GÉNÉRATION DE GRAPHES D'ACCESSIBILITÉ À PARTIR DE STRUCTURES RÉPLICABLES

par

Michel Embe Jiague

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 16 décembre 2008

Your file Votre référence
ISBN: 978-0-494-49496-7
Our file Notre référence
ISBN: 978-0-494-49496-7

Le 10 février 2009

le jury a accepté le mémoire de M. Michel Embe Jiague dans sa version finale.

Membres du jury

M. Richard St-Denis
Directeur
Département d'informatique

M. Benoît Fraikin
Membre

M. Marc Frappier
Président-rapporteur
Département d'informatique

À la mémoire de Mami Meni, ma grand-mère bien aimée.

Sommaire

Ce mémoire explore une méthode symbolique pour le calcul hors ligne de contrôleurs non bloquants basée principalement sur un algorithme de génération de graphes d'accessibilité. Dans le but de réduire l'explosion de l'espace d'états, l'algorithme considère une classe particulière de systèmes définis à partir de structures réplicables. De telles structures résultent de la modélisation de systèmes à événements discrets paramétrés et de cas respectivement issus du domaine de la théorie du contrôle et du domaine des *workflows*. La principale caractéristique de ce nouvel algorithme est qu'il travaille sur des expressions symboliques à la place de valeurs numériques lors de la génération du graphe d'accessibilité.

Remerciements

Je remercie mon directeur de recherche Richard St-Denis pour son implication dans ce travail de maîtrise.

Je remercie Marc Frappier et Benoît Fraikin dont les suggestions et remarques ont permis l'aboutissement de ce mémoire.

Je remercie aussi mes camarades du GRIL, en particulier Jérémy Milhau et Pierre Konopacki, pour leur soutien continu.

Je remercie également mes parents, en particulier M. et Mme Jiague, mes frères et sœurs pour leur soutien affectif et inconditionnel.

Enfin, que tous celles et ceux qui ont, de près ou de loin, contribué à l'aboutissement de ce travail trouvent ici l'expression sincère de ma gratitude.

Abréviations

BPEL Business Process Execution Language

CTR Concurrent Transaction Logic

DBMS Database Management System

RP Réseau de Petri

SED Système à événements discrets

SEDP Système à événements discrets paramétré

SFBC State Feedback Control

TL Temporal Logic

UIMS User Interface Management System

WfMC Workflow Management Coalition

WFMS Workflow Management System

WS-BPEL (BPEL) Web Services Business Process Execution Language

WSDL Web Service Definition Language

XML Extensible Markup Language

YAWL Yet Another Workflow Language

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction

Contexte

Les *workflows* représentent un vaste domaine d'application de l'informatique à la gestion des processus *métier* ou d'*affaires* des entreprises ou des organisations. Comme le souligne van der Aalst [1], ceux-ci sont l'évolution naturelle des systèmes d'information. La figure 1 illustre cette évolution d'une approche monolithique, où les systèmes d'information sont implémentés directement dans des applications destinées au support de tâches individuelles, vers une approche où il y a une grande séparation des *responsabilités* entre des systèmes spécialisés comme des systèmes de gestion de bases de données (DBMS–Database Management System), des systèmes de gestion des interfaces utilisateurs (UIMS–User Interface Management System) ou même des systèmes de gestion de *workflows* (WFMS–Workflow Management System).

Les *workflows* proviennent du besoin de *contrôler*, *surveiller*, *optimiser* et *supporter* les processus d'affaires. Aussi, comme avec les systèmes de gestion de bases de données, il est important de séparer la représentation de l'exécution des *workflows*. De nombreuses

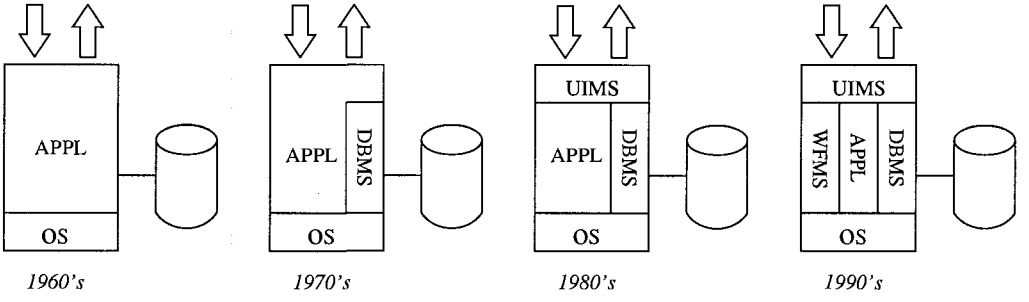


Figure 1 – Évolution des systèmes d'information [1]

études ont été menées dans cette voie et on dispose de nos jours de plusieurs outils, les uns plus ou moins formels que les autres, pour la modélisation et l'analyse des *workflows*. Parmi ceux-ci bien sûr, on distingue les réseaux de Petri. Ces derniers sont déjà des outils exploités dans d'autres domaines, fournissant à la fois une notation graphique puissante et une solide base théorique qui se prête bien à divers types d'analyse.

