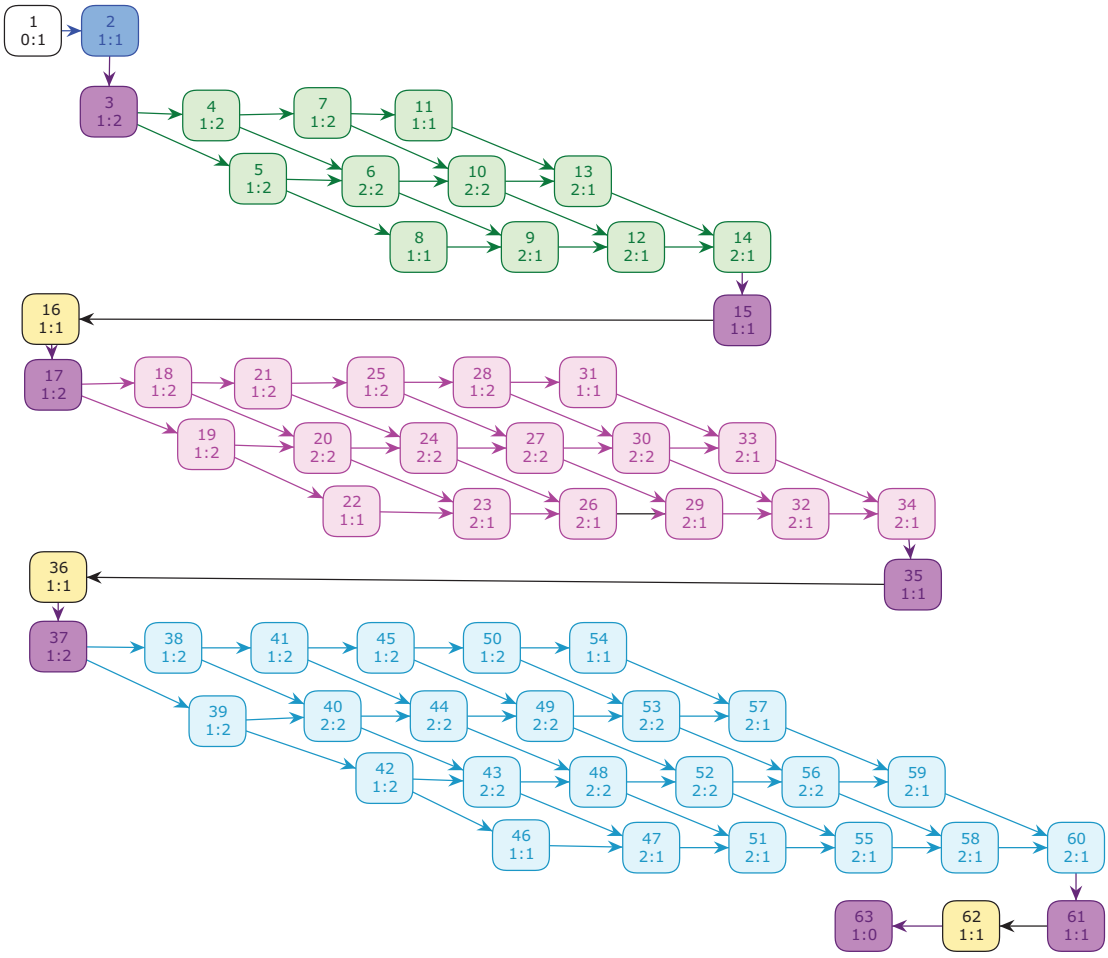


FIGURE 4.26 – Graphe d'espace d'états du réseau $N(A \mid\mid (B\ A))$ sur le flux $((b,1),(a,2),(a,3))$

4.3 Conclusion

Dans ce chapitre, nous avons fait évoluer en deux étapes principales le modèle de reconnaissance de chroniques dit « à un seul jeton » que nous avons construit dans le Chapitre 3. L'objectif était d'obtenir un modèle qui soit à la fois modulaire et convergent – dans le sens où quel que soit l'ordre des transitions tirées, il fournisse toujours le même ensemble de reconnaissances – tout en conservant un fort degré de concurrence dans le choix du tirage des transitions.

Tableau 4.27 – Récapitulatif des caractéristiques des trois modèles de reconnaissance construits

Modèle	*à un seul jeton* (Chapitre 3)	*multi-jeton* (Section 4.1)	*contrôlé* (Section 4.2)
Modularité	oui	oui	oui
Modélisation des reconnaissances	un jeton contenant la liste des reconnaissances	un jeton pour chaque reconnaissance	un jeton pour chaque reconnaissance
« confluence »	non, nécessité de suivre une stratégie de tirage précise	non, nécessité de suivre une stratégie de tirage précise	oui
Concurrence	non	début de concurrence	oui
Gestion du flux d'évènements	non	non	oui

La première étape a consisté à construire un modèle « multi-jetons » en éclatant les jetons de listes de reconnaissances des réseaux à un seul jeton en plusieurs jetons, un par reconnaissance. Cette démarche nous a permis d'approcher le contrôle intégré du tirage des transitions. Ceci constitue donc une première étape vers ce que nous appelons la confluence autonome des réseaux. Cependant, ces évolutions ne sont pas suffisantes pour que la définition d'une stratégie de tirage des transitions ne soit plus nécessaire.

Notre seconde étape a donc été de faire évoluer le modèle multi-jetons vers un modèle dit « contrôlé » en implémentant une structure de gestion des évènements et en introduisant la notion

de jeton de contrôle. Ces jetons permettent successivement l'activation de différentes parties du réseau, le tirage des transitions est alors obligatoirement correct. Nous avons porté attention à préserver le plus de concurrence possible dans les réseaux, en évitant toute sérialisation inutile de transitions. Nous avons bien entendu préservé la modularité des réseaux. Avec le visualisateur d'espace d'états de CPN Tools, nous avons vérifié que des réseaux représentatifs exhibaient bien de la concurrence, et que la « confluence » était assurée.

Le Tableau 4.27 récapitule les différentes caractéristiques de nos trois modèles.

Nous avons confronté le modèle obtenu en réseaux de Petri colorés avec les besoins d'expression en termes de chroniques dans le cadre des deux applications aux drones que nous présentons dans le chapitre suivant. Il est rapidement apparu que, comme évoqué dans le Chapitre 2, la possibilité d'exprimer des contraintes sur des attributs d'évènements est primordiale. Or notre modèle en réseaux de Petri colorés a été développé en même temps que diverses extensions syntaxiques et sémantiques du langage des chroniques étaient en cours, et, de ce fait, possède deux limitations majeures :

— D'une part, il n'implémente qu'une partie du langage des chroniques (à savoir le langage restreint aux opérateurs de séquence, de conjonction, de disjonction et d'absence) et il ne permet donc pas de reconnaître des comportements dont la description nécessite l'expression de contraintes temporelles ou de contraintes sur des attributs d'évènements. L'élaboration de ce modèle est très complexe principalement du fait de la contrainte de modularité.

— D'autre part, le modèle construit est très éloigné de la sémantique ensembliste arborescente du langage définie dans le Chapitre 2. Il faut donc démontrer l'adéquation de cette sémantique avec la sémantique opérationnelle fournie par les réseaux si l'on souhaite s'appuyer sur celle-ci, comme nous l'avons fait dans la Section 3.4.

Ces deux extensions du modèle en réseaux de Petri colorés n'ont pu être réalisées dans le temps imparti. Pour répondre aux besoins des applications, nous devons de plus disposer d'un modèle intégrable facilement dans des programmes de simulation et homogène aux outils de développement de ces simulations. Il faut que le modèle soit utilisable aisément par des ingénieurs pouvant être peu formés aux réseaux de Petri. Dans le Chapitre 5, nous allons donc développer un nouveau modèle du processus de reconnaissance répondant à ces problématiques.

Chapitre 5

Bibliothèque C++ de reconnaissance de comportements et applications à la surveillance de la sécurité d'avions sans pilote

Dans les Chapitres 3 et 4, nous avons construit un modèle en réseaux de Petri colorés implémentant le processus de reconnaissance de chroniques. Ce modèle permet de mettre en avant des caractéristiques du langage, mais il possède des limitations majeures. Nous souhaitons disposer d'un programme implémentant le processus *complet* de reconnaissance de chroniques et qui soit justifié par la sémantique ensembliste arborescente du Chapitre 2 tout en étant utilisable par des ingénieurs. Du fait des difficultés apparaissant avec les réseaux de Petri colorés, nous développons dans ce chapitre une bibliothèque en C++ dont l'algorithmique est directement calquée sur la sémantique du Chapitre 2. La structure ensembliste inductive de cette sémantique se prête parfaitement à une telle implémentation, et ceci nous permet de bénéficier de l'expressivité et de l'efficacité de C++. Notons que, comme évoqué dans la Section 1.5.1, une implémentation du processus de reconnaissance de chronique, Chronicle Recognition System/Onera (CRS/Onera), existe déjà, mais celle-ci n'a pas été élaborée à partir de la sémantique du Chapitre 2 : le langage étudié est différent et l'algorithmique repose sur une structure d'automates dupliqués radicalement différente de la structure ensembliste de notre sémantique. Le langage Chronicle Recognition System (CRS) est traduit en classes d'automates dont l'instantiation permet la reconnaissance.

Dans ce chapitre, nous allons donc implémenter dans la Section 5.1 une bibliothèque C++ fondée sur le modèle théorique de la reconnaissance de comportements présenté dans le Chapitre 2. Nous avons mis à disposition cette bibliothèque sous la licence GNU LGPL [1]. Nous utilisons ensuite cette bibliothèque pour traiter deux cas d'étude illustrant la portée de notre travail :

1. `https://code.google.com/p/crl/`

— dans la Section 5.2, nous utilisons Chronicle Recognition Library (CRL) pour surveiller, en cas de pannes, la cohérence interne entre les différents agents mis en jeu au sein d'un système d'avion sans pilote où chaque agent a sa propre vision de l'état du système [CCKP12b, CCKP13b] ;
— dans la Section 5.3, CRL nous permet d'assurer qu'un avion sans pilote respecte les procédures de sécurité associées à son évolution dans l'espace aérien, qu'il soit contrôlé ou non contrôlé [PBC+].

5.1 Développement d'une bibliothèque C++ implémentant la reconnaissance de chroniques : Chronicle Recognition Library (CRL)

Nous allons donc développer dans ce chapitre une bibliothèque C++, CRL, implémentant le processus de reconnaissance de comportements et directement calquée sur le modèle théorique du Chapitre 2. Cette bibliothèque a été déposée auprès de l'Agence de Protection des Programmes [2] et est disponible sous la licence GNU LGPL.

Nous allons maintenant décrire les différentes facettes du fonctionnement et de l'implémentation de CRL. Pour ce faire, nous allons commencer par présenter de quelle manière sont représentées les chroniques à reconnaître par le système et comment sont gérés les évènements du flux à analyser. Nous rentrons ensuite plus profondément dans le processus de reconnaissance puis détaillons quelques algorithmes pour mettre en avant la similarité avec les définitions ensemblistes du Chapitre 2 (Définition 16). Nous indiquons ensuite comment le problème de la gestion du temps continu a été traité, puis nous introduisons la notion de fenêtre de validité qui permet d'optimiser les performances du système mais qui ne fait pas partie du cadre théorique défini dans le Chapitre 2.

Définition des chroniques à reconnaître

La bibliothèque CRL manipule des *moteurs de reconnaissance* auxquels doivent être fournies la ou les chroniques à reconnaître.

Comme dans le cadre théorique présenté dans le Chapitre 2, une chronique est représentée par un arbre binaire dont les nœuds et les feuilles sont respectivement des opérateurs et des évènements simples. Lorsque l'utilisateur définit les chroniques à reconnaître par le système, il peut doter chaque niveau d'une chronique (nous entendons par là, chaque nœud de l'arbre représentant la chronique) des éléments suivants, comme dans la définition théorique du langage :

— un prédicat à vérifier exprimant des contraintes sur des attributs associés aux évènements du flux ;
— une fonction de transformation d'attributs, qui permet de calculer de nouveaux attributs à partir des propriétés des évènements du flux.

Ces prédicats et ces fonctions n'ont aucune restriction si ce n'est leur signature. Ce sont des fonctions C++ fournies par l'utilisateur, permettant ainsi d'exploiter pleinement le langage de

2. Inter Deposit Digital Number : IDDN.FR.001.440022.000.R.P.2013.000.20900

programmation. Ceci offre une forte expressivité et une grande flexibilité tout en restant strictement dans le cadre théorique posé dans le Chapitre 2.

À chaque opérateur du langage des chroniques est associée une classe C++ dans la bibliothèque. Ceci permet à l'utilisateur de ne définir qu'une seule fois des sous-classes de chroniques spécifiques et ainsi de construire facilement plusieurs chroniques similaires. Ces sous-classes peuvent posséder des prédicats et des fonctions de transformation d'attributs dépendant de variables de la sous-classe. Cette facilité d'écriture sera largement employée et donc illustrée dans le cadre de l'application présentée dans la Section 5.3.

Gestion du flux d'évènements

Un moteur de reconnaissance donné est doté d'un *flux tampon*. Il traite les évènements au fur et à mesure, et calcule l'ensemble de toutes les reconnaissances de chaque chronique, en indiquant, pour chaque reconnaissance, quels sont les évènements qui en sont à l'origine.

Dans cette implémentation les évènements sont représentés par des triplets `(nom, date, ordre)` et non plus seulement par un couple formé d'un nom et d'une date :

— le `nom` est associé à un identifiant numérique unique qui a une correspondance dans un dictionnaire de noms d'évènements – en effet, le processus de reconnaissance est fondé sur la comparaison de noms d'évènements, donc pour limiter le temps de calcul nous comparons des identifiants numériques ;

— la `date` correspond à la date d'occurrence de l'évènement, plusieurs évènements peuvent être datés au même instant ;

— l'`ordre` est distinct pour chaque évènement, ce qui fournit un ordre total sur les évènements et permet de trier le flux tampon à traiter.

Deux évènements peuvent avoir lieu à la même `date` mais leurs `ordres` doivent être différents ce qui assure le déterminisme du système[3]. Un évènement peut également être intégré au flux tampon sans être muni de date ou d'ordre. Dans ce cas, l'évènement est daté au `temps courant` et/ou un ordre lui est donné selon les dates et ordres du reste des évènements du flux tampon.

Processus général de reconnaissance d'une chronique

Le processus de reconnaissance est défini par induction pour chaque classe d'opérateur. Il calcule deux ensembles pour chaque nœud de l'arbre – et donc pour chaque sous-chronique des chroniques étudiées – après chaque évènement :

— l'ensemble des reconnaissances général de la sous-chronique ;

— l'ensemble des nouvelles reconnaissances apparues suite à l'évènement traité.

Cet ensemble de nouvelles reconnaissances est donc vidé après le traitement de chaque évènement. Il permet d'assurer que seules des nouvelles reconnaissances sont ajoutées à l'ensemble de reconnaissance général – évitant ainsi des doublons éventuels – et, pour des raisons d'optimisation, il offre un critère pour déterminer s'il y a des nouvelles reconnaissances au niveau des ancêtres du nœud.

3. Notons que dans le formalisme théorique, l'ordre total est assuré par le fait que deux évènements ne peuvent se produire à la même date (*cf.* p. 8).

Ce processus de reconnaissance est illustré sur la chronique peu complexe (E || F)&G avec le flux d'évènements $\{(e, d_e, 1), (h, d_h, 2), (g, d_g, 3), (f, d_f, 4)\}$. Il s'agit ici d'expliciter l'évolution des ensembles de reconnaissances donc, pour plus de clarté, on ne considère ni prédicat ni fonction de transformation d'attributs. Le Tableau 5.1 présente les ensembles de reconnaissances généraux ainsi que les ensembles de nouvelles reconnaissances en vert pour chaque sous-chronique et après chaque évènement du flux tampon. Ces ensembles sont organisés selon la structure arborescente de la chronique et l'ensemble des reconnaissances de la chronique recherchée est l'ensemble des reconnaissances de la racine de l'arbre. Dans notre exemple, après avoir traité les quatre évènements, il y a deux reconnaissances et l'ensemble final des reconnaissances est $\{\langle(e, d_e, 1), (g, d_g, 3)\rangle, \langle(g, d_g, 3), (f, d_f, 4)\rangle\}$. Notons que l'évènement $(g, d_g, 3)$ participe aux deux reconnaissances.

Tableau 5.1 – Évolution des ensembles de reconnaissance pour la chronique (E || F)&G sur le flux d'évènements $\varphi = (e, h, g, f)$.

	$(e, d_e, 1)$	$(h, d_h, 2)$	$(g, d_g, 3)$	$(f, d_f, 4)$
E	$\{(e, d_e, 1)\}$ $\{(e, d_e, 1)\}$	$\{(e, d_e, 1)\}$ $\{\}$	$\{(e, d_e, 1)\}$ $\{\}$	$\{(e, d_e, 1)\}$ $\{\}$
F	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{(f, d_f, 4)\}$ $\{(f, d_f, 4)\}$
E || F	$\{(e, d_e, 1)\}$ $\{(e, d_e, 1)\}$	$\{(e, d_e, 1)\}$ $\{\}$	$\{(e, d_e, 1)\}$ $\{\}$	$\{(e, d_e, 1), (f, d_f, 4)\}$ $\{(f, d_f, 4)\}$
G	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{(g, d_g, 3)\}$ $\{(g, d_g, 3)\}$	$\{(g, d_g, 3)\}$ $\{\}$
(E || F)&G	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{\langle(e, d_e, 1), (g, d_g, 3)\rangle\}$ $\{\langle(e, d_e, 1), (g, d_g, 3)\rangle\}$	$\{\langle(e, d_e, 1), (g, d_g, 3)\rangle, \langle(g, d_g, 3), (f, d_f, 4)\rangle\}$ $\{\langle(g, d_g, 3), (f, d_f, 4)\rangle\}$

Écriture des chroniques avec CRL

L'écriture de chroniques est facilitée par l'utilisation du C++. Ce langage permet en effet de redéfinir des opérateurs ou de définir ce que l'on appelle des macros (via le préprocesseur). Il est alors aisé de créer des chroniques et la lisibilité est correcte : par exemple l'opérateur « && » du C++ a été utilisé pour représenter la chronique conjonction, le « + » pour la séquence, etc.