Problématique

Dans les organisations, les *workflows* indiquent des procédures de traitement de l'information. Généralement, un *workflow* traite de nombreuses *instances* de même type d'information. Ces instances représentent des *cas* et, dans cette étude, leur nombre est signifié par un paramètre dont la valeur n'est pas connue à priori. Bien que l'on sache comment modéliser des *workflows* paramétrés, par exemple à l'aide de réseaux de Petri, il est plus difficile d'analyser leurs propriétés. Quant au contrôle de *workflows*, qu'ils soient ou non paramétrés, aucune étude sérieuse n'a été entreprise dans le contexte de la théorie du contrôle des systèmes à événements discrets (SED). Généralement, le contrôle se limite à l'allocation des ressources ou à l'ordonnancement de tâches et il est réalisé d'une manière ad hoc en utilisant les techniques développées dans ces deux champs d'étude [31, 25, 6, 7] sans lien avec une théorie du contrôle. Dans le contexte de la théorie du contrôle des SED, un *workflow* peut être considéré comme un SED ayant un comportement libre soumis à des contraintes désirables. Son comportement peut être restreint à l'aide d'un contrôleur qui tient compte des contraintes à satisfaire. Le contrôle qu'exerce un contrôleur sur la conduite d'un *workflow* peut s'effectuer à partir de son état courant, mais le calcul d'un tel contrôleur nécessite l'ensemble des états possibles du *workflow*. Si le *workflow* est représenté par un réseau de Petri, alors ces états sont obtenus par la construction d'un graphe d'accessibilité. Dans les situations usuelles, le nombre de cas dans le *workflow* est *arbitrairement* grand et il y a alors explosion combinatoire du nombre d'états. Une solution potentielle pour contourner ce problème consiste à manipuler des expressions symboliques à la place de valeurs numériques lors de la génération du graphe d'accessibilité. Ainsi, considérant la méconnaissance d'une théorie du contrôle par les experts du domaine des *workflows* et des problèmes inhérents dus à l'explosion combinatoire du nombre d'états lors du calcul de contrôleurs, le calcul systématique d'un contrôleur non bloquant pour un *workflow* est un problème peu

abordé dans le littérature [9].

Lorsque le nombre de cas est connu, c'est-à-dire qu'il est donné par une valeur fixée à l'avance, il suffit de substituer le paramètre (représentant le nombre de cas) par cette valeur et ensuite utiliser les algorithmes usuels de la théorie du contrôle des SED [8, 35, 17, 15]. Mais le problème d'explosion du nombre d'états subsiste.

Pour contourner ce problème, une solution intéressante a été proposée par Bherer et al. [4]. Elle consiste à considérer explicitement le paramètre et à manipuler des expressions symboliques à la place de valeurs numériques lors de la génération du graphe d'accessibilité. Toutefois, selon la nature des contraintes à satisfaire, il en résulte des systèmes d'inéquations qui sont en fait des *problèmes de programmation linéaires en nombre entier*. Ce problème est connu pour être un problème de la classe NP-complet [13, 12]. Un traitement algébrique fait par un programme est aussi envisageable, mais le problème général est indécidable. Dans un cas comme dans l'autre, il faut considérer des classes particulières de contraintes pour espérer obtenir des algorithmes satisfaisants. Par exemple, Li et Wonham [19] ont proposé une méthode pour le calcul *hors ligne* de contrôleurs sous une *forme fermée* pour une famille particulière de problèmes de programmation linéaire en nombres entiers. Bherer et al. [5] ont developpé une méthode de synthèse de contrôleurs *en ligne* pour des systèmes paramétrés construits à partir d'une structure instanciable. Selon notre connaissance, il s'agit là des seuls travaux qui tiennent compte de paramètres lors du calcul de contrôleurs.

Objectifs

Le but de ce mémoire est de reconsidérer l'algorithme de génération de graphes d'accessibilité pour des graphes d'états proposé par Bherer et al. [4] afin de pallier à ses lacunes. Premièrement, l'algorithme est décrit sous forme abstraite. Aussi plusieurs de ses opérations sont présentées de manière textuelle. Deuxièmement, son bon fonctionnement n'a pas été démontré par une preuve d'exactitude. Troisièmement, il n'a pas été programmé afin d'exposer ses forces ou ses faiblesses. Quatrièmement, il n'a pas été intégré à un algorithme

de synthèse de contrôleurs, car seul le cas où tous les événements sont contrôlables a été retenu.

Suite à ces constats, les objectifs de ce mémoire sont les suivants :

1. Obtenir un problème décidable. Des hypothèses doivent être posées tant sur le type de contraintes à satisfaire que sur le nombre et le type de paramètres à considérer.
2. Augmenter la rigueur de la preuve de correction de l'algorithme. Sans nécessairement viser à obtenir une preuve formelle d'exactitude, le nouvel algorithme doit être conçu selon une approche systématique.
3. Traduire l'algorithme sous une forme concrète. Des structures de données appropriées doivent être choisies et l'algorithme doit être traduit dans un langage de programmation de façon à obtenir un prototype exécutable qui illustre son fonctionnement.
4. Intégrer l'algorithme dans un algorithme de synthèse de contrôleurs. Contrairement à un algorithme de synthèse basé sur le calcul d'un point fixe, l'algorithme de génération d'un graphe d'accessibilité se combine parfaitement avec un algorithme de synthèse basé sur une exploration de l'espace d'états.

Méthodologie

Basé sur la théorie du contrôle des SED, nous restreignons le problème général de contrôle à celui pour lequel les contraintes sont spécifiées sur la forme d'un ensemble de mauvais états. Ainsi, un contrôle basé sur les états convient parfaitement à ce cas particulier. Nous supposons aussi que les états des *workflows* sont totalement observables. Quant aux hypothèses faites sur les paramètres, elles sont données sous forme d'invariants.

Nous proposons de développer l'algorithme en utilisant une méthode basée sur des invariants formulés à partir de différents ensembles. Ces invariants sont directement traduisibles sous la forme d'opérations sur des chaînes de bits. Ces dernières seront emmagasinées dans des structures de données appropriées afin de rendre l'implémentation des opérations élémentaires efficaces. La définition d'invariants constitue une étape cruciale dans la concep-

tion de l'algorithme. Ils sont aussi utiles pour démontrer l'exactitude d'un algorithme et pour détecter des situations inattendues lors de la mise au point d'un programme. Dans le cas qui nous intéresse, les invariants introduits dans le code permettent de valider chaque nouveau nœud généré dans le graphe d'accessibilité.