Sur l'exemple précédent avec la chronique $(E \parallel F)\&G$, on peut définir en quelques lignes un moteur de reconnaissance, y insérer la chronique recherchée et la tester en envoyant un flux d'évènements :

```
RecognitionEngine engine;
engine.addChronicle ( ($(E)||$(F)) && $(G) );
engine << "E" << "H" << "G" << "F" << flush;
```

Dans cet exemple, les symboles « && » et « || » sont des opérateurs C++ correspondant aux chroniques de conjonction et de disjonction, et « $ » est un symbole de macro du préprocesseur C/C++ qui crée une chronique pour la reconnaissance d'un évènement simple dont le nom est passé en argument de la macro.

On le voit, l'écriture dans la bibliothèque CRL a été facilitée au maximum, dans l'optique de permettre à des non-spécialistes du langage C++ de parvenir à l'utiliser. Bien sûr, l'utilisation des prédicats et des fonctions de création d'attributs nécessite plus d'investissement de la part de l'utilisateur, mais de nombreux exemples sont à sa disposition en guise d'inspiration.

Algorithmes spécifiques à chaque opérateur et calqués sur la sémantique

Une fonction `process`, définie pour chaque classe d'opérateur, est chargée du traitement des évènements : elle met à jour inductivement les ensembles de reconnaissances et renvoie un booléen (`hasNewRecognitions`) qui indique si la chronique possède de nouvelles reconnaissances. Elle correspond directement à l'implémentation des ensembles de reconnaissances de la Définition 16 (p.55). Pour illustrer le lien direct avec la sémantique ensembliste dont découlent les algorithmes de traitement des évènements, nous donnons dans l'Algorithme 1 celui relatif à la séquence.

Notons que nous disposons dans CRL d'une date et d'un ordre pour chaque évènement, alors que, dans la définition théorique du langage, un évènement était seulement daté. Dans l'implémentation des différents opérateurs, nous utilisons, selon les opérateurs, parfois l'ordre et parfois la date pour spécifier les contraintes temporelles. Ainsi, les opérateurs de disjonction, de séquence (comme on le voit dans l'Algorithme 1), de conjonction, d'absence, de cut, et de changement d'état reposent sur l'ordre, alors que les opérateurs meets, overlaps, starts, during, finishes, equals, lasts, at most, at least, et then sont fondés sur les dates des évènements. Ceci permet une distinction plus fine des différents opérateurs.

Gestion du temps continu

La gestion d'un modèle de temps continu et la prise en compte de contraintes sur des délais soulèvent des problèmes d'implémentation : il faut pouvoir déterminer à quels instants le système de reconnaissance doit être interrogé pour toujours obtenir les reconnaissances dès que possible et ne pas instaurer de délai de traitement supplémentaire au calcul des reconnaissances.

Il est clair que le système doit être observé à chaque occurrence d'évènement, tout évènement pouvant faire évoluer un ensemble de reconnaissances. Cependant, ceci n'est pas suffisant pour les chroniques incluant un délai comme C `then` δ où il y a une évolution des reconnaissances indépendamment de l'occurrence d'un évènement.

Le système ne pouvant être surveillé constamment, on implémente une fonction « Look-ahead » telle que définie dans la Section 2.5 (p.64). Cette fonction renvoie le prochain instant où re-examiner le système. La Propriété 6 assure qu'il n'y a pas d'évolution du système entre deux instants fournis par la fonction, ce qui valide l'implémentation.

Algorithme 1 Fonction `process` pour la séquence $C_1\ C_2$

Entrée : *date* courante
Entrée : évènement *evnmt* à traiter
Sortie : booléen indiquant si la chronique possède des nouvelles reconnaissances

```
si la chronique a déjà été traitée alors
  renvoie objet.hasNewRecognitions
fin si
objet.hasNewRecognitions ← faux
objet.C1.process(date,evnmt)
si objet.C2.process(date,evnmt) alors
  pour tout r2 ∈ objet.C2.newRecognitionSet faire
    pour tout r1 ∈ objet.C1.recognitionSet faire
      si Omax(r1) < Omin(r2) [4] alors
        si le prédicat est vérifié sur r1 et r2 alors
          r ← ⟨r1, r2⟩
          X*_r ← X*_r1 ∪ X*_r2
          si il y a une fonction de transformation d'attributs alors
            X_r ← X_r ∪ D∘f[X*_r1 ∪ X*_r2]
          fin si
          objet.newRecognitionSet ← objet.newRecognitionSet ∪ r
          objet.recognitionSet ← objet.recognitionSet ∪ r
          hasNewRecognitions ← vrai
        fin si
      fin si
    fin pour
  fin pour
fin si
alreadyProcessed ← vrai
renvoie objet.hasNewRecognitions
```

Fenêtre de validité

Du fait de notre volonté d'établir la liste exhaustive et détaillée des reconnaissances de chaque chronique (notions de multiplicité et d'historisation évoquées dans la Section 1.1), les ensembles de reconnaissances manipulés peuvent vite devenir d'une taille conséquente et le temps de calcul ne peut aller qu'en s'allongeant. En effet, si l'on prend l'exemple trivial de la conjonction $A\&B$, lorsqu'un évènement A a eu lieu, il sera apparié avec *tout* évènement B futur ou passé. Même sur cet exemple simpliste, la taille des ensembles de reconnaissance peut exploser si les évènements A et B sont fréquents. Comme nous souhaitons, en plus de l'historisation et de l'exhaustivité, avoir un processus de reconnaissance utilisable en temps réel, il est important de pouvoir limiter au maximum le temps de calcul.

Nous avons donc mis en place la notion de *fenêtre de validité* d'une reconnaissance qui ne fait pas

4. Les fonctions $O_{\max}(\cdot)$ et $O_{\min}(\cdot)$ sont les fonctions analogues à $T_{\max}(\cdot)$ et $T_{\min}(\cdot)$ mais relatives à l'ordre et non à la date.

partie du cadre théorique du Chapitre 2. L'utilisateur peut définir une fenêtre de validité représentée par une durée de péremption qui s'applique sur les reconnaissances. Celles-ci sont supprimées du moteur lorsque la différence entre leur date de reconnaissance et le temps courant est supérieure ou égale à la durée de péremption. Ceci permet d'élaguer les ensembles de reconnaissances en supprimant les reconnaissances les plus anciennes, et donc de limiter les agencements possibles de nouvelles reconnaissances.

L'utilisateur peut appliquer le principe de fenêtre de validité aux chroniques de son choix et à la profondeur qu'il souhaite. Il peut définir une durée de péremption juste à un niveau de profondeur d'une chronique donnée (c'est-à-dire à une sous-chronique particulière), ou bien sur l'ensemble des sous-chroniques d'une chronique (ce qui signifie que l'on élague tous les ensembles de reconnaissances partielles liées à la chronique), ou sur l'intégralité d'un moteur de reconnaissance. Plusieurs valeurs différentes peuvent être attribuées à la durée de péremption selon la chronique et le contexte étudié, ce qui permet à l'utilisateur d'adapter le processus notamment selon les fréquences d'occurrence des évènements observés. En effet, comme évoqué précédemment, un tel procédé n'a d'intérêt que si les évènements constituant la chronique sont très fréquents. Dans d'autres cas où les évènements concernés sont rares, il peut être au contraire plus intéressant de conserver l'intégralité de l'historisation.

Nous avons donc décrit dans cette section (5.1) l'implémentation de la bibliothèque CRL permettant d'effectuer de la reconnaissance de chroniques. Nous allons maintenant traiter deux cas d'étude illustrant les capacités de la bibliothèque et du langage des chroniques.

5.2 Surveillance de cohérence au sein d'un UAS en cas de pannes

Comme évoqué dans la Section 1.6 (p.39), dans le cadre de l'industrie aérospatiale par exemple, les méthodes formelles, et, plus spécifiquement, les systèmes d'analyse d'évènements complexes sont des outils significatifs pour l'étude de systèmes critiques. L'insertion d'avions sans pilotes, Unmanned Aircrafts (UAs), dans l'espace aérien (contrôlé ou non) rendrait possible de nombreuses applications civiles. Il y a donc une forte volonté générale pour avancer dans cette voie. L'un des problèmes de sécurité principaux à résoudre est d'arriver à assurer la cohérence globale du système, ce qui est nécessaire pour piloter en sécurité un UA.

Nous nous attaquons à ce problème dans le cadre de l'analyse de sécurité de missions, en offrant la possibilité de détecter des états d'incohérence entre les différentes entités composant le système. Nous formalisons ces situations incohérentes à l'aide de chroniques pour ensuite pouvoir les détecter automatiquement à l'aide de CRL. Ceci permet d'offrir à la fois les deux opportunités suivantes :

— en situation réelle, surveiller de manière autonome le système en déclenchant automatiquement une alarme en cas de situation incohérente ;
— dans le cadre d'une simulation, avoir un outil d'assistance pour l'amélioration de la conception du système.

Notre travail repose sur une partie du projet d'Insertion des Drones dans l'Espace Aérien et Sécurité (IDEAS) et touche aux problèmes de cohérence dans un système d'avion sans pilote, Unmanned

Aircraft System (UAS), en cas de pannes éventuellement multiples.

5.2.1 Description de l'architecture du système d'avion sans pilote étudié

Cette application s'inscrit donc dans la lignée de certains travaux réalisés pour le projet IDEAS centré sur la certification des drones pour leur insertion dans l'espace aérien. Commençons par décrire le système que nous étudions.

Officiellement, un drone, ou Unmanned Aircraft (UA), est défini par la Federal Aviation Administration (FAA) et l'Organisation de l'Aviation Civile Internationale (OACI) [Hou11] comme « *an aircraft that is operated without the possibility of direct human intervention from within or on the aircraft* ». Le système étudié est l'Unmanned Aircraft System (UAS) défini comme « *an unmanned aircraft and associated elements [...] required for the pilot in command to operate safely and efficiently in the national airspace system* ».

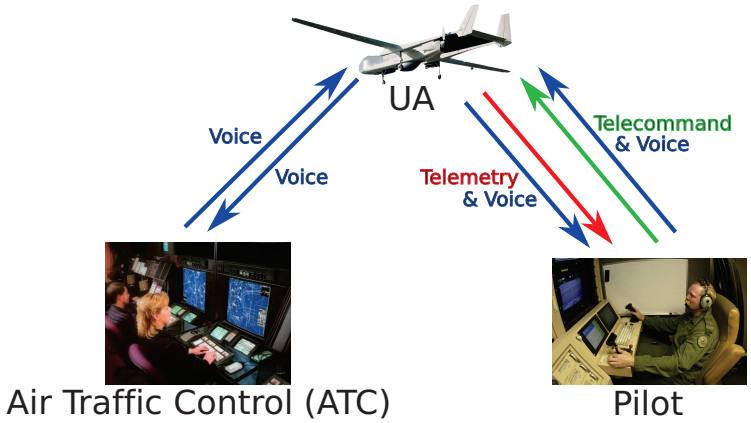


Figure 5.2 – Exemple des modes de communication entre l'ATC, l'UA et la RPS (Pilot)

Il existe plusieurs types d'architecture d'UAS qui correspondent à différents éléments associés à l'UA et à diverses liaisons de communication entre ces éléments et l'UA. Nous utilisons l'architecture correspondant à la Figure 5.2. Le système est alors composé de l'UA, d'une station de contrôle Remote Pilot Station (RPS), où exerce le pilote, et de l'Air Traffic Control (ATC), selon les liaisons de communication représentées. L'UA est sous la responsabilité et le contrôle du pilote. L'ATC assure la sécurité de la zone de l'espace aérien dont il est chargé. Il y a quatre types de liaisons entre ces éléments. Le pilote et l'ATC échangent par communications orales via ondes radio (Voice) relayées par le drone. Le pilote manœuvre l'UA à distance via la télécommande (Telecommand (TC)). Par ailleurs, le drone envoie à la station de contrôle un certain nombre de paramètres de vol via la télémesure (Telemetry (TM)). La fonction d'anti-collision « *sense and avoid* » (S&A) informe le pilote de routes de collision. Nous la regroupons avec la télémesure. La caractéristique principale de cette architecture est que l'UA en vol sert de nœud de communication entre les deux éléments au sol qui sont le pilote et l'ATC.

Le flux dynamique des données entre les différents agents du système est donc très élaboré, et il l'est d'autant plus entre les différents systèmes mis en jeu lorsque l'on considère plusieurs UASs. De plus, chaque agent déduit de ses propres observations l'état des autres agents. Dans le cas d'une panne, la situation peut donc être très complexe, ce qui est encore intensifié dans le cas de pannes multiples. Par conséquent, ces systèmes hautement automatisés sont très critiques, ce qui nécessite de fortes garanties certifiant l'absence de risque. Les méthodes formelles comme notre système de reconnaissance de comportements s'offrent donc comme une solution de choix pour ce problème.

Nous utilisons notre système de reconnaissance de chroniques pour surveiller la cohérence entre les différents agents au sein d'un UAS en cas de pannes. Dans le cadre du projet IDEAS, les pannes pouvant survenir au sein de l'UAS ont été étudiées [Lan09]. Des scénarios, comme celui de la Figure 5.3 pour le cas d'une panne de TC, ont été établis pour codifier le comportement à suivre et ainsi assurer la sécurité du système et de son environnement même en cas critique. Considérons par exemple la situation suivante d'un UA dont la TC a eu des problèmes de fonctionnement pour être ensuite complètement hors d'usage. La procédure adéquate a été correctement suivie et l'ATC a activé le mode d'urgence correspondant au déroutement de l'UA. Cependant, la TC est soudainement récupérée. Le pilote est alors occupé à rediriger son UA sur sa route initiale et il oublie de prévenir l'ATC pour annuler la procédure d'urgence. Dans cette situation, l'ATC considère que l'UA est en mode de déroutement vers un aéroport proche alors qu'il est en fait sur une autre route. Il est important de pouvoir détecter ce genre de situation critique car l'ATC ne possède alors pas les informations appropriées pour correctement organiser la circulation du trafic aérien et la séparation entre les différents appareils.

Nous nous sommes donc proposés d'effectuer un procédé de certification permettant de vérifier la synchronisation des différents agents d'un UAS en cas de pannes éventuellement multiples.

5.2.2 Modélisation du problème

La première étape de notre travail consiste à mettre en place formellement le problème. Nous étudions les diagrammes de [Lan09]. Celui relatif à la panne de TC est présenté dans la Figure 5.3. Afin d'exploiter les scénarios de panne proposés dans le projet IDEAS, nous avons normalisé ces diagrammes. Pour ce faire, nous avons fait appel aux diagrammes de classes et aux diagrammes états-transitions du langage Unified Modeling Language (UML) [UML11]. Nous commençons par rappeler brièvement le formalisme des deux types de diagrammes UML que nous employons. Nous établissons ensuite un modèle UML de l'UAS, puis nous traitons d'abord le cas d'une simple panne de TC et ajoutons ensuite la gestion d'une double panne avec la panne radio.

Le langage UML : le diagramme de classes et le diagramme états-transitions

Le langage UML (Unified Modeling Language, communément traduit par « langage de modélisation unifié ») est un langage de modélisation graphique orientée objet. Dans ce formalisme, il existe différents types de diagrammes. Nous en utilisons deux :

— le *diagramme de classes* : diagramme structurel représentant une vue statique des différentes classes composant le système ;

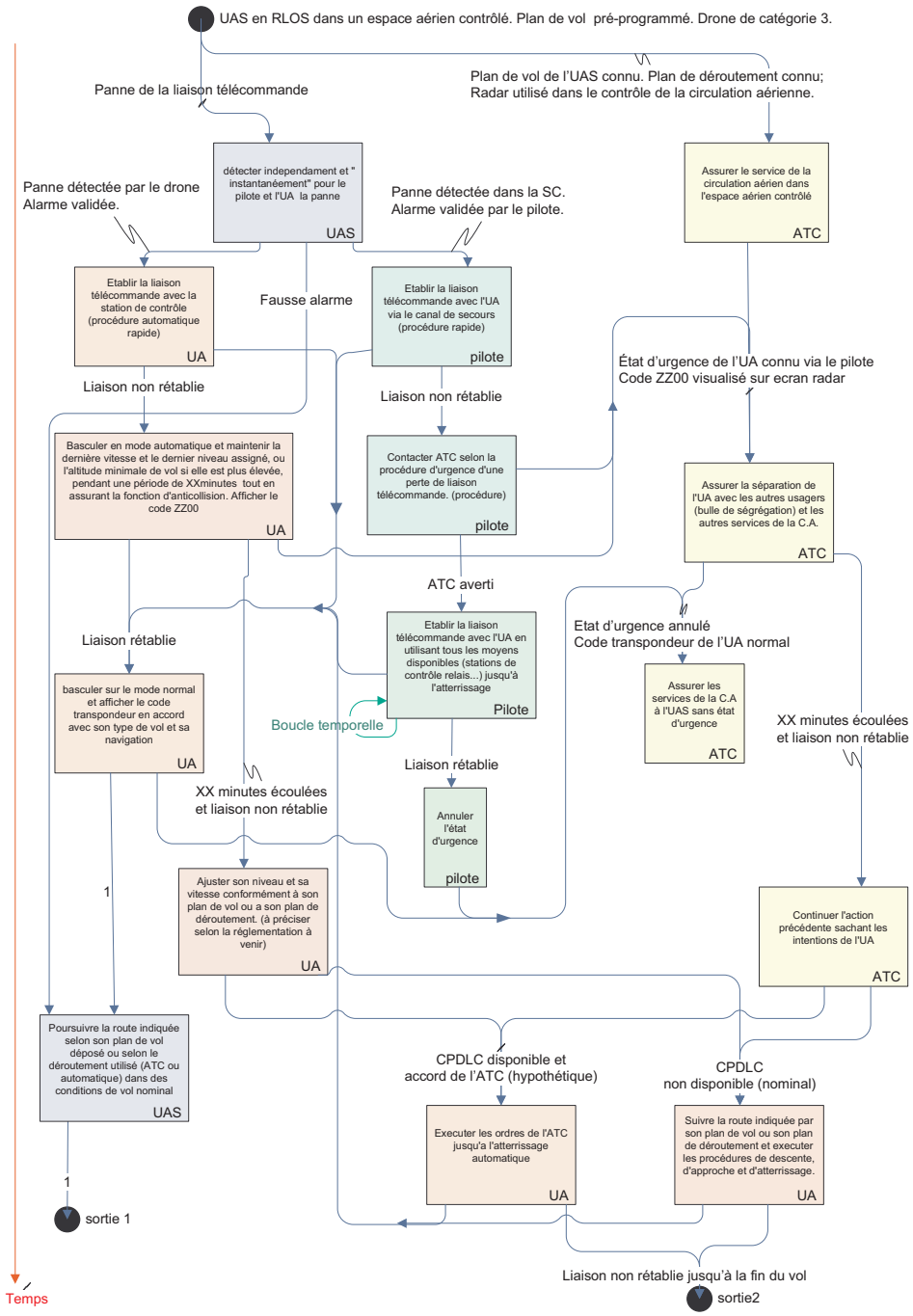


Figure 5.3 – Scénario de panne de télécommande [Lan09]

— le *diagramme états-transitions* : diagramme comportemental offrant une vue dynamique du système en permettant de décrire sous forme de machine à états son évolution.