L'implémentation de l'algorithme est réalisée selon une approche orientée objet de manière à réutiliser des classes existantes, en particulier celles contenues dans un paquetage pour la manipulation de graphes [1].

Un graphe d'accessibilité généré par l'algorithme est essentiellement un graphe de transition d'états qui modélise le comportement libre d'un système (ou d'un *workflow*). Avec l'ensemble des états interdits, ces éléments constituent les principales données d'entrée de tout algorithme de synthèses de contrôleurs. Toutefois, il existe des algorithmes, comme celui de Barbeau et al. [2], qui ne nécessite pas un modèle explicite du comportement libre, mais une procédure pour le construire à partir de structures élémentaires qui décrivent les comportements des composants du système. De tels algorithmes permettent l'identification de situations indésirables (mauvais états ou blocages) pendant la construction du graphe d'accessibilité. Une procédure de *retour arrière* permet alors de construire la fonction de contrôle de type SFBC (State Feedback Control) de manière incrémentale tout en élaguant les mauvais chemins du graphe d'accessibilité. Nous retenons donc l'algorithme de Barbeau et al. pour montrer comment obtenir un algorithme plus puissant, c'est-à-dire un algorithme qui calcule un contrôleur sous une forme fermée pour un *workflow* paramétré.

Organisation du mémoire

Le reste du mémoire est organisé comme suit. Le premier chapitre introduit les concepts nécessaires à la compréhension du mémoire, en particulier certains propres aux réseaux de Petri, à la théorie du contrôle des SED et aux *workflows*. Le deuxième chapitre présente tout d'abord l'algorithme de génération d'un graphe d'accessibilité sous une forme abstraite, puis introduit les invariants nécessaires pour une conception plus détaillée de ses opérations élémentaires et enfin, quelques preuves utiles pour démontrer son exactitude. Le troisième

1. `http://jgrapht.sourceforge.net/`

chapitre détaille divers aspects de la codification et de l'expérimentation de l'algorithme de génération d'un graphe d'accessibilité. Le quatrième chapitre montre comment utiliser l'algorithme de génération d'un graphe d'accessibilité dans une procédure de synthèse de contrôleurs. Une conclusion clot ce mémoire.

Chapitre 1

Notions de base

1.1 Réseaux de Petri

Les réseaux de Petri ont été inventés par Carl Adam Petri. Ce dernier les a formalisés en 1962 dans sa thèse de doctorat [24]. Murata en fait la revue dans [21], et avec He, ils présentent leurs évolutions dans [14].

1.1.1 Définition

Définition 1.1. Un réseau de Petri (RP) est un triplet (P, T, ε), où :

- P est un ensemble fini de *places* ;
- T est un ensemble fini de *transitions* ;
- $\varepsilon \subseteq (P \times T) \cup (T \times P)$ est la *relation d'incidence* qui représente l'ensemble des *arcs*, des places aux transitions et des transitions aux places.

Dans cette définition, il faut noter que :

- $P \cap T = \emptyset$;
- $P \cup T \neq \emptyset$.

Un RP est donc un graphe biparti orienté alterné dont l'ensemble de nœuds est partitionné en P et T, c'est-à-dire qu'entre les types de nœuds, il y a une alternance dans le graphe.

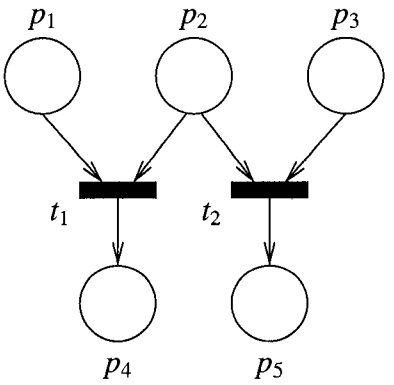


Figure 1.1 – Exemple d'un RP

Une représentation graphique est aussi employée pour noter les RP. Dans cette représentation les cercles dénotent les places, les traits épais (ou rectangles pleins) dénotent les transitions et les flèches représentent les arcs.

Exemple Soit (P, T, ε) un RP où

- $P = \{p_1, p_2, p_3, p_4, p_5\}$;
- $T = \{t_1, t_2\}$;
- $\varepsilon = \{(p_1, t_1), (p_2, t_1), (p_2, t_2), (p_3, t_2), (t_1, p_4), (t_2, p_5)\}$.

La figure 1.1 montre la représentation graphique de celui-ci.

Dans la suite, on considère un RP donné sous la forme d'un triplet (P, T, ε).

Définition 1.2. On appelle *places d'entrée* d'une transition $t \in T$ (*preset de t*) l'ensemble

$$\{p \mid (p, t) \in \varepsilon\},$$

noté ${}^{(p)}t$.

Définition 1.3. On appelle *places de sortie* d'une transition $t \in T$ (*postset de t*) l'ensemble

$$\{p \mid (t, p) \in \varepsilon\},$$

noté $t^{(p)}$.

Définition 1.4. On appelle *transitions d'entrée* d'une place $p \in P$ (*preset de p*) l'ensemble

$$\{t \mid (t, p) \in \varepsilon\},$$

noté ${}^{(t)}p$.

Définition 1.5. On appelle *transitions de sortie* d'une place $p \in P$ (*postset de p*) l'ensemble

$$\{t \mid (p, t) \in \varepsilon\},$$

noté $p^{(t)}$.

1.1.2 Marquage

Définition 1.6. Un *marquage* est une fonction $m : P \to \mathbb{N}$ qui associe à chaque place un nombre de *jetons*.

Les *jetons* constituent l'élément dynamique d'un RP et sont représentés graphiquement par des *points noirs*. Pour le marquage m, le terme $m(p)$ désigne le nombre de jetons de la place $p \in P$. L'ensemble $\mathcal{M} = \mathbb{N}^{|P|}$ est l'ensemble de tous les marquages du RP.

Définition 1.7. Le couple (N, m_0), où N est un RP et $m_0 \in \mathcal{M}$ est un *marquage initial*, est un *réseau de Petri marqué*.

Exemple Considérons le réseau de l'exemple précédent. Augmenté du marquage initial

$$\begin{aligned} m_0 : P &\to \mathbb{N} \\ p_1 &\mapsto 1 \\ p_2 &\mapsto 1 \\ p_3 &\mapsto 1 \\ p_4 &\mapsto 0 \\ p_5 &\mapsto 0 \end{aligned}$$

Sa représentation graphique est donnée à la figure 1.2.

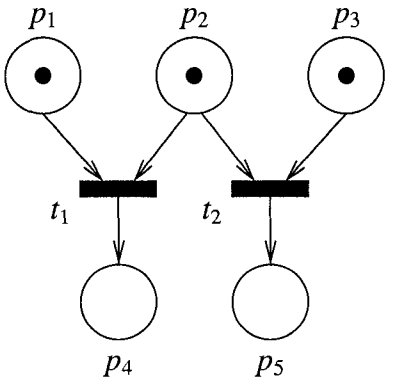


Figure 1.2 – Exemple de marquage dans un RP

Définition 1.8. Un ensemble de transitions $\tau \subseteq T$ est *actif* par un marquage m d'un RP si $m(p) \geq |p^{(t)} \cap \tau|$ pour tout $p \in P$.

Un ensemble de transitions est donc actif si le nombre de jetons à chaque place du RP est suffisant pour le franchissement éventuel des transitions de l'ensemble.

Soit τ un ensemble de transitions actif par un marquage m. L'ensemble τ est alors *franchissable* et conduit à un nouveau marquage m' tel que $m'(p) = m(p) + |{}^{(t)}p \cap \tau| - |p^{(t)} \cap \tau|$ pour tout $p \in P$.

On note :

- $m[\tau\rangle$ si τ est *franchissable* à m ;
- $m[\tau\rangle m'$ si $m[\tau\rangle$ et que τ conduit au marquage m'.

Sous l'hypothèse de *non simultanéité* (*no concurrency assumption*), τ est un singleton, c'est-à-dire $\tau = \{t\}$, est réduit à une seule transition active par m. On dit alors que t est *franchissable* et conduit à un marquage m' tel que $m'(p) = m(p) + |{}^{(t)}p \cap \{t\}| - |p^{(t)} \cap \{t\}|$ pour tout $p \in P$.

Dans le cas général où τ n'est pas réduit à un singleton, on parle d'hypothèse de *simultanéité*, situation dans laquelle plusieurs transitions peuvent être franchies en même temps.

Définition 1.9. Une *séquence franchissable* à partir d'un marquage m_0 est une suite ordonnée (possiblement vide) d'ensembles de transitions $\sigma = \tau_1 ... \tau_k$ tel que

$$m_0[\tau_1\rangle m_1[\tau_2\rangle m_2 ... [\tau_k\rangle m_k.$$

On note :
- $m_0[\sigma\rangle$ si σ est *franchissable* à m_0 ;
- $m_0[\sigma\rangle m'$ si $m_0[\sigma\rangle$ et que σ conduit au marquage m'.

Sous l'hypothèse de *non simultanéité*, chaque τ_i est un singleton et σ est une séquence de transitions appelée *trajectoire*.

1.1.3 Graphe d'accessibilité

Définition 1.10. Un marquage m est dit *accessible* dans (N, m_0) si et seulement s'il existe une séquence franchissable σ tel que $m_0[\sigma\rangle m$.

Définition 1.11. L'ensemble des *marquages accessibles* pour un RP marqué (N, m_0) est

$$R(N, m_0) = \{m \in \mathcal{M} |\ (\exists \sigma \mid \sigma \in (2^T)^* : m_0[\sigma\rangle m)\}.$$

Définition 1.12. Le *graphe d'accessibilité* est le graphe orienté (V, E) où :
- $V = R(N, m_0)$;
- $E = \{(m_1, m_2) \in V \times V \mid (\exists \tau \mid \tau \in 2^T : m_1[\tau\rangle m_2)\}$.

Les sommets d'un graphe d'accessibilité constituent donc l'espace d'états du RP marqué (N, m_0). De plus, sous l'hypothèse de *non simultanéité*, les arcs du graphe d'accessibilité sont étiquetés par les transitions du RP.

1.1.4 Cas particuliers

On distingue, dans les RP, deux types particuliers de réseau.

Définition 1.13. Un *graphe marqué* est un RP (P, T, ε) pour lequel $|^{(t)}p| = |p^{(t)}| = 1$ pour tout $p \in P$, c'est-à-dire que chaque place a exactement une seule transition d'entrée et une seule transition de sortie.

Remarque Un transition active est *persistante*, c'est-à-dire qu'elle demeure active jusqu'à ce qu'elle soit franchie.

Exemple Le RP de la figure 1.3 est un graphe marqué.

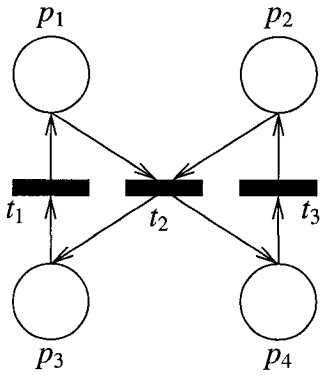


Figure 1.3 – Exemple d'un graphe marqué

Définition 1.14. On appelle *graphe d'état* un RP (P, T, ε) pour lequel $|^{(p)}t| = |t^{(p)}| = 1$ pour tout $t \in T$, c'est-à-dire que chaque transition a exactement une seule place d'entrée et une seule place de sortie.

Remarque Cette classe de RP correspond aux automates classiques de la théorie des automates, et la position de l'unique jeton indique l'état courant du système ainsi représenté.

Exemple Le RP de la figure 1.4 est un graphe d'état.