Un diagramme de classes est composé de *classes* reliées par des *liens d'association* et *d'agrégation*. Une classe correspond à un ensemble d'objets vérifiant des attributs. Elle est représentée par un rectangle dont la partie supérieure (qui est la seule détaillée dans nos diagrammes) contient le nom de la classe. Dans la Figure 5.4 qui présente le système que nous étudions, il y a douze classes. Les relations d'association, représentées par un simple trait, sont des connexions sémantiques (liens logiques) entre au moins deux classes, comme entre les classes `RPS`, `UA`, et `ATC-UA` dans la Figure 5.4 qui sont ainsi associées car elles représentent les différentes entités de notre système. Les relations d'agrégation sont des relations d'association exprimant une subordination entre deux classes. Elles sont représentées par un trait sur l'une des extrémités duquel figure un losange. Ceci traduit que la classe à cette extrémité contient l'autre classe. Par exemple, dans la Figure 5.4, la classe `RPS` contient trois sous-classes qui correspondent à la TC, la TM et la connexion avec l'ATC. Aux bouts des liens peuvent figurer des indices de multiplicité indiquant le nombre exact d'instances de chaque classe. Un entier n correspond à exactement n instances, et * correspond à un nombre quelconque d'instances.

Un diagramme états-transitions est, comme son nom l'indique, composé d'*états* reliés par des *transitions* (*cf.* Figure 5.5). Un état modélise une situation durant laquelle une certaine condition invariante est maintenue. Il est représenté par un rectangle aux coins arrondis contenant son nom : par exemple, dans la Figure 5.5, l'état nominal de la TC pour le pilote, `RPS_TC_Nominal`. Plusieurs états munis des transitions qui les relient peuvent être regroupés dans un seul et même *super-état* qui permet de structurer le diagramme. Ainsi, dans la Figure 5.5, nous avons pu distinguer les états relatifs à chacune de trois entités, dans `UA`, `RPS`, et `ATC`. Un super-état peut être divisé par des traits pointillés pour former plusieurs zones concurrentes. Il s'agit alors d'un *état composite*. Les diagrammes de chaque zone sont exécutés en parallèle, mais, au sein de chaque secteur, les états ne sont pas concurrents. Ceci nous permet d'étudier en parallèle, par exemple, du point de vue de l'`UA`, les états des la `TC`, du `Code` et du `Pilot`. Chaque région d'un état composite doit nécessairement contenir un et un seul *état initial* représenté par un cercle plein et relié à l'état qui sera actif par défaut dans cette zone. Il y a donc trois états initiaux dans l'état composite `UA`. Les transitions permettent de passer d'un état à un autre sous certaines conditions. Elles sont représentées par des flèches et sont étiquetées éventuellement d'un *évènement*, d'une *garde* et d'une *action* sous le formalisme : `évènement` [`garde`]/`action`. Une transition donnée est déclenchée si son évènement a lieu et si sa garde, à valeur booléenne, est vérifiée. Le diagramme change alors d'état actif et l'action de la transition est effectuée. Par exemple, dans l'`UA`, pour passer de l'état `Nominal_Code` à l'état `Code_7600`, il faut que l'évènement `Code to 7600` ait lieu, que la garde `[in UA TC Nominal]` soit vérifiée (c'est-à-dire que l'`UA` soit dans l'état `UA_TC_Nominal`), et, lors du changement d'état, le message `ATC change code to 7600` est envoyé. Les transitions liant un état initial à l'état par défaut ont un statut particulier et ne peuvent être étiquetées que par une seule action.

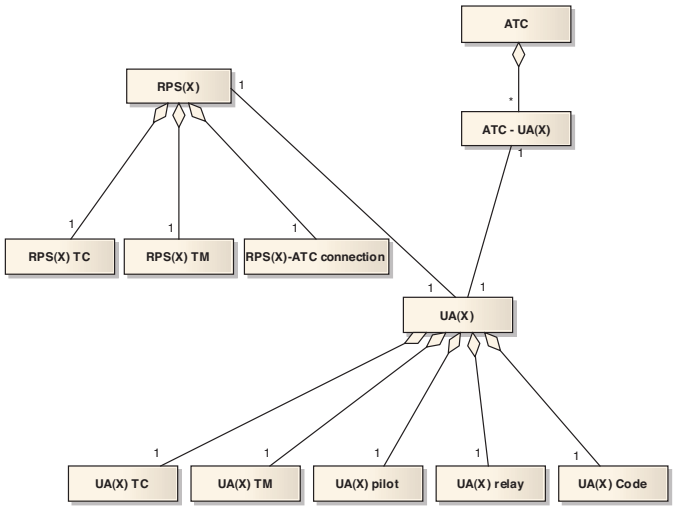


Figure 5.4 – Diagramme de classes du système

Positionnement du problème : diagramme de classe du système

Afin de positionner le problème et de définir clairement le système, nous modélisons tout d'abord l'UAS avec un diagramme de classes présenté Figure 5.4. Sur ce diagramme, il apparait qu'à un drone (`UA(X)`) est associé un unique pilote (`RPS(X)`) et une unique partie de l'ATC ($\texttt{ATC} - \texttt{UA(X)}$). Le drone est composé de plusieurs éléments : la télécommande (`UA(X) TC`), la télémesure (`UA(X) TM`), l'auto-pilote (`UA(X) Pilot`), le relai de radio (`UA(X) Relay`), et le code qu'il envoie à l'ATC (`UA(X) Code`). De même, la RPS est constituée d'une télécommande (`RPS(X) TC`), d'une télémesure (`RPS(X) TM`) et d'une connexion à l'ATC ($\texttt{RPS(X)} - \texttt{ATC connection}$). Pour une liaison donnée, nous distinguons donc son fonctionnement aux deux extrémités : par exemple, pour la télécommande, l'émission d'instructions de la part du pilote, `RPS(X) TC`, et la réception de ces instructions par le drone, `UA(X) TC`, ce qui permet de modéliser plus finement une panne.

Perte de télécommande

La première panne à laquelle nous nous sommes intéressés est la panne de télécommande : le pilote reçoit des informations du drone via la télémesure mais il ne peut pas émettre d'ordre au drone. Nous avons commencé par la construction d'un diagramme états-transitions associé à la procédure à suivre en cas de panne de TC. Dans la section suivante, nous traitons le cas de la panne radio que nous intégrons dans le diagramme états-transitions initial afin de pouvoir traiter la situation des deux pannes simultanées. La Figure 5.5 présente le diagramme résultant de ces deux études. Ses états ainsi que leur signification sont regroupés dans le Tableau 5.6 ; les états en vert correspondent aux états ajoutés lors de la seconde étude pour traiter le cas de la panne radio.

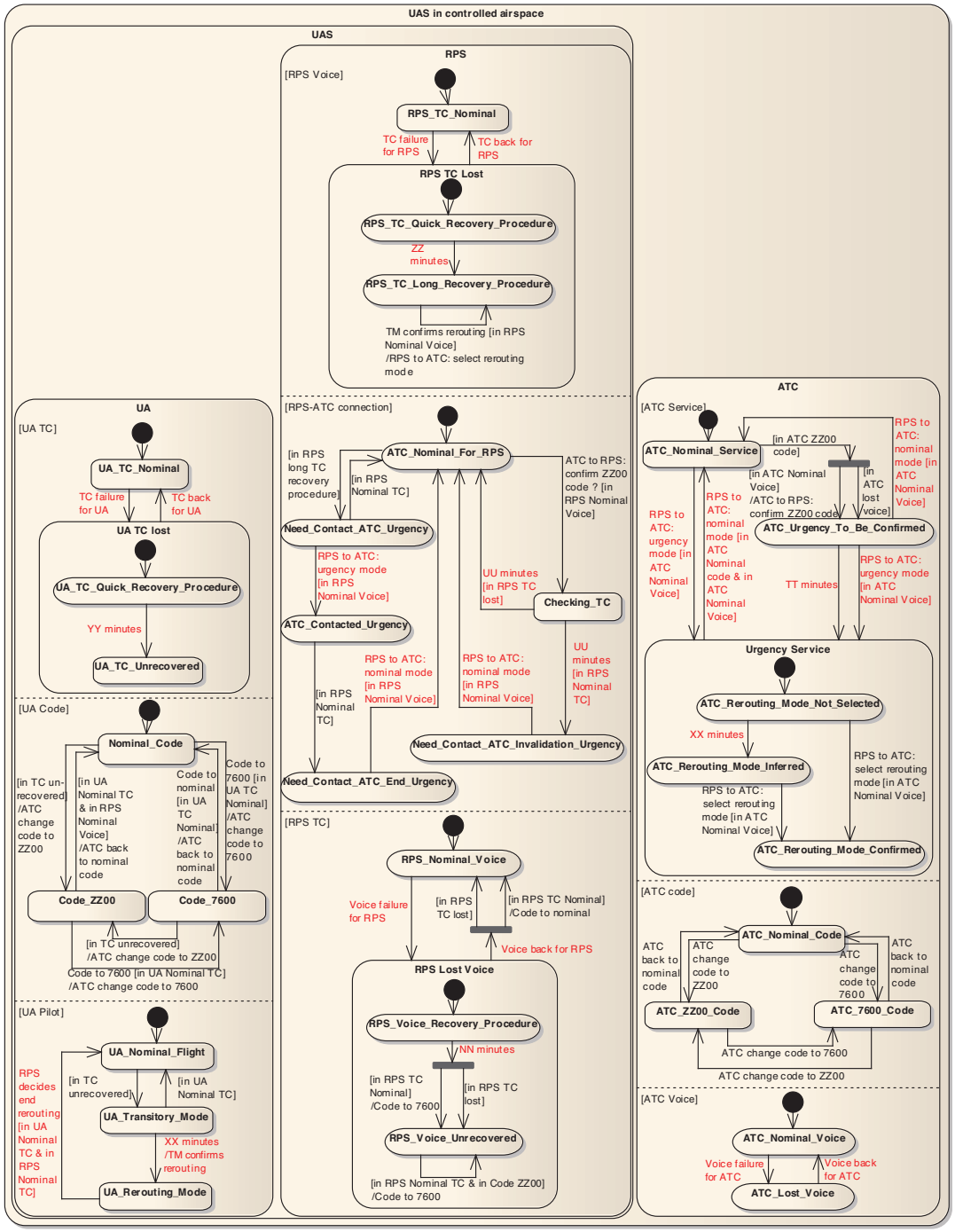


FIGURE 5.5 – Diagramme états-transitions de la perte de télécommande et de radio

Décrivons maintenant le comportement souhaité du système en cas de panne de télécommande.

En l'absence de panne, le drone est en état nominal (TC nominale, code nominal envoyé à l'ATC, vol nominal) et il a un plan de vol pré-programmé qui est connu du pilote et de l'ATC. La RPS et l'ATC sont alors aussi en état nominal (TC nominale ainsi qu'ATC nominal pour la RPS, et service nominal et code nominal pour l'ATC).

Lorsque le drone perçoit la perte de liaison TC, il commence par suivre une procédure automatique rapide visant à rétablir le lien perdu. En cas d'échec au bout de YY minutes, il abandonne cette procédure, envoie le code `ZZ00` à l'ATC, et bascule alors en mode de vol transitoire. Ce mode varie selon le drone. Cela consiste en un comportement fixe durant une durée déterminée (comme, par exemple, maintenir la dernière vitesse et le dernier niveau de vol assignés). Si la télécommande recommence à fonctionner pendant ce mode, le drone repasse en état nominal, que ce soit au niveau de la TC, du code envoyé à l'ATC ou du vol. Après XX minutes, si la liaison n'est toujours pas rétablie, le drone quitte le mode de vol transitoire et passe alors en procédure de déroutement. Il s'agit d'atterrir dans l'aéroport de déroutement le plus proche défini dans son plan de vol. Si la liaison TC est restaurée alors que le drone est en procédure de déroutement, c'est au pilote de décider s'il souhaite que le drone retourne en état nominal ou s'il juge qu'il est plus prudent de continuer la procédure de déroutement.

Parallèlement, lorsque la RPS perçoit la perte de liaison TC, le pilote commence par suivre une procédure rapide pour tenter de rétablir le lien. Si cette procédure est toujours infructueuse au bout de ZZ minutes, le pilote passe alors dans une procédure longue où il va tenter de rétablir la liaison jusqu'à ce qu'elle soit effectivement restaurée ou bien que le drone atterrisse. Lorsque le pilote passe dans cet état, il doit contacter l'ATC et l'informer que le mode d'urgence est activé. De plus, lorsque la télémesure lui indique que l'UA est en déroutement, le pilote doit également en avertir l'ATC. Si la liaison est rétablie durant cette procédure, la télécommande, du point de vue du pilote, retourne en état nominal et le pilote doit le signaler à l'ATC.

Lorsque l'ATC réceptionne le message de la RPS indiquant l'activation du mode d'urgence, l'ATC passe alors en service d'urgence, tout d'abord dans un premier mode où le déroutement n'est pas encore activé. C'est après que la RPS ait informé l'ATC du déroutement que celui-ci se met en mode de déroutement confirmé. Si l'ATC est dans le premier mode et qu'il ne reçoit aucun message au bout de XX minutes, il passe alors en mode de déroutement inféré, pour passer ensuite en mode de déroutement confirmé dès qu'il reçoit le message correspondant de la RPS. De plus, à la réception d'un message de la RPS informant d'un retour en mode nominal, l'ATC revient aussi en service nominal.

Il reste une situation qui n'a pas été décrite : celle où l'ATC, suite à un changement de code à `ZZ00` réalise qu'il y a un problème de télécommande avant le pilote (c'est-à-dire que celui-ci n'a pas encore envoyé de message à l'ATC l'informant du mode d'urgence). L'ATC envoie alors un message à la RPS pour lui demander confirmation du code `ZZ00`. La RPS vérifie alors l'état de la TC. Au bout de UU minutes, il envoie alors le message adapté à l'ATC : ou bien il active le mode d'urgence, ou bien il infirme le code `ZZ00`.

TABLEAU 5.6 – Récapitulatif des états du diagramme UML de la Figure 5.5

État	**Signification**
`UA Nominal TC`	Pour le drone, la télécommande est en état nominal.
`Quick TC recovery procedure`	Le drone suit une procédure rapide visant à rétablir la liaison TC.
`TC unrecovered`	Le drone arrête d'essayer de rétablir la liaison TC.
`Nominal code`	Le code envoyé à ATC par le drone est nominal.
`ZZ00 code`	Le code envoyé à l'ATC par le drone est le code d'urgence `ZZ00`.
`7600 code`	Le code envoyé à l'ATC par le drone est le code d'urgence `7600`.
`UA Nominal flight`	Le drone est en vol nominal.
`UA Transitory mode`	Le drone est en mode transitoire de vol (mode automatique).
`UA Rerouting mode`	Le drone est en déroutement (atterrissage sur l'aéroport de déroutement le plus proche).
`RPS Nominal TC`	Pour le pilote, la télécommande est en état nominal.
`RPS quick TC recovery procedure`	Le pilote suit une procédure rapide visant à rétablir la liaison TC.
`RPS long TC recovery procedure`	Le pilote suit une procédure longue visant à rétablir la liaison TC.
`ATC nominal for RPS`	Pour le pilote, l'ATC est en mode nominal.
`Need to contact ATC (urgency procedure)`	Le pilote doit contacter l'ATC pour l'avertir de l'état d'urgence.
`ATC contacted (urgency mode)`	Pour le pilote, l'ATC est en état d'urgence.
`Need to contact ATC (end of urgency procedure)`	Le pilote doit contacter l'ATC pour clore l'état d'urgence.
`Checking TC`	Le pilote vérifie l'état de la TC.
`Need to contact ATC (invalidation of urgency)`	Le pilote doit contacter l'ATC pour invalider le mode d'urgence de ce dernier.
`RPS Nominal Voice`	Pour le pilote, la liaison radio est nominale.
`RPS Voice Recovery Procedure`	Le pilote suit une procédure visant à rétablir la liaison radio.
`RPS Voice Unrecovered`	Pour le pilote, la liaison radio est perdue.
`Nominal service`	L'ATC est en service nominal.
`Urgency to be confirmed`	L'ATC a diagnostiqué un état d'urgence et en attend la confirmation de la part du pilote.
`ATC Rerouting mode not selected`	L'ATC est en mode d'urgence et pense que le drone n'est pas en déroutement.
`ATC Rerouting mode inferred`	L'ATC est en mode d'urgence et suppose que le drone est en déroutement.
`ATC Rerouting mode confirmed`	L'ATC est en mode d'urgence et sait que le drone est en déroutement.
`ATC Nominal Code`	Le code reçu par l'ATC est le code nominal.
`ATC ZZ00 code`	Le code reçu par l'ATC est le code `ZZ00`.
`ATC 7600 code`	Le code reçu par l'ATC est le code `7600`.
`ATC Nominal Voice`	Pour l'ATC, la liaison radio est nominale.
`ATC Lost Voice`	Pour l'ATC, la liaison radio est perdue.

Perte de liaison radio

Nous allons maintenant considérer la perte de liaison radio. Dans le projet IDEAS, les diagrammes décrivant les procédures à suivre en cas de pannes sont disjoints. Or, nous souhaitons pouvoir considérer des pannes multiples simultanées, donc en normalisant ces diagrammes sous forme UML nous les regroupons en un seul. Ceci nous amène à réfléchir aux priorités à donner à des procédures concurrentes, ce qui n'était pas nécessaire lorsque les procédures étaient considérées indépendamment. Par exemple, la perte de TC doit être accompagnée de l'affichage du code transpondeur `ZZ00` alors que la perte de communication radio est associée au code `7600`. Il n'est pas possible de transmettre simultanément les deux codes, donc nous choisissons de privilégier l'affichage annonçant la panne de TC car celle-ci est jugée plus dangereuse que la panne radio.

Nous ajoutons donc un état correspondant au code du transpondeur réglé à `7600`, du côté de l'UA et du côté de l'ATC, en concurrence avec le code `ZZ00` mais c'est ce dernier qui prime. La procédure à suivre en cas de panne radio est sensiblement la même qu'en panne de TC. Les différentes entités cherchent à rétablir la liaison puis au bout d'un délai imparti renoncent et considèrent la liaison comme perdue. Comme annoncé avec les différentes priorités des codes de transpondeur, si une panne de TC se produit alors que la procédure associée à une panne radio est déjà amorcée, la procédure associée à la panne de TC prend le dessus pour les actions étant en concurrence les unes avec les autres.

Implémentation en C++ des diagrammes

Avec les diagrammes des Figures 5.4 et 5.5, nous avons donc entièrement modélisé le système et les procédures qu'il doit suivre en cas de panne radio et/ou de panne de TC. Le cycle de vie du système se reflète dans les états actifs du diagramme. Afin de pouvoir effectuer des simulations, nous avons codé ce diagramme états-transitions en C++ à l'aide de la bibliothèque Meta State Machine (MSM) [Hen11] fournie dans les bibliothèques boost (Version 1.48.0) qui permet l'implémentation directe de machines à états. Le programme décrit le fonctionnement du diagramme UML. Il prend en entrée un flux d'évènements qui correspondent aux évènements étiquetant les transitions du diagramme et indiqués en rouge – les évènements en noir résultent d'actions internes à la simulation et sont donc engendrés automatiquement selon l'évolution du système. Le flux fait évoluer les états actifs du diagramme. La succession de ces états actifs pour un flux d'évènements donnés fournit alors un scénario de simulation que l'on peut examiner.

5.2.3 Objectifs de la reconnaissance de comportements dans ce cas d'étude

Il s'agit donc d'analyser les scénarios découlant des simulations de la section précédente pour y détecter des comportements normaux ou anormaux spécifiés par des chroniques.

La simulation produite respecte exactement les directives spécifiées et requises par le projet IDEAS. Cependant, il n'est de nos jours pas encore autorisé de faire voler un UA dans l'espace aérien, qu'il soit contrôlé ou non. En effet, les réglementations comme comme celles étudiées dans ce chapitre n'ont pas encore été finalisées. Nous proposons d'étudier si certaines situations incohérentes se produisent dans les simulations, et ce afin de remplir *deux objectifs*. La reconnaissance

de comportements au cours de la simulation permet de confirmer ou d'infirmer certains choix de réglementation en faisant ressortir les différentes manières d'atteindre un état incohérent représentant un danger. *Notre premier objectif est donc d'offrir un outil d'assistance durant l'étape de développement des réglementations.* En effet, comme l'on considère des pannes multiples, le système devient très complexe et il est difficile de se le représenter entièrement correctement, ce qui rend utile une telle assistance.

À terme, les dernières causes possibles de brèches dans la sécurité du système devraient être humaines, c'est-à-dire dues au pilote ou au contrôleur du trafic aérien. La possibilité d'atteindre ce genre de situation incohérente doit rester dans le modèle car elle représente une réalité qui ne peut être évitée. Cependant, ces situations peuvent être détectées à l'aide d'un outil de reconnaissance de comportements. *Ainsi, notre second objectif est d'offrir une méthode de détection des dernières situations incohérentes atteignables ne pouvant être empêchées, permettant d'activer des alarmes et donc de réduire les risques potentiels.*

Par ailleurs, il y aurait une troisième utilisation possible de la reconnaissance de comportements, corollaire de la deuxième : l'analyse d'un incident a posteriori par dépouillement des enregistrements. Cette application ne nécessite pas de disposer d'un moyen de reconnaissance en ligne. Elle a été largement étudiée précédemment dans le cas d'analyses de simulations HLA [Ber09].

5.2.4 Écriture et formalisation des situations incohérentes à détecter

Identification des situations dangereuses

La première étape pour mettre en place un tel système est d'identifier avec l'aide d'un expert les états incohérents qui doivent être évités afin de pouvoir ensuite les spécifier au système et lancer le processus de reconnaissance. Par exemple, les comportements suivants sont considérés comme dangereux :

1. *Incoherent ATC Voice* : le code du transpondeur de l'UA indique le code `7600` à l'ATC, ce qui signifie qu'il y a une perte de liaison radio, mais le contrôleur aérien ne s'en rend pas compte, ce qui s'exprime par le fait que, dans le diagramme, l'état `ATC Lost Voice` ne devient pas actif ;
2. *Incoherent flight mode UA/ATC* : après une panne ayant été résolue, l'UA revient en mode de vol nominal mais l'ATC reste en mode d'urgence ;
3. *RPS rushed decision* : l'ATC pense que le drone est en déroutement mais ce n'est pas le cas ;
4. *ATC late* : la TC est perdue, et, depuis, δ minutes se sont écoulées sans que l'ATC passe en mode d'urgence ;
5. *ATC incoherent* : l'UA est passé en mode de déroutement, suite à quoi l'ATC a inféré par lui-même que le drone était en déroutement, mais cela n'a pas été confirmé par le pilote, même après le délai de latence imparti de δ minutes.

L'objectif est donc de mettre en place un système de reconnaissance de ces comportements. Lorsque l'un d'entre eux est détecté, il faut alors examiner quelles sont ses origines :

— si l'état incohérent est dû à un trou dans la réglementation, il faut la compléter ;

— sinon, c'est que la source est une erreur humaine, et il faut alors prévoir d'activer une alarme pour prévenir le pilote et/ou le contrôleur du trafic aérien de la situation potentiellement dangereuse.

Écriture des chroniques associées

Pour implémenter ce système, la seconde étape est de transcrire formellement ces comportements sous forme de chroniques qu'il sera ensuite possible de fournir à la bibliothèque CRL pour être reconnues. Ces chroniques permettront donc de surveiller le système.

Il s'agit d'abord de spécifier quels évènements de base sont choisis pour construire les chroniques. Dans notre formalisme, les évènements de base sont ponctuels, et nous choisissons donc les évènements d'entrée et de sortie des différents états actifs du diagramme de la Figure 5.5, ce qui permet de décrire intégralement l'evolution du système. L'entrée et la sortie d'un état `xxxx` sont dénotés respectivement `to_xxxx` et `from_xxxx`.

Les situations incohérentes décrites précédemment s'expriment alors formellement avec les chroniques suivantes :

1. *Incoherent ATC Voice*
   (`to_ATC_Nominal_Code to_ATC_7600_Code` then 5) − [`to_ATC_Lost_Voice`]
2. *Incoherent flight mode UA/ATC*
   (`from_UA_Nominal_Flight`
   ((`to_UA_Nominal_Flight` then 10) − [`from_UA_Nominal_Flight`]))
   −[`to_ATC_Nominal_Service`]
3. *RPS rushed decision*
   (((`to_UA_Nominal_Flight` || `to_UA_Transitory_Mode`) then δ)
   −[`to_UA_Rerouting_Mode`])
   & `to_ATC_Rerouting_Mode_Confirmed`
4. *ATC late*
   (`to_UA_TC_Unrecovered` then δ)
   −[`to_Urgency_Service` || `to_UA_TC_Nominal`]
5. *ATC incoherent*
   `to_UA_Rerouting_Mode` ((`to_ATC_Rerouting_Mode_Inferred` then δ)
   −[`to_ATC_Rerouting_Mode_Confirmed` || `to_UA_Nominal_Flight`])

Nous avons donc, d'une part modélisé entièrement le système, et d'autre part écrit des chroniques pour le superviser. Il s'agit maintenant d'intégrer cela avec la bibliothèque CRL pour compléter la mise en place du système et remplir les objectifs fixés.

5.2.5 Utilisation de CRL pour reconnaître les situations incohérentes

Nous allons donc utiliser CRL pour reconnaître les chroniques définies dans la section précédente dans des simulations produites par le programme issu du diagramme états-transitions modélisant le système. Plus précisément, le mode opératoire est le suivant :

— nous lançons un scénario constitué d'une séquence d'évènements (qui sont des évènements marqués en rouge et étiquetant des transitions du diagramme) en fournissant au programme boost modélisant le diagramme un fichier d'entrée contenant cette séquence d'évènements ;
— ce scénario fait évoluer les état actifs du système ;
— ceci produit un fichier de sortie contenant la séquence des évènements d'entrée et de sortie des états actifs du diagramme, ce qui constitue le flux à analyser car ce sont ces évènements que nous avons choisis pour construire nos chroniques ;
— nous analysons ce flux d'évènements avec CRL pour chercher à reconnaître les chroniques définies dans la section précédente.

Un premier scénario : une erreur humaine

Étudions un premier scénario très simple qui est à considérer comme un exemple instructif : une perte de la liaison radio se produit, et celle-ci n'est reconnue que par le pilote (évènement `Voice failure for RPS`). Nous faisons donc tourner la simulation avec ce simple évènement. L'évolution des états actifs du diagramme modélisant le système est fourni à la bibliothèque CRL sous la forme de la séquence des entrées et sorties des états concernés, ce qui donne le résultat suivant :

```
t = 0  Engine created
t = 0  Added chronicle :
         ([(to_ATC_Nominal_Code to_ATC_7600_Code) + 5] − to_ATC_Lost_Voice)
t = 0  Added Event : to_ATC_Nominal_Code
t = 0  Added Event : Voice_failure_for_RPS
t = 0  Added Event : from_RPS_Nominal_Voice
t = 0  Added Event : to_RPS_Voice_Recovery_Procedure
t = 4  Added Event : from_RPS_Voice_Recovery_Procedure
t = 4  Added Event : to_RPS_Voice_Unrecovered
t = 4  Added Event : from_Nominal_Code
t = 4  Added Event : to_Code_7600
t = 4  Added Event : from_ATC_Nominal_Code
t = 4  Added Event : to_ATC_7600_Code
t = 9  Chronicle recognition :
         ([(to_ATC_Nominal_Code to_ATC_7600_Code) + 5] − to_ATC_Lost_Voice)
         Reco Set = {⟨⟨(to_ATC_Nominal_Code, 0), (to_ATC_7600_Code, 4)⟩, (t, 9)⟩}
```

La chronique *Incoherent ATC Voice* est donc reconnue : `to_ATC_Nominal_Code` à l'instant $t = 0$ a été suivi de `to_ATC_7600_Code` à l'instant $t = 4$, puis, jusqu'à l'instant $t = 9$, l'évènement pouvant annuler la reconnaissance (à savoir que l'ATC reconnaisse la panne radio) n'a pas eu lieu. Grâce à l'historisation des évènements, on peut voir dans l'ensemble de reconnaissances indiqué que la cause de l'incohérence est une inattention de la part du contrôleur du trafic aérien qui n'a pas réagi à la panne. Nous sommes donc dans le second cas où la source d'erreur est humaine, et une alarme doit être activée par la chronique pour prévenir l'ATC de la situation et tenter de rétablir une situation correcte.

Un deuxième scénario : un trou dans la réglementation

Considérons maintenant un deuxième scénario, plus complexe, impliquant plusieurs pannes : la perte de la liaison radio est reconnue à la fois par le pilote et l'ATC (évènements `Voice failure for RPS` et `Voice failure for ATC`), peu après, une panne de TC, également reconnue à la fois par le pilote et l'ATC, se produit (évènements `TC failure for UA` et `TC failure for RPS`). Cependant, la TC est rétablie 15 min plus tard (évènements `TC back for UA` et `TC back for RPS`). À ce moment là, le pilote décide que la situation n'est pas trop inquiétante (une simple panne radio peut effectivement être considérée comme peu alarmante, elle peut être provisoire et due à un simple obstacle) et il règle l'UA de nouveau sur son mode de vol nominal (évènement `RPS decides end rerouting`). Lorsque l'on fait tourner la simulation sur ce scénario, on obtient le résultat suivant sur CRL (où l'on ne rapporte pas cette fois-ci tous les évènements du flux) :

```
⋮
t = 65  Chronicle recognition :
            ([from_UA_Nominal_Flight ([to_UA_Nominal_Flight + 10]
            − from_UA_Nominal_Flight)] − to_ATC_Nominal_Service)
            Reco Set = {⟨(from_UA_Nominal_Flight, 35),
                        ⟨(to_UA_Nominal_Flight, 55), (t, 65)⟩⟩}
```

La chronique *Incoherent flight mode UA/ATC* est donc reconnue. Cette fois-ci, l'état d'incohérence n'est pas dû à une erreur humaine, ce qui signifie que la règlementation, représentée par le diagramme UML états-transitions, doit être corrigée. Nous avons mis en évidence le fait qu'il manque une transition dans la modélisation de l'ATC, entre `Urgency service` et `ATC_Nominal_Service`. En effet, l'ATC devrait pouvoir retourner en mode de service nominal même s'il n'y a pas de communication radio établie avec le pilote. Une transition activée par la sortie de l'état `ATC_ZZ00_Code` (qui indique la fin de la panne de TC) doit donc être ajoutée au diagramme. Une fois que cette amélioration a été effectuée, et que l'on a donc modifié le système et la procédure à suivre, le scénario précédent ne produit plus de reconnaissance de la chronique *Incoherent flight mode UA/ATC* ce qui montre que le comportement a été correctement corrigé.

Un dernier exemple

Les autres chroniques que nous avons définies dans la Section 5.2.4 sont utilisées de la même façon que les deux précédentes. Donnons un dernier scénario dangereux qui provoque cette fois-ci la reconnaissance de la chronique *ATC late*. On considère un UA dont la TC est périodiquement hors service, par exemple à cause d'un environnement perturbateur, et le code d'urgence du transpondeur associé à la perte de TC ne cesse donc de s'activer et de se désactiver. Suite à cette situation répétitive, le pilote comme l'ATC sont fatigués et moins réactifs, alors, lorsque la TC est tout d'un coup définitivement perdue, personne ne réagit. Après un délai prédéfini dans son plan de vol, l'UA commence à se dérouter vers un aéroport proche. Si ni le pilote, ni l'ATC ne réalisent le changement de situation, d'autres appareils pourraient ne pas être correctement séparés de l'UA qui, lui, ne peut plus recevoir d'instructions ce qui constitue donc un danger potentiel. La chronique *ATC late* est alors reconnue, ce qui permet de déclencher une alarme et d'éviter une issue possiblement

dangereuse.

Notons que le scénario évoqué comme exemple dans la Section 5.2.1 provoque une reconnaissance de la chronique *RPS rushed decision*.

5.3 Surveillance du bon respect de procédures de sécurité à suivre par un drone

Dans la lignée du cas d'application précédent, nous allons dans cette section utiliser notre méthode de reconnaissance de comportements autour d'un second cas lié au système critique que représente un drone, à savoir la surveillance de procédures de sécurité à suivre par un UA lorsqu'il circule dans l'espace aérien. Nous commençons par décrire brièvement le cadre du problème (5.3.1). Nous établissons ensuite les chroniques critiques à reconnaître pour mettre en place le système de surveillance (5.3.3), puis nous appliquons ce système à des scénarios de simulation dans lesquels nous cherchons à reconnaître les comportements critiques (5.3.2).

5.3.1 Cadre du problème

On considère ici un UA, intégré dans un système analogue à celui présenté dans la Section 5.2.1. Il circule successivement au travers de l'espace aérien contrôlé et non contrôlé. L'espace aérien est divisé en plusieurs zones (des boîtes tri-dimensionnelles d'une certaine classe nommée d'une lettre de A à G) et le type de zone détermine la réglementation à suivre. C'est le rôle de l'ATC de diriger les avions volant dans les zones contrôlées (classes A à E), et son objectif principal est d'éviter toute collision tout en organisant le flux de la circulation. À l'entrée et à la sortie d'une zone, un avion doit suivre des règles spécifiques, à savoir notamment régler sa fréquence radio pour communiquer avec l'ATC, respecter des points de passage (*waypoints*), et attendre d'avoir reçu une autorisation (*clearance*) avant de pouvoir entrer dans une nouvelle zone. La Figure 5.8 représente la trajectoire d'un drone traversant plusieurs zones aux alentours de l'aéroport d'Ajaccio, ainsi que les fréquences radio associées à chacune des zones.

Comme évoqué dans le cas d'étude précédent, l'insertion des UAs dans la circulation aérienne générale est un problème d'actualité. Les avancées technologiques disponibles permettent maintenant aux UAs de réaliser de nombreuses missions très intéressantes, mais les règles strictes qui régissent le trafic aérien rendent difficile la circulation libre d'UAs, principalement pour des raisons de sûreté et de sécurité. Il est donc intéressant de fournir des mécanismes de surveillance assurant que toutes les procédures pertinentes sont correctement suivies par tout UA. Nous nous proposons de fournir un tel outil à l'aide de notre bibliothèque de reconnaissance de comportements CRL.