1.2 Systèmes à événements discrets

Un *système* se définit de façon générale comme un ensemble fonctionnel dont les parties sont interconnectées et échangent de la matière, de l'énergie ou de l'information. Dans la théorie des SED, un système est représenté par un *modèle* qui est une abstraction plus

ou moins concrète (suivant le niveau de détail voulu) dudit système. Dans ce modèle, les *événements* sont les éléments d'un ensemble dénombrable mais souvent fini, dont l'occurrence provoque la transition d'un état du système modélisé vers un autre. Le but de cette abstraction est de pouvoir analyser ou étudier le fonctionnement d'un système, concevoir et synthétiser un système, synthétiser un contrôleur pour un système, évaluer les performances d'un système et l'optimiser.

Synthétiser un contrôleur pour un système revient à déterminer ses entrées pour en contraindre le comportement dans le but d'atteindre un fonctionnement désiré. Un contrôleur f prend donc en entrée la sortie s d'un système G. Les deux composantes mises ensemble constituent ce que l'on appelle la *boucle de rétroaction* (voir la figure 1.5).

Il existe plusieurs méthodes pour la synthèse d'un contrôleur suivant le formalisme utilisé pour modéliser le système à contrôler [8, 15, 17, 35]. Dans la suite de cette section, seul le contrôle basé sur les langages est détaillé. Les notions présentées font appel à des éléments de la théorie des langages réguliers et des automates.

Définition 1.15. Soit K un langage défini sur un alphabet Σ. L'ensemble des préfixes de K est défini par

$$\overline{K} := \{u \mid u \in \Sigma^* \wedge \exists v \in \Sigma^* \wedge uv \in K\}.$$

K est dit *clos* si et seulement si $K = \overline{K}$.

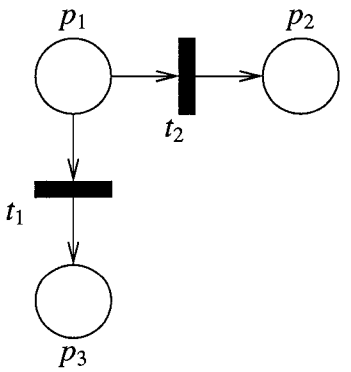


Figure 1.4 – Exemple d'un graphe d'état

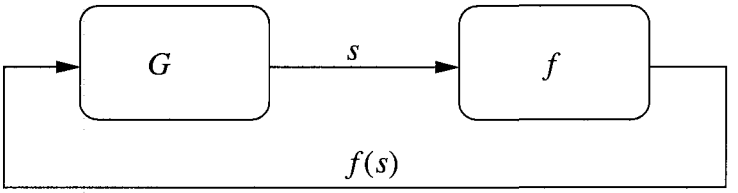


Figure 1.5 – Boucle de rétroaction

Définition 1.16. Un SED est défini à l'aide de deux langages : un langage *clos* K et un langage *marqué* K_m (représentant les tâches complètes), où $K \subseteq \Sigma^*$, $K \neq \emptyset$ et $K_m \subseteq K$.

En utilisant la construction de Myhill-Nerode, le SED est représenté par un générateur $G = (X, \Sigma, \delta, x_0, X_m)$, où $L(G) = K$ et $L_m(G) = K_m$.

Définition 1.17 (Hypothèse)**.** Un SED *contrôlé* est un SED dont l'ensemble d'événements Σ est partitionné en deux sous-ensembles disjoints : Σ_c, l'ensemble des événements *contrôlables*, et Σ_{uc}, l'ensemble des événements *incontrôlables*.

Définition 1.18. Un *contrôleur basé sur une loi de contrôle* est une application $f : K \to 2^{\Sigma}$ qui permet dynamiquement l'occurrence de certains événements dans le système.

La fonction f est la *politique de contrôle* et, pour chaque chaîne $s \in K$, la valeur de $f(s)$, appelée une *action de contrôle*, est l'ensemble des événements permis par le contrôleur f après l'exécution de s par G ($s \in L(G)$). L'expression f/G dénote la boucle de rétroaction.

Le langage généré par f/G est défini de façon inductive :

- $\epsilon \in L(f/G)$;
- $(s \in L(f/G) \wedge s\sigma \in L(G) \wedge \sigma \in f(s)) \Leftrightarrow s\sigma \in L(f/G)$.

Le langage marqué défini par f/G est $L_m(f/G) = L(f/G) \cap L_m(G)$.

Définition 1.19. Le SED f/G est *bloquant* si

$$\overline{L_m(f/G)} \neq L(f/G).$$

Dans le cas contraire c'est-à-dire si $\overline{L_m(f/G)} = L(f/G)$, le SED f/G est dit *non bloquant*. Cela signifie que toutes les tâches du SED (si on considère les chaînes du langage comme

des tâches) peuvent être complétées et donc qu'il n'existe pas de situation où une tâche commencée ne peut arriver à terme.

Définition 1.20. Soit deux langages K et L inclus dans Σ^* tels que $K \subseteq L$, $L = \overline{L}$ et $\Sigma_{uc} \subseteq \Sigma$. K est dit *contrôlable* (par rapport à L et Σ_{uc}) si

$$\overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}.$$

Ceci signifie que si l'on augmente toute sous-chaîne de K d'un événement incontrôlable (et qu'on obtient toujours une chaîne physiquement possible) alors la nouvelle chaîne est toujours une sous-chaîne de K. Autrement dit, l'ensemble des sous-chaînes de K est stable par ajout d'un événement incontrôlable.

Il est possible d'exercer un contrôle sous *observation partielle*. Dans ce cas l'ensemble des événements Σ est partitionné en deux sous-ensembles disjoints : Σ_o, l'ensemble des événements *observables*, et Σ_{uo}, l'ensemble des événements *non observables*. Une projection P masque les événements non observables et le contrôleur (voir la figure 1.6) est une fonction $f_P : P(L(G)) \rightarrow 2^{\Sigma}$.