On considère un UA décollant de l'aéroport d'Ajaccio en Corse (France). Dans cette région, les feux de maquis sont très fréquents et la mission de l'UA est d'aller surveiller une certaine zone située en dehors de l'espace aérien contrôlé. Pour ce faire, l'UA doit traverser successivement des zones contrôlées pour ensuite quitter l'espace aérien contrôlé et atteindre son but. Nous allons exposer notre système de surveillance sur une simulation de ce drone. La trajectoire de l'UA est calculée par le battlelab de l'Onera, Battle Lab for Aerospace and Defence Experimentations (BLADE) [CBP10],

Figure 5.7 – Vue d'ensemble aérienne de la trajectoire de l'uas étudié

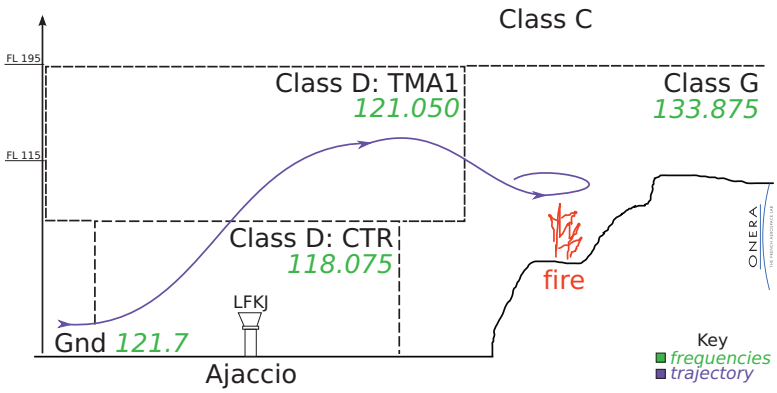


Figure 5.8 – Représentation schématique des différentes zones traversées par l'uas

qui fournit un fichier détaillant les positions successives de l'avion chaque seconde, ainsi qu'une vue d'ensemble aérienne de la trajectoire dont on a un aperçu dans la Figure 5.7. Une seconde vision (simplifiée) de la trajectoire est décrite par la Figure 5.8, mettant en avant les différentes zones de l'espace aérien à traverser par l'UA.

À partir de cet ensemble de positions, différents scénarios peuvent avoir lieu, pouvant mettre en scène aussi bien un comportement nominal de l'UA que des infractions à la réglementation. Les comportements dangereux possibles peuvent être décrits par le langage des chroniques, puis être ajoutés à un moteur de reconnaissance de CRL pour être reconnus.

5.3.2 Mise en place du système de surveillance : écriture des chroniques critiques à reconnaître

Nous allons maintenant utiliser la bibliothèque CRL pour traiter le cas d'étude présenté dans la section précédente. On considère donc un UA décollant d'Ajaccio et traversant successivement diverses zones de l'espace aérien. Il doit respecter une réglementation particulière et l'objectif est d'identifier si et quand il s'éloigne de ces procédures.

Une première étape est d'identifier les évènements qui doivent être considérés en entrée pour composer la base du flux à analyser. Dans la mise en place de cette étude, nous nous attachons à réduire la quantité de données nécessaires en entrée du processus de reconnaissance. Certains évènements n'ont pas besoin d'être fournis en entrée car ils peuvent être déduits des autres évènements. Par exemple, le décollage de l'appareil peut être inféré des coordonnées de position. Ainsi, nous réduisons la taille des données d'entrée et nous écrivons différents niveaux de chroniques où les chroniques d'un niveau sont définies à partir des chroniques des niveaux inférieurs. Ceci permet de réduire la complexité du processus de reconnaissance. Les finales du plus haut niveau sont celles qui correspondent à la reconnaissance des comportements dangereux recherchés, alors que les chroniques des niveaux inférieurs complètent et enrichissent le flux d'évènements en entrée.

Le flux d'évènements à analyser est composé des coordonnées de position ainsi que d'événements basiques liés principalement au réglage de la fréquence radio et à des échanges entre les différentes entités du système étudié (envoyer l'ordre d'un changement de fréquence, demander ou donner une clearance pour une action spécifique, informer qu'un waypoint donné est atteint...). Ces évènements sont enrichis d'attributs qui identifient par exemple l'appareil (ID), la fréquence ($FREQ$) ou le waypoint (WP) concernés.

Nous utilisons ensuite ces évènements de bas niveau pour écrire une première strate de chroniques correspondant à la reconnaissance d'évènements simples avec des contraintes sur leurs attributs, comme présenté dans le Tableau 5.9 avec les chroniques de Niveau 1. Par exemple, `OnGround` correspond à un UA donné (identifié par ID) dont le niveau de vol est inférieur à une altitude donnée (précisée par H). Un second exemple est `FreqDiffFrom` qui est reconnue lorsqu'un UA donné a une fréquence radio réglée différemment de la fréquence attendue, et qui enregistre l'instant t de la reconnaissance sous un paramètre dans une propriété pour le moment anonyme. Comme mentionné dans la Section 5.1, la définition de ces chroniques est effectuée grâce à des classes C++ qui permettent d'écrire une définition générique pour tous les UAs et toutes les fréquences, par exemple.

Tableau 5.9 – Quelques chroniques écrites pour surveiller les procédures de sécurité

	Nom	Formule de la chronique et éventuels prédicat et/ou fonction de transformation d'attributs *ou* description
Niv. 1	`OnGround`(ID, H)	`position` $\diamondsuit.id = ID \wedge \diamondsuit.fle < H$
	`FreqDiffFrom`$(ID, FREQ)$	`set_frequency` $\diamondsuit.id = ID \wedge \diamondsuit.freq \neq FREQ$ fonction de transformation d'attributs enregistrant l'instant de reconnaissance sous le nom *time*
	`ClearanceTakeOff`(ID)	`clear_takeoff` $\diamondsuit.id = ID$
	`InsideBox`(ID, BOX)	`position` $\diamondsuit.id = ID \wedge inside_{BOX}(\diamondsuit.lat, \diamondsuit.lon, \diamondsuit.fle)$
	...	...
Niv. 2	`TakeOff`(ID, H)	@(`OnGround`(ID, H) !! `AboveGround`(ID, H))
	`WaypointReached`(ID, WP)	@(`OutsideBox`(ID, WP) !! `InsideBox`(ID, WP))
	`QuitZone`$(ID, ZONE)$	@(`InZone`$(ID, ZONE)$!! `OutZone`$(ID, ZONE)$)
	...	...
Niv. 3	`NoClearanceToTakeOff`(ID)	(`ClearanceTakeOff`(ID) !! `TakeOff`$(ID, 15)$) at least 60 || ($(\tau, 0)$ `TakeOff`$(ID, 15)$) -[`ClearanceTakeOff`(ID)]
	`NoFrequencyToTakeOff`(ID)	(`FreqDiffFrom`$(ID, 118.075)$ `TakeOff`$(ID, 15)$ then 5) -[`FreqDiffFrom`$(ID, -1)$] || ((`FreqDiffFrom`$(ID, 118.075) \rightarrow x$ during (`TakeOff`(ID,15) then 5)) -[`FreqDiffFrom`$(ID, -1) \rightarrow y$], $x.time < y.time$) || ($(\tau, 0)$ `TakeOff`$(ID, 15)$ then 5) -[`FreqDiffFrom`$(ID, -1)$]
	`NoRightToTakeOff`(ID)	Interdiction de décoller sans avoir reçu de clearance au moins 1 min avant, et sans avoir réglé la fréquence radio à celle de la tour (118.075) avant ou au plus 5 s plus tard.
	`IncoherentFrequency`(ID)	Interdiction de changer de zone sans avoir réglé la fréquence radio correctement, au plus 5 s plus tard.
	`IncoherentAutoMode`(ID)	Interdiction d'entrer en mode de vol automatique sans être dans l'espace non contrôlé (classe G).
	...	...

Nous définissons ensuite des chroniques légèrement plus complexes, formant une seconde strate, et utilisant les chroniques de Niveau 1, comme exposé dans le Tableau 5.9. Nous allons détailler ici la structure de la chronique `WaypointReached`. Pour des raisons pratiques, un waypoint n'est pas représenté par un point unique dans l'espace, mais par une (petite) boîte centrée autour d'un point, ce qui permet d'avoir une certaine tolérance vis-à-vis de petites imprécisions. Le waypoint donné est atteint lorsque l'UA identifié change d'état et passe d'une position à l'extérieur de la boîte à une position dedans. La chronique est ensuite rendue ponctuelle par l'opérateur @. Ceci permet ensuite d'utiliser la chronique dans des chroniques de niveau supérieur comme s'il s'agissait d'un évènement du flux analysé, c'est-à-dire une activité datée à un instant ponctuel t correspondant à la fin de sa reconnaissance.

Contrairement aux chroniques de niveaux 1 et 2 qui pouvaient faire partie du flux d'évènements, les chroniques de Niveau 3 sont complexes et chargées de détecter des comportements dangereux anormaux. Ainsi, lors d'un vol nominal, les chroniques de Niveau 3 ne devraient pas être reconnues, contrairement à certaines chroniques de niveaux inférieurs. Nous allons étudier plus en détail l'une des chroniques de Niveau 3 présentées dans le Tableau 5.9, à savoir `NoFrequencyToTakeOff`. Cette chronique correspond à une situation où un UA décolle sans avoir activé la fréquence radio correcte. Le comportement dont il s'agit peut être reconnu dans trois situations différentes :

— « Le pilote passe à une fréquence incorrecte, puis décolle après quoi 5 s s'écoulent. Cependant, durant *tout* ce temps, la fréquence radio n'a pas été corrigée. » Notons que, dans l'absence, `FreqDiffFrom`$(ID, -1)$ désigne l'UA ID qui modifie sa fréquence radio à une fréquence différente de -1, c'est-à-dire à toute fréquence.

— « Le pilote règle la mauvaise fréquence radio dans les 5 s après le décollage, et elle n'est pas corrigée d'ici la fin de ces 5 s. » Il est nécessaire de préciser ici grâce à un prédicat que l'absence s'applique à partir de l'instant t du dernier changement incorrect de fréquence. Nous utilisons donc ici un prédicat ayant une portée sur les deux chroniques construisant une absence. Ce type de construction est très fréquent et justifie le cadre théorique mis en place dans le Chapitre 2.

— « Aucune fréquence n'est réglée avant au moins 5 s après le décollage. » Rappelons que $(\tau, 0)$ désigne un instant de temps pur de date 0 qui correspond à l'origine du temps choisie, c'est-à-dire ici au début du scénario.

Les définitions formelles des autres chroniques ne sont pas toutes précisées dans le Tableau 5.9 mais leurs structures sont comparables à celle de la chronique `NoFrequencyToTakeOff`.

Ces trois niveaux de chroniques offrent ainsi une mise en place complète pour la détection de comportements dangereux liés à des procédures de sécurité négligées.

5.3.3 Application à des scénarios de simulation avec CRL

Nous pouvons maintenant surveiller le système de drone à l'aide des chroniques définies dans la section précédente. Nous allons appliquer notre mise en place à plusieurs simulations de situations dangereuses ou non, mais cela illustre que notre cadre de travail peut s'appliquer aussi bien en temps réel car la quantité et la précision des données utilisées sont adéquates.

Comme annoncé dans la Section 5.3.1, les données simulées sont obtenues en ajoutant divers

évènements procéduraux à un fichier contenant les positions successives de l'appareil étudié. Ce fichier est généré par le battlelab BLADE de l'Onera qui fournit une trajectoire réelle au départ de l'aéroport d'Ajaccio en passant par un ensemble donné de waypoints. Les données produites sont précises et abondantes : la latitude, la longitude et l'altitude (ou niveau de vol) sont procurées toutes les secondes à la précision du millionième de degré, accompagnées de la vitesse de l'appareil.

Divers scénarios ont été testés avec la mise en place précisée dans la section précédente, dont notamment des scénarios nominaux où les chroniques de Niveaux 1 et 2 sont correctement reconnues mais les chroniques de Niveau 3 ne sont pas détectées, comme attendu. Nous allons maintenant observer plus en détail l'étude d'un scénario potentiellement dangereux.

Considérons un UA identifié par l'ID 153 qui commence par régler sa fréquence radio à 121.7, ce qui correspond à la fréquence correcte pour un appareil au sol (*cf.* Figure 5.8). Le même avion quitte alors le parking avec pour objectif un décollage proche. Une fois qu'il atteint le waypoint précédant le décollage, il informe l'ATC, qui, en retour, ordonne le passage à la fréquence radio 118.075, c'est-à-dire la fréquence associée à la zone de l'espace aérien où se déroule le décollage. Suite aux instructions, l'UA change correctement de fréquence, puis, après l'avoir réclamée, reçoit une clearance pour un décollage. 5 s plus tard, il commence à rouler et décolle ensuite 23 s après. Jusqu'à ce moment là, les procédures de sécurité ont été correctement suivies. Cependant, quelques secondes après, l'UA règle sa fréquence radio à la fréquence 121.050 qui correspond à la fréquence de la zone aérienne suivante, mais ce changement est effectué trop tôt dans la procédure. L'appareil se trouve donc dans une situation potentiellement dangereuse, puisqu'il n'est plus en contact avec le bon représentant de l'ATC.

Dans ce scénario, la fréquence requise pour le décollage est correctement réglée avant le décollage mais changée à tort peu après. Cette situation est dangereuse puisque l'ATC ne peut plus communiquer avec l'appareil, ce qui peut porter atteinte à sa mission d'éviter toute collision. Comme voulu, cette situation est correctement détectée par CRL ce qui est rapporté ci-dessous. Pour des raisons de clarté, seuls les évènements liés à l'UA 153 sont indiqués mais des évènements entrelacés concernant d'autres appareils étaient considérés et cela n'affecte pas le résultat du processus de reconnaissance. De plus, du fait de leur forte périodicité, seul un évènement de position est précisé, à titre d'exemple.

```
t = 0  Engine created
t = 0  Added chronicle : NoFrequencyToTakeOff ID=153
t = 0  Added Event : set_frequency ID=153 FREQ=121.7
t = 7  Added Event : quit_parking ID=153
t = 32 Added Event : inform_WP_reached ID=153 WP=GRD
t = 35 Added Event : order_frequency ID=153 FREQ=118.075
t = 37 Added Event : set_frequency ID=153 FREQ=118.075
t = 37 Added Event : ask_clearance_takeoff ID=153
t = 45 Added Event : clearance_takeoff ID=153
t = 50 Added Event : position LAT=41.9288 LON=8.80548 FLE=14.9821
t = 75 Added Event : set_frequency ID=153 FREQ=121.05
t = 78  Chronicle recognition :   NoFrequencyToTakeOff ID=153
        Reco Set = {⟨⟨⊥, ⟨(set_frequency, 75), ⟨(position, 73), (τ, 78)⟩⟩⟩, ⊥⟩}
```

Notons que l'ensemble des reconnaissances permet non seulement d'identifier quels sont les évènements qui ont mené à la situation dangereuse (ici, l'évènement clé est `set_frequency` à $t = 75$) mais il expose également quelle branche de la disjonction a été reconnue.

Un certain nombre de tels scénarios ont été testés avec notre cadre de reconnaissance de chroniques, de façon à couvrir une grande variété d'arrangements d'évènements possibles ; et les comportements dangereux des chroniques de Niveau 3 ont été fidèlement reconnus.

5.4 Conclusion

Dans ce chapitre, nous avons décrit le fonctionnement de la bibliothèque CRL que nous avons conçue. Nous disposons, au travers de CRL, d'une implémentation efficace d'un processus de reconnaissance de comportements à l'aide de formalisme des chroniques. Cette mise en place est fondée sur la base théorique solide présentée dans le Chapitre 2 ce qui rend CRL adaptée à être employée pour les systèmes critiques nécessitant de fortes garanties, comme les UA. En tant que bibliothèque C++, CRL est dès à présent utilisable dans un contexte industriel au travers d'une API dédiée. Nous avons déposé la bibliothèque auprès de l'Agence pour la Protection des Programmes, et elle est disponible sous la licence libre GNU LGPL [5].

Nous avons utilisé CRL pour traiter deux cas d'étude critiques liés à la sécurité d'un UA. Dans le premier cas, nous avons surveillé la cohérence entre les différents agents mis en jeu au sein d'un système de drone (UAS) dans le cadre de la procédure à suivre lors d'une ou plusieurs pannes de liaisons de communication. Dans ce contexte, le processus de reconnaissance de comportements permet non seulement d'activer des alarmes dans le cas de situations dangereuses, mais il offre aussi un outil d'assistance lors de l'élaboration des procédures à suivre en cas de panne.

Dans le second cas, nous avons surveillé un UA circulant dans l'espace aérien et devant suivre des procédures de sécurité variées et précises, dépendant des zones aériennes qu'il traverse. CRL nous permet de détecter les situations dangereuses où l'UA ne respecte pas scrupuleusement ces procédures. Des alarmes peuvent alors être déclenchées ce qui peut permettre d'éviter les dangers susceptibles de découler de la situation.

Ces deux cas d'études mettent en avant les fonctionnalités de CRL et illustrent notamment l'utilité des différents opérateurs du langage des chroniques. Notons que, dans ces deux cas, l'analyse de comportements a uniquement été effectuée sur des simulations mais cela est dû à la difficulté d'obtenir des données réelles. CRL peut aussi bien être utilisée pour l'analyse de cas réels.

5. `https://code.google.com/p/crl/`

Conclusion et perspectives

Dans le cadre de cette thèse, nous nous sommes concentrés sur la formalisation et l'extension d'un langage de description de comportements, le langage des chroniques. Nous avons également développé deux modèles d'implémentation d'un processus de reconnaissance de ces comportements pour ensuite utiliser l'un de ces deux modèles pour traiter deux cas d'application.

Nous nous sommes donc attelés à la problématique de l'analyse de flux importants d'évènements afin d'y détecter des situations complexes pouvant être aussi bien dangereuses que souhaitées, domaine du Complex Event Processing (CEP). Cette technique est applicable en ligne et procède à une historisation des évènements impliqués, ce qui permet, d'une part, de diagnostiquer dès leur occurrence les situations recherchées, et, d'autre part, de pouvoir remonter aux évènements étant à l'origine de ces situations.

Les domaines d'application de cette technique sont très variés, et nous nous sommes restreints au domaine de l'aéronautique, et plus particulièrement à l'insertion de drones civils dans l'espace aérien. Dans ce cadre, nous avons traité deux cas d'application. Tout d'abord, nous nous sommes attachés à l'analyse des réglementations à suivre par un système d'avion sans pilote en cas de pannes. Des études détaillées ont été menées à l'Onera afin d'établir des réglementations visant à introduire les avions sans pilotes dans l'espace aérien. Il s'agissait notamment d'imposer et de codifier un comportement à suivre en cas de pannes. Ces systèmes étant très critiques, il est impératif d'obtenir des garanties fortes sur ces réglementations, et ce niveau de garantie ne peut être obtenu que par des méthodes formelles, *i.e.* une analyse logique du système. Nous avons donc mis à profit notre processus de reconnaissance de comportements pour faire ressortir les situations incohérentes pouvant apparaître entre les différentes entités du système qui est composé du drone, du pilote et du contrôleur de trafic aérien, et où les échanges de données sont très complexes. Nous avons pu mettre en avant à la fois des situations incohérentes dues à une erreur humaine (de la part du pilote ou du contrôleur aérien) et des situations incohérentes dues à un trou dans la réglementation. Cette mise en place a donc permis d'exposer des points précis de la réglementation à compléter, tout en fournissant un outil pouvant activer des alarmes dans le cas de situations dangereuses dues à l'homme [CCKP13b, CCKP12b]. Une seconde application concernant l'insertion de drones civils dans l'espace aérien a également été réalisée dans le cadre de la surveillance de procédures de sécurité liées à la traversée de différents types de zones de l'espace aérien. Le processus de reconnaissance de comportements a été affecté à la reconnaissance de situations dangereuses provenant du non respect de procédures de sécurité [PBC$^+$].

La mise en place de ces applications repose sur un langage logique, le langage des chroniques, permettant de décrire formellement des agencements complexes d'évènements à l'aide de différents opérateurs temporels. Nous avons construit une formalisation mathématique complète autour de ce langage, définissant entre autres la notion d'événement et surtout celle de la reconnaissance d'une chronique, notion qui est exprimée par une sémantique ensembliste arborescente construite dans une induction structurelle. La problématique double de disposer de reconnaissances en ligne (c'est-à-dire au fur et à mesure du flux d'évènements) et avec historisation est centrale. Tout en continuant à respecter ces contraintes, nous avons étendu l'expressivité du langage en ajoutant la possibilité d'exprimer de nouvelles contraintes temporelles plus précises, et en raffinant la notion d'événement, leur donnant la possibilité d'avoir des attributs de toutes sortes sur lesquels il est ensuite possible d'exprimer des contraintes à l'aide de prédicats [PBC+]. Ces extensions sont cruciales car de nombreuses applications industrielles nécessitent de pouvoir exprimer des contraintes de corrélation entre certains attributs liés aux évènements d'une même reconnaissance.

Afin de pouvoir appliquer ce cadre théorique à la reconnaissance de comportements, nous avons réalisé deux modèles du processus de reconnaissance de chroniques. Nous avons proposé une première modélisation conçue à l'aide de réseaux de Petri colorés, un outil d'informatique théorique offrant un langage de spécification formelle bien adapté ainsi qu'une représentation graphique simple. Les outils disponibles autour des réseaux de Petri nous ont permis de simuler et de visualiser les étapes de la reconnaissance de chroniques, et aussi de tester la correction des réseaux. En outre, nous avons prouvé un résultat d'adéquation des constructions de réseaux de Petri avec la sémantique ensembliste du langage des chroniques [CCKP12a]. Nous avons également fait évoluer ce modèle de reconnaissance de chroniques en y incorporant une structure de contrôle permettant de déterminiser ce modèle initialement non-déterministe et assurant la gestion du flux d'évènements qui n'était pas encore implémentée. Nous avons réalisé ces extensions tout en conservant bien entendu à la fois une construction modulaire et de la concurrence [CCKP13a].

Parallèlement, un outil de reconnaissance de chroniques, Chronicle Recognition Library (CRL) dont les algorithmes sont directement calqués sur la sémantique ensembliste a également été élaboré en C++. Cet outil est davantage adapté à nos utilisations applicatives pour l'industrie aéronautique car il est homogène avec les outils utilisés et il permet de tirer parti de l'intégralité du cadre théorique. C'est donc CRL que nous avons utilisé pour réaliser les applications évoquées ci-dessus. La bibliothèque CRL est disponible librement sous la licence GNU LGPL [6] [PBC+, CCKP13b, CCKP12b].

De nombreuses perspectives tant théoriques qu'applicatives s'offrent dans la poursuite de ce travail. Les différents axes principaux sont les suivants :

Réseaux de Petri Une première extension du modèle en réseaux de Petri colorés serait de modifier le formalisme de représentation des reconnaissances dans les réseaux en passant des listes d'évènements à des arbres d'évènements. Ceci permettrait d'uniformiser les modes de représentation avec le cadre théorique formel et fournirait ensuite une voie pour achever la démonstration de la correction du modèle en réseaux de Petri colorés vis à vis de la sémantique ensembliste arborescente.

6. `https://code.google.com/p/crl/`

Par ailleurs, il faudrait compléter le modèle en réseaux de Petri colorés pour pouvoir prendre en compte l'ensemble des constructions du langage disponibles dans le cadre théorique. Ceci implique également de prolonger en parallèle la preuve d'adéquation.

Ce modèle étendu en réseaux de Petri pourra ensuite être utilisé dans le cadre d'une application mettant en jeu un système modélisé lui-même en réseaux de Petri colorés. Pour ce faire, il faudrait mettre en place un mécanisme de collecte des évènements à analyser et provenant du modèle. Il suffirait ensuite de connecter ce mécanisme à la place `Events` des réseaux qui doit contenir le flux des évènements à traiter.

Pour pouvoir utiliser le modèle en réseaux de Petri colorés pour traiter d'autres types d'applications, il faudra également élaborer un programme qui réponde au problème de la génération automatique de réseaux, ce qui est envisageable grâce à leur construction modulaire inductive formelle. Il s'agit alors de construire automatiquement les réseaux associés aux chroniques recherchées et le programme correspondant à ces réseaux.

Cadre théorique formel Au sein du cadre théorique formel, il serait intéressant de chercher à étendre l'expressivité du langage en ajoutant par exemple la possibilité d'exprimer des quantifications sur les attributs d'évènements, ou bien en ajoutant de nouvelles constructions permettant réaliser plus simplement la reconnaissance de répétitions. Ceci permettrait de décrire précisément des situations plus complexes et donc de traiter davantage d'applications.

Pour chercher à optimiser les formules de chroniques à reconnaître, il faudrait mettre en avant des règles de transformation pouvant mener à une formulation plus efficace de la même chronique. Un système d'analyse de satisfiabilité de chroniques pourrait également être utile à l'utilisateur.

Il serait aussi intéressant d'étudier les différents formalismes de modélisation d'incertitudes afin de définir lesquels sont adaptés à l'introduction d'incertitudes à différents niveaux du formalisme, que ce soit sur les comportements à reconnaître, les dates d'occurrence ou les autres attributs des évènements, ou même au niveau de l'occurrence elle-même des évènements.

Bibliothèque CRL et applications En parallèle de ces extensions théoriques, il faudrait faire évoluer la bibliothèque CRL pour pouvoir tirer parti de ces extensions.

L'élargissement du champ d'application de la reconnaissance de chroniques, aussi bien dans le cadre de simulations qu'en temps réel, permettrait de faire avancer CRL et de mettre en avant d'éventuelles autres évolutions nécessaires.

Pour améliorer l'efficacité en temps du programme, il serait intéressant de mettre à profit la concurrence, mise en avant notamment par le modèle en réseaux de Petri colorés, pour paralléliser le processus.

Pour finir, l'écriture de chroniques, qui représente actuellement l'une des difficultés principales dans le domaine du CEP, doit pour le moment être réalisée « à la main » par des experts. Il faudrait étudier différentes méthodes (apprentissage, statistique,...) pour mettre en place un outil d'aide à l'écriture de chroniques permettant de se rapprocher d'une couverture exhaustive des comportements à reconnaître dans le cadre d'une application donnée.

Bibliographie

[AC06] Raman Adaikkalavan and Sharma Chakravarthy. SnoopIB: Interval-based event specification and detection for active databases. *Data & Knowledge Engineering*, 59(1):139–165, 2006. *(2 citations pp. 37 et 38)*

[AD01] Armen Aghasaryan and Christophe Dousson. Mixing chronicle and Petri net approaches in evolution monitoring problems. In *Proceedings of the Eleventh International Workshop on Principles of Diagnosis (DX2001)*, 2001. *(cité p. 25)*

[AEFF12] Alexander Artikis, Opher Etzion, Zohar Feldman, and Fabiana Fournier. Event processing under uncertainty. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 32–43. ACM, 2012. *(cité p. 17)*

[AFR+10] Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic, and Rudi Studer. A rule-based language for complex event processing and reasoning. In *Proceedings of the Fourth International Conference on Web Reasoning and Rule Systems (RR)*, pages 42–57. Springer, 2010. *(4 citations pp. 20, 21, 22, et 211)*

[AFSS09] Darko Anicic, Paul Fodor, Roland Stuhmer, and Nenad Stojanovic. Event-driven approach for logic-based complex event processing. In *International Conference on Computational Science and Engineering (CSE)*, volume 1, pages 56–63. IEEE, 2009. *(cité p. 22)*

[All81] James F. Allen. An interval-based representation of temporal knowledge. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 221–226, 1981. *(cité p. 10)*

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, pages 832–843, 1983. *(2 citations pp. 10 et 47)*

[AMPP12] Alexander Artikis, Robin Marterer, Jens Pottebaum, and Georgios Paliouras. Event processing for intelligent resource management. In *Proceedings of European Conference on Artificial Intelligence (ECAI). IOS Press*, pages 943–948, 2012. *(cité p. 19)*

[Ani11] Darko Anicic. *Event Processing and Stream Reasoning with ETALIS*. PhD thesis, Karlsruhe Institute of Technology, 2011. *(cité p. 20)*

[AP09] Alexander Artikis and George Paliouras. Behaviour recognition using the event calculus. In *Artificial Intelligence Applications and Innovations III*, pages 469–478. Springer, 2009. *(2 citations pp. 17 et 39)*

[APPS10] Alexander Artikis, Georgios Paliouras, François Portet, and Anastasios Skarlatidis. Logic-based representation, reasoning and machine learning for event recognition. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. ACM, 2010. *(cité p. 18)*

[ARFS11] Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Retractable complex event processing and stream reasoning. In *Proceedings of the 5th international conference on Rule-based reasoning, programming, and applications*, pages 122–137. Springer, 2011. *(cité p. 23)*

[ARFS12] Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Real-time complex event recognition and reasoning–a logic programming approach. *Applied Artificial Intelligence*, 26(1-2):6–57, 2012. *(2 citations pp. 20 et 22)*

[ASP10a] Alexander Artikis, Marek Sergot, and Georgios Paliouras. A logic programming approach to activity recognition. In *Proceedings of the 2nd ACM international workshop on Events in multimedia*, pages 3–8. ACM, 2010. *(cité p. 39)*

[ASP10b] Alexander Artikis, Anastasios Skarlatidis, and Georgios Paliouras. Behaviour recognition from video content: a logic programming approach. *International Journal on Artificial Intelligence Tools*, 19(02):193–209, 2010. *(2 citations pp. 17 et 39)*

[ASP12] Alexander Artikis, Marek Sergot, and Georgios Paliouras. Run-time composite event recognition. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 69–80. ACM, 2012. *(2 citations pp. 9 et 19)*

[ASPP12] Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. Logic-based event recognition. *Knowledge Engineering Review*, 27(4):469–506, 2012. *(2 citations pp. 9 et 35)*

[AWG+13] Alexander Artikis, Matthias Weidlich, Avigdor Gal, Vana Kalogeraki, and Dimitrios Gunopulos. Self-adaptive event recognition for intelligent transport management. In *Big Data, 2013 IEEE International Conference on*, pages 319–325. IEEE, 2013. *(cité p. 20)*

[AWS+14] Alexander Artikis, Matthias Weidlich, Francois Schnitzler, Ioannis Boutsis, Thomas Liebig, Nico Piatkowski, Christian Bockermann, Katharina Morik, Vana Kalogeraki, Jakub Marecek, et al. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *Proceedings of the 17th International Conference onExtending Database Technology (EDBT 2014). Athens, Greece*, 2014. *(cité p. 20)*

[BCC07] Olivier Bertrand, Patrice Carle, and Christine Choppy. Chronicle modelling using automata and coloured Petri nets. In *The 18th International Workshop on Principles of Diagnosis (DX-07)*, pages 229–234, 2007. *(cité p. 30)*

[BCC08] Olivier Bertrand, Patrice Carle, and Christine Choppy. Towards a coloured Petri nets semantics of a chronicle language for distributed simulation processing. In *CHINA 2008 Workshop (Concurrency metHods: Issues aNd Applications)*, pages 105–119, June 2008. *(cité p. 30)*

[BCC09] Olivier Bertrand, Patrice Carle, and Christine Choppy. Coloured Petri nets for chronicle recognition. In *Proceedings of the fourteenth International Conference on Reliable Software Technologies, Ada-Europe*, June 2009. *(cité p. 30)*

[BD91] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991. *(cité p. 28)*

[Ber02] Bruno Berstel. Extending the Rete algorithm for event management. In *Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME)*, pages 49–51. IEEE, 2002. *(cité p. 37)*

[Ber09] Olivier Bertrand. *Détection d'activités par un système de reconnaissance de chroniques et application au cas des simulations distribuées HLA*. PhD thesis, Université Paris 13 and ONERA, 2009. *(3 citations pp. 9, 30, et 181)*

[Bor85] Gary C. Borchardt. Event calculus. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 524–527, 1985. *(pas de citation)*

[BSC02] Amine Boufaied, Aaudine Subias, and Michel Combacau. Chronicle modeling by Petri nets for distributed detection of process failures. In *Proceedings of the Second IEEE International Conference on Systems, Man and Cybernetics (SMC'02)*. IEEE Computer Society Press, 2002. *(cité p. 28)*

[BSC04] Amine Boufaied, Audine Subias, and Michel Combacau. Distributed fault detection with delays consideration. In *15th International Workshop on Principles of Diagnosis, Carcassonne*, 2004. *(2 citations pp. 28 et 29)*

[BSC05] Amine Boufaied, Audine Subias, and Michel Combacau. Distributed time constraints verification modelled with time Petri nets. In *Proceedings of the 17th IMACS World Congress, Paris, France*, 2005. *(2 citations pp. 28 et 29)*

[BTF07] Rahul Biswas, Sebastian Thrun, and Kikuo Fujimura. Recognizing activities with multiple cues. In *Human Motion — Understanding, Modeling, Capture and Animation*, pages 255–270. Springer, 2007. *(cité p. 35)*

[CBDO98] Patrice Carle, Philippe Benhamou, François-Xavier Dolbeau, and Manuel Ornato. La reconnaissance d'intentions comme dynamique des organisations. In *6es Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'98)*, 1998. *(cité p. 42)*

[CBP10] Raphael Cuisinier, Michael Brunel, and Stéphanie Prudhomme. Using open source to build comprehensive battlespace simulations. In *Simulation Technology and Training Conference (SimTecT)*, 2010. *(2 citations pp. 12 et 185)*

[CCK11] Patrice Carle, Christine Choppy, and Romain Kervarc. Behaviour recognition using chronicles. In *Proceedings of the 5th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 100–107, 2011. *(21 citations pp. 9, 10, 11, 30, 31, 42, 43, 44, 46, 47, 48, 50, 53, 54, 57, 60, 68, 70, 97, 104, et 115)*

[CCKP12a] Patrice Carle, Christine Choppy, Romain Kervarc, and Ariane Piel. Behavioural analysis for distributed simulations. In *Proceedings of the Nineteenth Asia-Pacific Software Engineering Conference (APSEC)*, 2012. *(3 citations pp. 11, 70, et 194)*

[CCKP12b] Patrice Carle, Christine Choppy, Romain Kervarc, and Ariane Piel. Handling Breakdowns in Unmanned Aircraft Systems. In *Proceedings of the Eighteenth International Symposium on Formal Methods (FM) - Doctoral Symposium*, 2012. *(4 citations pp. 11, 166, 193, et 194)*

[CCKP13a] Patrice Carle, Christine Choppy, Romain Kervarc, and Ariane Piel. A formal coloured Petri net model for hazard detection in large event flows. In *Proceedings of the twentieth Asia-Pacific Software Engineering Conference (APSEC)*, 2013. *(3 citations pp. 11, 134, et 194)*

[CCKP13b] Patrice Carle, Christine Choppy, Romain Kervarc, and Ariane Piel. Safety of unmanned aircraft systems facing multiple breakdowns. In *Proceedings of the First French-Singaporean Workshop on Formal Methods and Applications (FSFMA)*, 2013. *(4 citations pp. 11, 166, 193, et 194)*

[CCQW03] Guy Carrault, Marie-Odile Cordier, Rene Quiniou, and Feng Wang. Temporal abstraction and inductive logic programming for arrhythmia recognition from electrocardiograms. *Artificial Intelligence in Medicine*, 28(3):231–263, 2003. *(cité p. 39)*

[CD97] Luca Chittaro and Michel Dojat. Using a general theory of time and change in patient monitoring: experiment and evaluation. *Computers in Biology and Medicine*, 27(5):435–452, 1997. *(cité p. 39)*

[CD00] Marie-Odile Cordier and Christophe Dousson. Alarm driven monitoring based on chronicles. In *4th SafeProcess*, pages 286–291, 2000. *(2 citations pp. 27 et 39)*