Cependant dans ce mémoire, nous ne nous intéressons qu'à l'observation totale des SED.

Si le système d'intérêt est modélisé à l'aide d'un automate déterministe $(X, \Sigma, \delta, x_0, X_m)$ accessible, alors on peut effectuer un *contrôle basé sur les états*. Le contrôleur est une *fonction de contrôle* de type SFBC $f : X \rightarrow \Gamma$, où $\Gamma = \{\Sigma' \mid \Sigma_u \subseteq \Sigma' \wedge \Sigma' \subseteq \Sigma\}$.

Si $\sigma \in f(x)$, alors σ est *permis* en x, sinon il est *prohibé*. Les éléments de Γ sont des *actions de contrôle*.

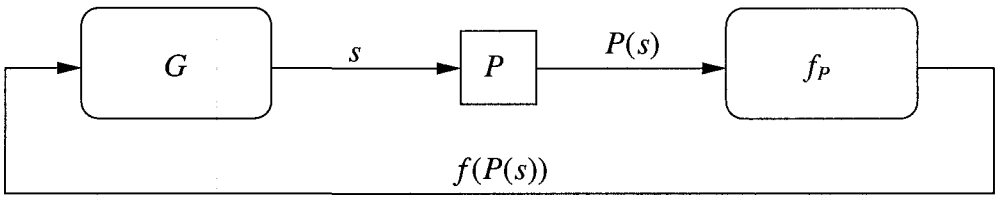


Figure 1.6 – Boucle de rétroaction sous observation partielle

1.3 *Workflows*

Les *workflows* sont des collections de tâches coordonnées ayant pour but d'exécuter un processus complexe bien défini [20]. Nous donnons dans la suite des définitions et notions relatives aux *workflows* et aux systèmes de gestion de *workflows*.

1.3.1 Définition

Définitions 1.21. Une *tâche* est un travail à effectuer par une ou plusieurs *ressources* dans un intervalle de temps prédéfini. Une *tâche* est atomique, c'est-à-dire qu'elle ne peut pas être divisée en de plus petites *tâches*. Une *ressource* est une entité, un individu ou un dispositif, qui peut effectuer une *tâche*, d'elle-même ou en coordination avec d'autres *ressources*. Un *document* est l'entrée ou la sortie d'une *tâche*.

Définitions 1.22. Une *procédure* est un ensemble ordonné d'*activités de contrôle*, paires de *tâches* et d'ensembles de *classes de ressources*, et de *(sous) procédures*. L'ordre, qui peut être partiel, modélise la relation de précédence pour les activités de contrôle, les tâches et les procédures qui doivent être exécutées. Une *activité de contrôle* spécifie le routage du travail dans la procédure et la synchronisation des tâches. Une procédure a une entrée et une sortie : les entités servant d'entrée et de sortie des procédures sont les documents.

Définition 1.23. Un *job* est un processus modélisant l'exécution d'une certaine quantité de travail par rapport à une procédure. Il peut être caractérisé par une séquence d'événements ou bien par une séquence d'états. Un job correspond toujours à une procédure, mais à un moment donné, il peut y avoir plusieurs jobs pour lesquels la même procédure est exécutée.

Définition 1.24. Un *workflow* est un ensemble partiellement ordonné de *jobs*. Cet ordre peut être défini de plusieurs façons, par exemple, les jobs x et y satisfont $x \prec y$ si et seulement si x démarre avant que y démarre.

1.3.2 Système de gestion de *workflows*

Dans un processus d'affaires, la gestion de *workflows* vise à s'assurer que la bonne activité est exécutée par le bon individu au bon moment [1]. Bien qu'il soit possible de faire de la gestion de *workflows* sans système de gestion de *workflows*, ceux-ci donnent une forme

concrète à des concepts, techniques et méthodes essentiels pour la gestion de *workflows*. Le *Workflow Management Coalition* (WfMC) définit un système de gestion de *workflows* comme un système qui permet de complètement définir, gérer et exécuter des *workflows* à travers l'exécution de logiciels dont l'ordre d'exécution dépend de la représentation informatisée de la logique du *workflow* [34].

De façon plus précise, un système de gestion de *workflows* fournit les fonctions suivantes :

- définition des *tâches*, *procédures* et *jobs* ;
- traitement de l'information nécessaire à l'exécution des *tâches* et *jobs* ;
- gestion des *ressources* ;
- routage de l'information entre les *procédures* et les *ressources* ;
- supervision du *workflow* et collection des informations de gestion.

1.3.3 Modélisation des *workflows*

La modélisation d'un *workflow* consiste en la création d'une spécification du *workflow* [23], dans le but général de faire de l'analyse ou encore d'exécuter le *workflow* dans un WFMS. Pour ce faire il existe différentes approches et différents formalismes, notamment les RP et les approches basées sur la logique (par exemple la logique temporelle TL [23] ou encore la logique concurrente transactionnelle $\mathcal{CTR}$ [10]).

1.3.4 Validation et vérification des *workflows*

La *validation*, en général, consiste à s'assurer que le modèle d'un système correspond à celui-ci par rapport à l'usage envisagé de ce modèle. Dans le domaine de l'informatique, cela revient à s'assurer que la spécification du système à construire correspond bien aux attentes des utilisateurs. La *vérification* quant à elle permet de déterminer si le système construit est conforme à cette spécification.

Dans le cadre des *workflows*, la validation et la vérification consistent à déterminer si le modèle possède des propriétés désirables et ce avant l'exécution du modèle par un WFMS.

Ces propriétés sont généralement des propriétés de validité (*soundness* [32]), de consistance faible (*weakness soundness*) ou d'irréductibilité des constructions du modèle [1].