[CGR+07] Marie-Odile Cordier, Xavier Le Guillou, Sophie Robin, Laurence Rozé, and Thierry Vidal. Distributed Chronicles for On-line Diagnosis of Web Services. In *The 18th International Workshop on Principles of Diagnosis (DX-07)*, pages 37–44, 2007. *(cité p. 39)*

[CKAK94] Sharma Chakravarthy, Vidhya Krishnaprasad, Eman Anwar, and Seung-Kyum Kim. Composite events for active databases: Semantics, contexts and detection. In *20th International Conference on Very Large Data Bases (VLDB)*, volume 94, pages 606–617, 1994. *(2 citations pp. 37 et 38)*

[CM94] Sharma Chakravarthy and Deepak Mishra. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994. *(2 citations pp. 37 et 38)*

[CM96] Luca Chittaro and Angelo Montanari. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, 12(3):359–382, 1996. *(cité p. 18)*

[CM10] Gianpaolo Cugola and Alessandro Margara. TESLA: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 50–61. ACM, 2010. *(pas de citation)*

[CM12] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012. *(4 citations pp. 13, 14, 15, et 34)*

[CMM12] Damien Cram, Benoît Mathern, and Alain Mille. A complete chronicle discovery approach: application to activity analysis. *Expert Systems*, 29(4):321–346, 2012. *(cité p. 39)*

[CMZ09] Antonio Cau, Ben Moszkowski, and Hussein Zedan. Interval temporal logic. *URL: http://www.cms.dmu.ac.uk/~cau/itlhomepage/itlhomepage.html*, 2009. *(pas de citation)*

[CP92] Søren Christensen and Laure Petrucci. Towards a modular analysis of coloured Petri nets. *Application and Theory of Petri Nets 1992*, pages 113–133, 1992. *(3 citations pp. 74, 75, et 76)*

[CV98] Silvia Coradeschi and Thierry Vidal. Accounting for temporal evolutions in highly reactive decision-making. In *Proceedings of the Fifth IEEE International Workshop on Temporal Representation and Reasoning*, pages 3–10, 1998. *(cité p. 39)*

[dCRN13] Otávio M. de Carvalho, Eduardo Roloff, and Philippe O. A. Navaux. A survey of the state-of-the-art in event processing. In *Proceedings of the 11th Workshop on Parallel and Distributed Processing (WSPPD)*, 2013. *(cité p. 35)*

[DD99] Christophe Dousson and Thang Vu Du'ong. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 16, pages 620–626, 1999. *(2 citations pp. 24 et 27)*

[DG94] Christophe Dousson and Malik Ghallab. Suivi et reconnaissance de chroniques. *Revue d'intelligence artificielle*, 8(1):29–61, 1994. *(2 citations pp. 26 et 27)*

[DGG93] Christophe Dousson, Paul Gaborit, and Malik Ghallab. Situation recognition: Representation and algorithms. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 166–172, Chambéry, France, 1993. *(4 citations pp. 23, 24, 25, et 42)*

[DGK+00] Patrick Doherty, Gösta Granlund, Krzystof Kuchcinski, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. The WITAS unmanned aerial vehicle project. In *ECAI*, pages 747–755, 2000. *(cité p. 39)*

[DKH09a] Patrick Doherty, Jonas Kvarnström, and Fredrik Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. In *6th International Conference on Recent Advances in Intrusion Detection (RAID'03)*, 2009. *(cité p. 39)*

[DKH09b] Patrick Doherty, Jonas Kvarnström, and Fredrik Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, pages 332–377, 2009. *(cité p. 39)*

[DL09] Pedro Domingos and Daniel Lowd. Markov logic: An interface layer for artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–155, 2009. *(cité p. 19)*

[DLM07] Christophe Dousson and Pierre Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 324–329, 2007. *(5 citations pp. 9, 23, 25, 26, et 211)*

[Doj96] Michel Dojat. Realistic model for temporal reasoning in real-time patient monitoring. *Applied Artificial Intelligence*, 10(2):121–144, 1996. *(cité p. 39)*

[Dou94] Christophe Dousson. *Suivi d'évolution et reconnaissance de chroniques*. PhD thesis, Université Paul Sabatier de Toulouse, 1994. *(2 citations pp. 25 et 211)*

[Dou96] Christophe Dousson. Alarm driven supervision for telecommunication network: II-On-line chronicle recognition. *Annales des Télécommunications*, 9/10(51):493–500, septembre/octobre 1996. *(2 citations pp. 24 et 211)*

[Dou02] Christophe Dousson. Extending and unifying chronicle representation with event counters. In *Proceedings of the Fifteenth Eureopean Conference on Artificial Intelligence (ECAI)*, pages 257–261, 2002. *(4 citations pp. 15, 23, 24, et 211)*

[DRVL95] Luc De Raedt and Wim Van Laer. Inductive constraint logic. In *Algorithmic Learning Theory*, pages 80–94. Springer, 1995. *(cité p. 27)*

[dSBAR08] Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. A survey of first-order probabilistic models. In *Innovations in Bayesian Networks*, pages 289–317. Springer, 2008. *(cité p. 35)*

[EN10] Opher Etzion and Peter Niblett. *Event processing in action*. Manning Publications Co., 2010. *(2 citations pp. 14 et 22)*

[FAR11] Paul Fodor, Darko Anicic, and Sebastian Rudolph. Results on out-of-order event processing. In *Proceedings of the Thirteenth international conference on Practical aspects of declarative languages*, pages 220–234. Springer, 2011. *(2 citations pp. 22 et 23)*

[FCD04] Françoise Fessant, Fabrice Clérot, and Christophe Dousson. Mining of an alarm log to improve the discovery of frequent patterns. In *Advances in Data Mining*, pages 144–152. Springer, 2004. *(cité p. 27)*

[For82] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(1):17–37, 1982. *(cité p. 37)*

[FTR+10] Lajos Jenő Fülöp, Gabriella Tóth, Róbert Rácz, János Pánczél, Tamás Gergely, Árpád Beszédes, and Lóránt Farkas. Survey on complex event processing and predictive analytics. Technical report, Université de Szeged, 2010. *(cité p. 35)*

[GA02] Antony Galton and Juan Carlos Augusto. Two approaches to event definition. In *Database and Expert Systems Applications*, pages 547–556. Springer, 2002. *(2 citations pp. 37 et 38)*

[GD94a] Stella Gatziu and Klaus R. Dittrich. Detecting composite events in active database systems using Petri nets. In *Proceedings of the 4th IEEE International Workshop on Research Issues in Data Engineering*, 1994. *(cité p. 35)*

[GD94b] Stella Gatziu and Klaus R. Dittrich. *Events in an active object-oriented database system*. Springer, 1994. *(cité p. 35)*

[Gha96] Malik Ghallab. On chronicles: Representation, on-line recognition and learning. In *KR*, pages 597–606, 1996. *(cité p. 23)*

[GL94] Malik Ghallab and Hervé Laruelle. Representation and control in IxTeT, a temporal planner. In *AIPS*, volume 1994, pages 61–67, 1994. *(cité p. 26)*

[Hei01] Fredrik Heintz. Chronicle recognition in the WITAS UAV project, a preliminary report. In *Swedish AI Society Workshop (SAIS2001)*, 2001. *(cité p. 39)*

[Hen11] Christophe Henry. "MSM library of boost". `www.boost.org/doc/libs/1_48_0/libs/msm/doc/HTML/index.html`, 2011. *(cité p. 180)*

[HJS91] Peter Huber, Kurt Jensen, and Robert Shapiro. Hierarchies in coloured Petri nets. *Advances in Petri Nets 1990*, pages 313–341, 1991. *(cité p. 74)*

[HN03] Somboon Hongeng and Ramakant Nevatia. Large-scale event detection using semi-hidden Markov models. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, pages 1455–1462, 2003. *(cité p. 35)*

[Hou11] House of Representatives, 12th Congress, 1st Session. FAA reauthorization and reform act of 2011, 11 février 2011. *(cité p. 172)*

[JK09] Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer Verlag Monograph, 2009. *(cité p. 72)*

[KDDR+11] Angelika Kimmig, Bart Demoen, Luc De Raedt, Vitor Santos Costa, and Ricardo Rocha. On the implementation of the probabilistic logic programming language problog. *Theory and Practice of Logic Programming*, 11(2-3):235–262, 2011. *(cité p. 19)*

[KDRR06] Kristian Kersting, Luc De Raedt, and Tapani Raiko. Logical hidden Markov models. *Journal of Artificial Intelligence Research*, 25(1):425–456, 2006. *(cité p. 35)*

[Kha97] Wael Khansa. *Réseaux de Petri p-temporels: contribution à l'étude des systèmes à évènements discrets*. PhD thesis, Université de Savoie, 1997. *(cité p. 28)*

[KM87] Krishna Kumar and Amitabha Mukerjee. Temporal event conceptualization. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1987. *(cité p. 35)*

[KS86] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986. *(cité p. 16)*

[KVNA11] Pekka Kaarela, Mika Varjola, Lucas PJJ Noldus, and Alexander Artikis. Pronto: support for real-time decision making. In *Proceedings of the Fifth ACM International Conference on Distributed Event-Based System*, pages 11–14. ACM, 2011. *(cité p. 39)*

[Lan09] Thibault Lang. IDEAS–T1.1: Architecture de système de drone et scénarios de missions. Technical report, ONERA, 2009. *(3 citations pp. 173, 174, et 212)*

[LGCR+08] Xavier Le Guillou, Marie-Odile Cordier, Sophie Robin, Laurence Rozé, et al. Chronicles for on-line diagnosis of distributed systems. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pages 194–198, 2008. *(cité p. 39)*

[LSA+11] David Luckham, Roy Schulte, Jeff Adkins, Pedro Bizarro, H-Arno Jacobsen, Albert Mavashev, Brenda M Michelson, and Peter Niblett. Event processing glossary-version 2.0. *Event Processing Technical Society*, 2011. *(cité p. 14)*

[Luc02] David Luckham. *The Power of Events*, volume 204. Addison-Wesley Reading, 2002. *(cité p. 14)*

[MCCDB10] Anis M'halla, Étienne Craye, Simon Collart Dutilleul, and Mohamed Benrejeb. Monitoring of a milk manufacturing workshop using chronicle and fault tree approaches. *Studies in Informatics and Control*, 19(4):377–390, 2010. *(cité p. 39)*

[McD82] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive science*, 6(2):101–155, 1982. *(pas de citation)*

[MD03] Benjamin Morin and Hervé Debar. Correlation on intrusion: an application of chronicles. In *6th International Conference on Recent Advances in Intrusion Detection (RAID'03)*, pages 94–112. Springer, 2003. *(cité p. 39)*

[MDR94] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994. *(cité p. 27)*

[MH68] John McCarthy and Patrick Hayes. *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University, 1968. *(pas de citation)*

[MNG+94] Robert Milne, Charles Jarvis Nicol, Malik Ghallab, Louise Travé-Massuyès, Kouamana Bousson, Christophe Dousson, Joseba Quevedo, Jose Aguilar, and Antoni Guasch. TIGER: real-time situation assessment of dynamic systems. *Intelligent Systems Engineering*, 3(3):103–124, 1994. *(cité p. 39)*

[MS99] Rob Miller and Murray Shanahan. The event calculus in classical logic–alternative axiomatizations. *Electronic Transactions on Artificial Intelligence (http://www. etaij. org)*, 4, 1999. *(cité p. 17)*

[MSS96] Masoud Mansouri-Samani and Morris Sloman. A configurable event service for distributed systems. In *Proceedings of the 3rd IEEE International Conference on Configurable Distributed Systems*, pages 210–217, 1996. *(cité p. 36)*

[MSS97] Masoud Mansouri-Samani and Morris Sloman. Gem: A generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96, 1997. *(cité p. 36)*

[OC94a] Manuel Ornato and Patrice Carle. Reconnaissance d'intentions sans reconnaissance de plan. In *2es Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, 1994. *(cité p. 29)*

[OC94b] Manuel Ornato and Patrice Carle. Une alternative à l'abduction pour la reconnaissance de plan. In *Secondes Rencontres des Jeunes Chercheurs en Intelligence Artificielle*, 1994. *(cité p. 29)*

[PAMP12] Jens Pottebaum, Alexander Artikis, Robin Marterer, and George Paliouras. User-oriented evaluation of event-based decision support systems. In *IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 162–169. IEEE, 2012. *(cité p. 19)*

[PB08] Adrian Paschke and Martin Bichler. Knowledge representation concepts for automated sla management. *Decision Support Systems*, 46(1):187–205, 2008. *(cité p. 17)*

[PBC+] Ariane Piel, Jean Bourrely, Patrice Carle, Christine Choppy, and Romain Kervarc. Complex behaviour recognition by online analysis of rich event flows *(to be submitted). (5 citations pp. 11, 42, 166, 193, et 194)*

[PKB07] Adrian Paschke, Alexander Kozlenkov, and Harold Boley. A homogeneous reaction rule language for complex event processing. In *Workshop on Event driven Architecture, Processing and Systems*, 2007. *(cité p. 17)*

[Por05] François Portet. *Pilotage d'algorithmes pour la reconnaissance en ligne d'arythmies cardiaques*. PhD thesis, Université Rennes 1, 2005. *(cité p. 39)*

[PS09] Yannick Pencolé and Audine Subias. A chronicle-based diagnosability approach for discrete timed-event systems: application to web-services. *Journal of Universal Computer Science*, 15(17):3246–3272, 2009. *(cité p. 39)*

[QCC+10] René Quiniou, Lucie Callens, Guy Carrault, Marie-Odile Cordier, Elisa Fromont, Philippe Mabo, and François Portet. Intelligent adaptive monitoring for cardiac surveillance. In *Computational Intelligence in Healthcare 4*, pages 329–346. Springer, 2010. *(cité p. 39)*

[QCCW01] René Quiniou, Marie-Odile Cordier, Guy Carrault, and Feng Wang. Application of ILP to cardiac arrhythmia characterization for chronicle recognition. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming*, pages 220–227. Springer, 2001. *(cité p. 27)*

[RWL+03] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. CPN tools for editing, simulating, and analysing coloured Petri nets. In *Applications and Theory of Petri Nets 2003*, pages 450–462. Springer, 2003. *(cité p. 73)*

[SA11] Nenad Stojanovic and Alexander Artikis. On complex event processing for real-time situational awareness. In *5th International Conference on Rule-Based Reasoning, Programming, and Applications*, pages 114–121. Springer, 2011. *(cité p. 39)*

[SAFP14] Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, and Georgios Paliouras. A probabilistic logic programming event calculus. *Journal of Theory and Practice of Logic Programming (TPLP)*, 2014. *(2 citations pp. 19 et 20)*

[SEC+10] Audine Subias, Ernesto Exposito, Christophe Chassot, Louise Travé-Massuyès, and Khalil Drira. Self-adapting strategies guided by diagnosis and situation assessment in collaborative communicating systems. In *21st International Workshop on Principles of Diagnosis (DX 10)*, pages 329–336, 2010. *(cité p. 39)*

[SPVA11] Anastasios Skarlatidis, Georgios Paliouras, George A Vouros, and Alexander Artikis. Probabilistic event calculus based on Markov logic networks. In *5th International Conference on Rule-Based Modeling and Computing on the Semantic Web*, pages 155–170. Springer, 2011. *(2 citations pp. 9 et 19)*

[TD08] Son D. Tran and Larry S. Davis. Event modeling and recognition using Markov logic networks. In *Computer Vision–ECCV 2008*, pages 610–623. Springer, 2008. *(cité p. 35)*

[UML11] OMG Unified Modeling Language^{TM} (OMG UML), Superstructure, version 2.4.1, 2011. *(cité p. 173)*

[VL10] Mika Varjola and Jobst Loffler. Pronto: Event recognition for public transport. In *Proceedings of the Seventeenth ITS World Congress*, 2010. *(cité p. 39)*

[WBG08] Karen Walzer, Tino Breddin, and Matthias Groch. Relative temporal constraints in the Rete algorithm for complex event detection. In *Proceedings of the Second International Conference on Distributed Event-Based Systems (DEBS)*, pages 147–155. ACM, 2008. *(cité p. 37)*

[WGB08] Karen Walzer, Matthias Groch, and Tino Breddin. Time to the rescue - supporting temporal reasoning in the Rete algorithm for complex event processing. In *Database and Expert Systems Applications*, pages 635–642. Springer, 2008. *(cité p. 37)*

Annexe A

Démonstrations de propriétés du langage des chroniques

A.0.1 Associativité

Propriété 8. *La disjonction, la conjonction, la séquence, meets et equals sont associatifs.*

Démonstration. Soit $C_1, C_2, C_3 \in X(\mathfrak{N})$. Soit φ un flot d'évènements et d un réel.

$$\begin{aligned}
\mathfrak{F}_{C_1||(C_2||C_3)}(\varphi, d) &= \mathfrak{F}_{C_1}(\varphi, d) \cup \mathfrak{F}_{C_2||C_3}(\varphi, d) \\
&= \mathfrak{F}_{C_1}(\varphi, d) \cup (\mathfrak{F}_{C_2}(\varphi, d) \cup \mathfrak{F}_{C_3}(\varphi, d)) \\
&= (\mathfrak{F}_{C_1}(\varphi, d) \cup \mathfrak{F}_{C_2}(\varphi, d)) \cup \mathfrak{F}_{C_3}(\varphi, d) \\
&= \mathfrak{F}_{(C_1||C_2)||C_3}(\varphi, d)
\end{aligned}$$

L'associativité de la disjonction découle donc de l'associativité de l'union ensembliste.

$$\begin{aligned}
\mathfrak{F}_{C_1\&(C_2\&C_3)}(\varphi, d) &= \{\mathcal{F}(r_1) \cup \mathcal{F}(r_2) : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_2\&C_3}(\varphi, d)\} \\
&= \{\mathcal{F}(r_1) \cup (\mathcal{F}(r_{2a}) \cup \mathcal{F}(r_{2b})) : r_1 \in R_{C_1}(\varphi, d) \\
&\qquad \wedge r_{2a} \in R_{C_2}(\varphi, d) \wedge r_{2b} \in R_{C_3}(\varphi, d)\} \\
&= \{(\mathcal{F}(r_1) \cup \mathcal{F}(r_{2a})) \cup \mathcal{F}(r_{2b}) : r_1 \in R_{C_1}(\varphi, d) \wedge r_{2a} \in R_{C_2}(\varphi, d) \\
&\qquad \wedge r_{2b} \in R_{C_3}(\varphi, d)\} \\
&= \{\mathcal{F}(r) \cup \mathcal{F}(r_{2b}) : r \in R_{C_1\&C_2}(\varphi, d) \wedge r_{2b} \in R_{C_3}(\varphi, d)\} \\
&= \mathfrak{F}_{(C_1\&C_2)\&C_3}(\varphi, d)
\end{aligned}$$

L'associativité de la conjonction découle donc aussi de l'associativité de l'union ensembliste.

$$
\begin{aligned}
\mathfrak{F}_{C_1\,(C_2\,C_3)}(\varphi,d) &= \{\mathcal{F}(r_1)\cup\mathcal{F}(r_2) : r_1\in R_{C_1}(\varphi,d)\wedge r_2\in R_{C_2\,C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\max}(r_1)<T_{\min}(r_2)\}\\
&= \{\mathcal{F}(r_1)\cup(\mathcal{F}(r_{2a})\cup\mathcal{F}(r_{2b})) : r_1\in R_{C_1}(\varphi,d)\wedge r_{2a}\in R_{C_2}(\varphi,d)\\
&\qquad\qquad \wedge r_{2b}\in R_{C_3}(\varphi,d)\wedge T_{\max}(r_1)<T_{\min}(r_{2a})\\
&\qquad\qquad \wedge T_{\max}(r_{2a})<T_{\min}(r_{2b})\}\\
&= \{(\mathcal{F}(r_1)\cup\mathcal{F}(r_{2a}))\cup\mathcal{F}(r_{2b}) : r_1\in R_{C_1}(\varphi,d)\wedge r_{2a}\in R_{C_2}(\varphi,d)\\
&\qquad\qquad \wedge r_{2b}\in R_{C_3}(\varphi,d)\wedge T_{\max}(r_1)<T_{\min}(r_{2a})\\
&\qquad\qquad \wedge T_{\max}(r_{2a})<T_{\min}(r_{2b})\}\\
&= \{\mathcal{F}(r)\cup\mathcal{F}(r_{2b}) : r\in R_{C_1\,C_2}(\varphi,d)\wedge r_{2b}\in R_{C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\max}(r)<T_{\min}(r_{2b})\}\\
&= \mathfrak{F}_{(C_1\,C_2)\,C_3}(\varphi,d)
\end{aligned}
$$

L'associativité de la séquence découle donc de l'associativité de l'union ensembliste.

$$
\begin{aligned}
\mathfrak{F}_{C_1\text{ meets }(C_2\text{ meets }C_3)}(\varphi,d) &= \{\mathcal{F}(r_1)\cup\mathcal{F}(r'_2) : r_1\in R_{C_1}(\varphi,d)\wedge r'_2\in R_{C_2\text{ meets }C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\max}(r_1)=T_{\min}(r'_2)\}\\
&= \{\mathcal{F}(r_1)\cup\mathcal{F}(r_2)\cup\mathcal{F}(r_3) : r_1\in R_{C_1}(\varphi,d)\wedge r_2\in R_{C_2}(\varphi,d)\\
&\qquad\qquad \wedge r_3\in R_{C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\max}(r_1)=T_{\min}(r_2\cup r_3)\\
&\qquad\qquad \wedge T_{\max}(r_2)=T_{\min}(r_3)\}\\
&= \{\mathcal{F}(r_1)\cup\mathcal{F}(r_2)\cup\mathcal{F}(r_3) : r_1\in R_{C_1}(\varphi,d)\wedge r_2\in R_{C_2}(\varphi,d)\\
&\qquad\qquad \wedge r_3\in R_{C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\max}(r_1)=T_{\min}(r_2)\wedge T_{\max}(r_2)=T_{\min}(r_3)\}\\
&= \{\mathcal{F}(r'_1)\cup\mathcal{F}(r_3) : r'_1\in R_{C_1\text{ meets }C_2}(\varphi,d)\wedge r_3\in R_{C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\max}(r'_1)=T_{\min}(r_3)\}\\
&= \mathfrak{F}_{(C_1\text{ meets }C_2)\text{ meets }C_3}(\varphi,d)
\end{aligned}
$$

« meets » est donc associatif.

$$
\begin{aligned}
\mathfrak{F}_{C_1\text{ equals }(C_2\text{ equals }C_3)}(\varphi,d) &= \{\mathcal{F}(r_1)\cup\mathcal{F}(r'_2) : r_1\in R_{C_1}(\varphi,d)\wedge r'_2\in R_{C_2\text{ equals }C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\min}(r_1)=T_{\min}(r'_2)\\
&\qquad\qquad \wedge T_{\max}(r_1)=T_{\max}(r'_2)\}\\
&= \{\mathcal{F}(r_1)\cup\mathcal{F}(r_2)\cup\mathcal{F}(r_3) : r_1\in R_{C_1}(\varphi,d)\wedge r_2\in R_{C_2}(\varphi,d)\\
&\qquad\qquad \wedge r_3\in R_{C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\min}(r_1)=T_{\min}(r_2\cup r_3)\\
&\qquad\qquad \wedge T_{\max}(r_1)=T_{\max}(r_1\cup r_3)\\
&\qquad\qquad \wedge T_{\min}(r_2)=T_{\min}(r_3)\wedge T_{\max}(r_2)=T_{\max}(r_3)\}\\
&= \{\mathcal{F}(r_1)\cup\mathcal{F}(r_2)\cup\mathcal{F}(r_3) : r_1\in R_{C_1}(\varphi,d)\wedge r_2\in R_{C_2}(\varphi,d)\\
&\qquad\qquad \wedge r_3\in R_{C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\min}(r_1)=T_{\min}(r_2)=T_{\min}(r_3)\\
&\qquad\qquad \wedge T_{\max}(r_1)=T_{\max}(r_2)=T_{\max}(r_3)\}\\
&= \{\mathcal{F}(r'_1)\cup\mathcal{F}(r_3) : r'_1\in R_{C_1\text{ equals }C_2}(\varphi,d)\wedge r_3\in R_{C_3}(\varphi,d)\\
&\qquad\qquad \wedge T_{\min}(r'_1)=T_{\min}(r_3)\wedge T_{\max}(r'_1)=T_{\max}(r_3)\}\\
&= \mathfrak{F}_{(C_1\text{ equals }C_2)\text{ equals }C_3}(\varphi,d)
\end{aligned}
$$

« equals » est donc associatif. □

A.0.2 Commutativité

Propriété 9. *La disjonction, la conjonction et equals sont commutatifs.*

Démonstration. Soit $C_1, C_2 \in X(\mathfrak{N})$. Soit φ un flot d'évènements et d un réel.

$$\begin{aligned}
\mathfrak{F}_{C_1||C_2}(\varphi,d) &= \mathfrak{F}_{C_1}(\varphi,d) \cup \mathfrak{F}_{C_2}(\varphi,d) \\
&= \mathfrak{F}_{C_2}(\varphi,d) \cup \mathfrak{F}_{C_1}(\varphi,d) \\
&= \mathfrak{F}_{C_2||C_1}(\varphi,d) \\
\mathfrak{F}_{C_1\&C_2}(\varphi,d) &= \{\mathcal{F}(r_1) \cup \mathcal{F}(r_2) : r_1 \in R_{C_1}(\varphi,d) \wedge r_2 \in R_{C_2}(\varphi,d)\} \\
&= \{\mathcal{F}(r_2) \cup \mathcal{F}(r_1) : r_1 \in R_{C_1}(\varphi,d) \wedge r_2 \in R_{C_2}(\varphi,d)\} \\
&= \mathfrak{F}_{C_2\&C_1}(\varphi,d)
\end{aligned}$$

La commutativité de la disjonction et de la conjonction découle donc de la commutativité de l'union ensembliste.

$$\begin{aligned}
\mathfrak{F}_{C_1 \text{ equals } C_2}(\varphi,d) &= \{\mathcal{F}(r_1) \cup \mathcal{F}(r_2) : r_1 \in R_{C_1}(\varphi,d) \wedge r_2 \in R_{C_2}(\varphi,d) \\
&\qquad \wedge T_{\min}(r_1) = T_{\min}(r_2) \wedge T_{\max}(r_1) = T_{\max}(r_2)\} \\
&= \{\mathcal{F}(r_2) \cup \mathcal{F}(r_1) : r_2 \in R_{C_2}(\varphi,d) \wedge r_1 \in R_{C_1}(\varphi,d) \\
&\qquad \wedge T_{\min}(r_2) = T_{\min}(r_1) \wedge T_{\max}(r_2) = T_{\max}(r_1)\}\} \\
&= \mathfrak{F}_{C_2 \text{ equals } C_1}(\varphi,d)
\end{aligned}$$

□

A.0.3 Distributivité

Propriété 10. *Tous les opérateurs sont distributifs sur la disjonction.*

Démonstration. Soit $C_0, C_1, C_2 \in X(\mathfrak{N})$. Soit φ un flot d'évènements et d un réel.

— La conjonction est distributive sur la disjonction :

$$\begin{aligned}
\mathfrak{F}_{C_0\&(C_1||C_2)}(\varphi,d) &= \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_1||C_2}(\varphi,d)\} \\
&= \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_1}(\varphi,d) \cup R_{C_2}(\varphi,d)\} \\
&= \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_1}(\varphi,d)\} \\
&\quad \cup \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_2}(\varphi,d)\} \\
&= \mathfrak{F}_{C_0\&C_1}(\varphi,d) \cup \mathfrak{F}_{C_0\&C_2}(\varphi,d) \\
&= \mathfrak{F}_{(C_0\&C_1)||(C_0\&C_2)}(\varphi,d)
\end{aligned}$$

— La séquence est distributive sur la disjonction :

$$\begin{aligned}
\mathfrak{F}_{C_0\,(C_1||C_2)}(\varphi,d) &= \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_1||C_2}(\varphi,d) \\
&\qquad \wedge T_{\max}(r_0) < T_{\min}(r_1)\} \\
&= \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_1}(\varphi,d) \cup R_{C_2}(\varphi,d) \\
&\qquad \wedge T_{\max}(r_0) < T_{\min}(r_1)\} \\
&= \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_1}(\varphi,d) \\
&\qquad \wedge T_{\max}(r_0) < T_{\min}(r_1)\} \\
&\quad \cup \{\mathcal{F}(r_0) \cup \mathcal{F}(r_1) : r_0 \in R_{C_0}(\varphi,d) \wedge r_1 \in R_{C_2}(\varphi,d) \\
&\qquad \wedge T_{\max}(r_0) < T_{\min}(r_1)\} \\
&= \mathfrak{F}_{C_0\,C_1}(\varphi,d) \cup \mathfrak{F}_{C_0\,C_2}(\varphi,d) \\
&= \mathfrak{F}_{(C_0\,C_1)||(C_0\,C_2)}(\varphi,d)
\end{aligned}$$

— Notons $\circledast$ un opérateur quelconque des opérateurs meets, overlaps, starts, during, finishes et equals.
Remarquons que l'on a alors :
$\mathfrak{F}_{C_1 \circledast C_2}(\varphi, d) = \{\mathcal{F}(r_1) \cup \mathcal{F}(r_2) : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_2}(\varphi, d) \wedge P_{\circledast}(r_1, r_2, \varphi, d)\}$, où $P_{\circledast}$ est une propriété dépendant de $\circledast$ et exprimant des contraintes temporelles.
Alors, si $C_2 = C_3 \,||\, C_4$:

$$\begin{aligned}
\mathfrak{F}_{C_1 \circledast (C_3 || C_4)}(\varphi, d) &= \{\mathcal{F}(r_1) \cup \mathcal{F}(r_2) : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_3}(\varphi, d) \cup R_{C_4}(\varphi, d) \\
&\qquad \wedge P_{\circledast}(r_1, r_2, \varphi, d)\} \\
&= \{\mathcal{F}(r_1) \cup \mathcal{F}(r_2) : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_3}(\varphi, d) \\
&\qquad \wedge P_{\circledast}(r_1, r_2, \varphi, d)\} \\
&\quad \cup \{\mathcal{F}(r_1) \cup \mathcal{F}(r_2) : r_1 \in R_{C_1}(\varphi, d) \wedge r_2 \in R_{C_4}(\varphi, d) \\
&\qquad \wedge P_{\circledast}(r_1, r_2, \varphi, d)\} \\
&= \mathfrak{F}_{C_1 \circledast C_3}(\varphi, d) \cup \mathfrak{F}_{C_1 \circledast C_4}(\varphi, d) \\
&= \mathfrak{F}_{(C_1 \circledast C_3) || (C_1 \circledast C_4)}(\varphi, d)
\end{aligned}$$

$\circledast$ est donc distributif sur la disjonction.

□

Table des figures et des tableaux

Table des symboles

Acronymes

ATC	Air Traffic Control ... 172
BLADE	Battle Lab for Aerospace and Defence Experimentations ... 185
CEC	Cached Event Calculus ... 18
CEP	Complex Event Processing ... 193
CRL	Chronicle Recognition Library ... 166
CRS	Chronicle Recognition System ... 165
CRS/Onera	Chronicle Recognition System/Onera ... 165
DSMS	Système de Gestion de Flux de Données – Data Stream Management System ... 16
EC	Event Calculus ... 211
ECA	Event-Condition-Action ... 37
EDBC	Event-Driven Backward Chaining ... 22
EP-IRM	Event Processing for Intelligent Ressource Management ... 19
ETALIS	Event-driven Transaction Logic Inference System ... 211
FACE	Frequency Analyser for Chronicle Extraction ... 27
GEM	Generalised Event Monitoring ... 36
IDEAS	Insertion des Drones dans l'Espace Aérien et Sécurité ... 171
ICL	Inductive Constraint Logic ... 27
IFP	Information Flow Processing ... 13
ILP	Inductive Logic Programming ... 27
MCPN	Modular Coloured Petri Net ... 74
RPS	Remote Pilot Station ... 172
RTEC	Event Calculus for Run-Time reasoning ... 19
SAMOS	Swiss Active Mechanism based Object-oriented database Systems ... 35
S-PN	SAMOS Petri nets ... 35
TC	Telecommand ... 172
TM	Telemetry ... 172
UA	Unmanned Aircraft ... 171
UAS	Unmanned Aircraft System ... 171
UML	Unified Modeling Language ... 173

Résumé en français L'analyse de flux d'évènements pour reconnaître des comportements complexes prédéfinis (Complex Event Processing – CEP) permet d'interpréter et de réagir à des quantités importantes de données ne pouvant être appréhendées telles quelles. Dans cette thèse, nous fournissons le cadre théorique général d'un CEP en adoptant une approche purement formelle qui assure une possibilité de vérification et d'analyse du processus de reconnaissance. Nous définissons un langage, le langage des chroniques, permettant de décrire les comportements complexes à reconnaître. Nous formalisons la notion de reconnaissance de chronique à l'aide d'une sémantique ensembliste fondée sur une représentation arborescente des reconnaissances. Dans une visée applicative, nous développons ensuite deux modèles du processus de reconnaissance. Le premier est réalisé avec le formalisme des réseaux de Petri colorés et permet de valider les principes de reconnaissance en faisant notamment ressortir les problèmes de concurrence et de modularité. Le second implémente directement le formalisme mathématique sous la forme d'une bibliothèque C++ appelée Chronicle Recognition Library (CRL) et disponible en open source. Nous tirons parti de cette implémentation pour traiter deux cas d'applications liés à l'insertion des drones dans l'espace aérien. La première vise à surveiller la cohérence d'un système de drones insérés dans le trafic aérien, en cas de pannes de liens de communication. Notre application permet d'une part de vérifier la cohérence des procédures actuellement mises en place en cas de pannes ; et d'autre part de compléter ces procédures par des alarmes dans les situations inévitables causées par des erreurs humaines. La seconde application surveille le bon respect des procédures de sécurité d'un drone partant en mission et traversant diverses zones, contrôlées ou non, de l'espace aérien.

Titre en anglais Online Event Flow Processing for Complex Behaviour Recognition

Résumé en anglais Recognising complex predefined behaviours by the analysis of event flows (Complex Event Processing – CEP) allows to interpret and react to large quantities of data which one would not be able to apprehend alone. In this Ph.D. thesis, we provide a general theoretical framework for CEP through a purely formal approach ensuring the possibility to check and analyse the recognition process. We define a language, the chronicle language, allowing the description of the complex behaviours to be recognised. We formalise the notion of chronicle recognition through a set semantics based on an arborescent representation of recognitions. In order to use this framework, we then develop two models of the recognition process. The first relies on coloured Petri nets and allows the validation of recognition principles including concurrency and modularity issues. The second model directly implements the mathematical formalism in a C++ library, Chronicle Recognition Library (CRL), which is available in open source. We use this implementation to fulfil two applications linked to the insertion of unmanned aircrafts in controlled airspace. The first application oversees the consistency of an unmanned aircraft system inside air traffic in case of communication link breakdowns. This application allows, on the one hand, to check the consistency of the procedures currently followed in case of failures ; and, on the other hand, to complete these procedures with alarms in case of unavoidable situations caused by human errors. The second application oversees that the security procedures of an unmanned aircraft flying through controlled or uncontrolled airspace are correctly followed.

Discipline Informatique

Mots-clés traitement d'évènements complexes ; reconnaissance de comportements ; logique temporelle ; modèle modulaire en réseaux de Petri colorés ; bibliothèque de reconnaissance de chroniques ; application aérospatiale ; drones.

Onera – The French Aerospace Lab
Département de Conception et évaluation de Performance des Systèmes
91123 Palaiseau, France
www.onera.fr