La consistance définie par van der Aaslt comprend en fait plusieurs propriétés désirables sur les modèles de *workflows*. Ces modèles de *workflows*, *WF-net*, sont basés sur une classe particulière de RP. La consistance, définie formellement, stipule que :

i) tous les processus commencés peuvent être terminés ;

ii) si le processus atteint la fin, alors tous les sous-processus doivent avoir terminé aussi ;

iii) toutes les tâches dans le *workflow* peuvent être exécutées.

La *consistance faible* définie dans le cadre de YAWL — Yet Another Workflow Language — assouplit la première condition de la consistance. En effet celle-ci, l'achèvement d'un processus qui a commencé, est un problème indécidable dans le cas général. Cette contrainte doit donc être relachée dans la pratique avec les *workflows*. L'*irréductibilité* concerne les régions d'annulation des *workflows*. Ces régions peuvent contenir des éléments qui ne peuvent être annulés pendant l'exécution de la tâche. Ces éléments peuvent être retirés du *workflow* sans en changer le comportement.

1.3.5 Contrôle des *workflows*

Au sens de la théorie du contrôle des SED, il n'y a, à notre connaissance, que peu de travail fait dans la direction du contrôle des *workflows* [18, 3].

1.3.6 Patrons de *workflows*

Les patrons sont une abstraction d'une forme concrète qui est récurrente dans des contextes non arbitraires [26]. Gamma et al. ont décrit vingt trois patrons correspondant à des problèmes récurrents dans les systèmes informatiques orientés objet [11]. Depuis 2000, van der Aalst et d'autres chercheurs ont effectué le même type de travail [33, 30, 28, 29, 27], mais appliqué aux *workflows*. Le but de ce travail continu est de fournir une base solide, indépendante de tout langage de modélisation de *workflows* et de tout produit commercial.

1. `http://www.yawl-system.com/research/verification.html`

Les *workflows* peuvent être observés sous plusieurs perspectives différentes. Jablonski a identifié ces perspectives dans [16]. Parmi celles-ci mentionnons :

1. la *perspective fonctionnelle* qui décrit les unités élémentaires de processus à exécuter ;
2. la *perspective opérationnelle* qui décrit comment les opérations d'un *workflow* sont implémentées ;
3. la *perspective comportementale* ou du *flot de contrôle* qui décrit la succession ou l'enchaînement des unités fonctionnelles ;
4. la *perspective informationnelle* qui détermine les données requises et produites par les unités fonctionnelles du *workflow* ;
5. la *perspective organisationnelle* qui désigne les acteurs (agents, individus, rôles, unités organisationnelles, systèmes) qui doivent exécuter les unités fonctionnelles du *workflow*.

Ces perspectives font partie intégrante d'un modèle plus complet et extensible. Pour certaines de ces perspectives, qui peuvent être vues comme des dimensions d'un *workflow*, var der Aalst et d'autres chercheurs ont défini des patrons [2]. Ceux-ci couvrent :

1. la perspective comportementale (quarante trois patrons ont été identifiés et sont détaillés dans [30]) ;
2. la perspective informationnelle (quarante patrons [28]) ;
3. la perspective des ressources (quarante trois patrons [29]) ;
4. la perspective de la gestion des exceptions ([27]).

1.3.7 La norme BPEL et une implémentation de la norme : ActiveBPEL

Plusieurs méthodes non formelles existent pour la représentation des *workflows*. WS-BPEL (ou plus simplement BPEL) [22], Web Services Business Process Execution Language, fait partie de celles-là. Il s'agit d'un langage basé sur les services Web permettant

2. `http://www.workflowpatterns.com/`

l'*orchestration* des *workflows*. En fait, BPEL étend le modèle d'interaction des services Web pour permettre le support des transactions d'affaires. Avec BPEL, les processus d'affaires sont modélisés de deux façons : de façon *abstraite* et de façon *exécutable*.

Les *processus d'affaires abstraits* sont des processus partiellement spécifiés qui n'ont pas pour but d'être exécutés. Ils masquent des détails de réalisation concrets, car ils ont un rôle descriptif pour l'observation du comportement du processus, agissant ainsi comme modèle (au sens *template* du terme) de processus. Ils sont donc comparables à des *bibliothèques* qui décrivent ce qui peut être fait par un processus, ainsi que les entrées et les sorties de celui-ci, sans préciser comment cela est fait. Les *processus d'affaires exécutables* sont des processus entièrement spécifiés et qui peuvent être exécutés.

Un processus décrit en BPEL est constitué d'*activités* reliées entre elles par des *liens*. Le chemin d'exécution suivi à travers les activités est déterminé, entre autres, par les valeurs de variables et l'évaluation d'expressions. BPEL, étant un dialecte XML, les activités, les liens et tous les éléments utilisés pour représenter les processus sont complètement décrits textuellement avec des balises.

BPEL est une norme proposée par l'OASIS. ActiveBPEL Community Edition Engine [3] consiste en une implémentation libre de BPEL développée en Java. Cet outil accepte en entrée des définitions de processus BPEL et éventuellement des fichiers WSDL et crée des représentations de procesus BPEL. Lorsqu'un message entrant déclenche une activité de départ, le moteur ActiveBPEL crée une instance du processus et l'exécute. Le moteur ActivePBEL, comme d'autres WFMS, prend en charge certains aspects techniques d'exécution comme la gestion de la persistance, les files de messages et les notifications.

Pour illustrer les éléments du langage, considérons l'exemple d'un processus d'achat représenté par la figure 1.7. Les éléments de la figure elle-même ne font pas partie de BPEL. Elle se lit comme suit. Après avoir reçu la commande d'un client, le processus démarre trois fils d'exécution parrallèles : le calcul du prix final de la commande, la sélection d'un service de livraison et la planification de la production et l'expédition de la commande.

3. `http://www.activevos.com/community-open-source.php`

Bien que certains des traitements peuvent s'exécuter en parallèle, il existe néanmoins des dépendances de contrôle et de données entre les trois fils d'exécution. Par exemple, le prix de l'expédition de la commande est nécessaire pour finaliser le calcul du prix et la date d'envoi est requise pour compléter la planification de la commande. Après la fin des trois fils d'exécution, la facture est traitée et envoyée au client. Pour donner un aperçu du langage BPEL, l'annexe A détaille la définition du processus BPEL qui correspond à la figure 1.7. Cette définition fait référence à des définitions WSDL de services Web qui ne seront pas détaillées pour des raisons de concision.

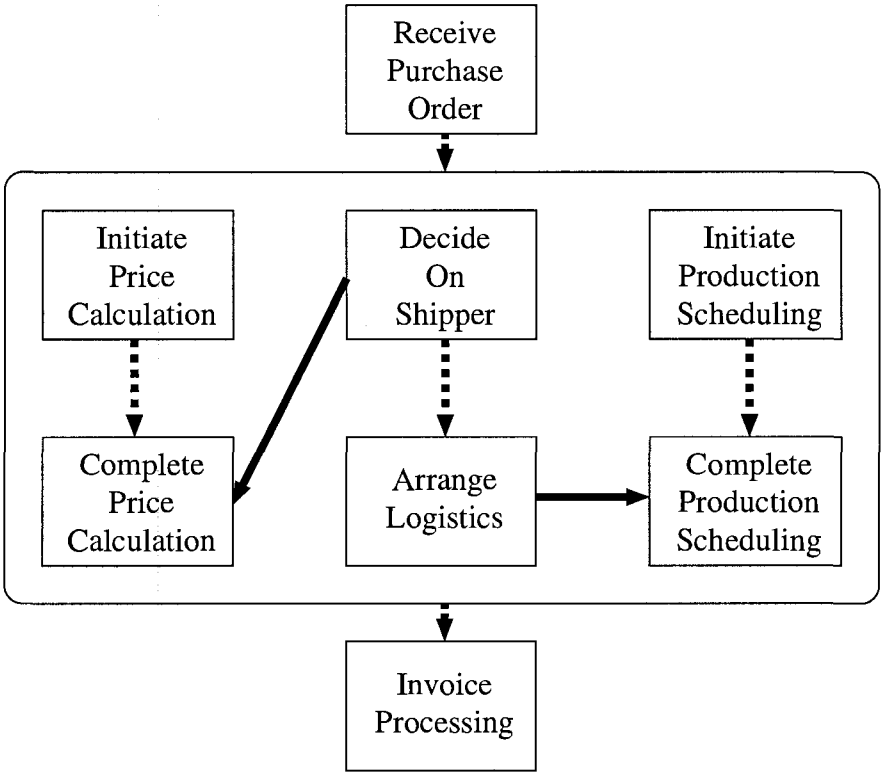


Figure 1.7 – Processus d'achat [22]

Chapitre 2

Algorithme pour la génération d'un graphe d'accessibilité

2.1 Description de l'algorithme

Dans la suite de ce mémoire, nous considérons un système à événements discrets paramétré (SEDP) $S = (S_i)_{1 \leq i \leq n}$, où $n \in \mathbb{N}^+$ et S_i est une structure réplicable [5], c'est-à-dire un automate fini déterministe $S_i = (Q_i, \Sigma_s \cup \Sigma_i, \delta_i, Q_{m,i})$ avec :

- Q_i un ensemble fini d'états indicés (par i) ;
- Σ_s un ensemble d'événements contrôlables non indicés ;
- Σ_i un ensemble d'événements indicés partitionné en deux sous-ensembles, $\Sigma_{c,i}$, l'ensemble des événements contrôlables, et $\Sigma_{u,i}$, l'ensemble des événements incontrôlables et $\Sigma_s \cap \Sigma_i = \emptyset$;
- $\delta_i : Q_i \times (\Sigma_s \cup \Sigma_i) \to Q_i$ la fonction partielle de transition de l'automate ;
- $Q_{m,i} \subseteq Q_i$ le sous-ensemble des états marqués.

La structure réplicable modélise le comportement de processus similaires, comme avec un *workflow* où les *cas* traités le sont suivant la même procédure, l'indice i représentant l'identité d'un processus ou d'un cas.

Exemple La figure 2.1 représente une structure réplicable. Celle-ci est utilisée dans la suite pour illustrer les éléments introduits dans ce chapitre.

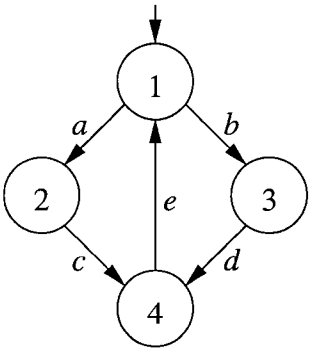


Figure 2.1 – Exemple d'une structure réplicable

L'algorithme effectue principalement la génération d'un graphe d'accessibilité G (voir section 1.1.3) à partir des éléments de S_i (mais en omettant leur indice et donc en considérant S_i comme un graphe d'état — voir section 1.1.4) et du Q-marquage initial m_0 (section 2.2) du SEDP. Il est issu de [4].

```
1  procedure Generate_Accessibility_Graph(in S, m0, out G)
2    CLOSED := {}
3    OPEN := {m0}
4    E := {} // Ensemble des arcs du graphe d'accessibilité
5    repeat
6      select m in OPEN
7      for each σ ∈ Σs ∪ Σ such that Δ(m,σ)!
8        m' := Δ(m,σ)
9        E := E ∪ {(m,m')}
10       if m' ∉ CLOSED
11         OPEN := OPEN ∪ {m'}
12     OPEN := OPEN \ {m}
13     CLOSED := CLOSED ∪ {m}
14   until OPEN = {}
15   G := (CLOSED, E)
```

Algorithme 2.1 – Algorithme de génération d'un graphe d'accessibilité

L'algorithme maintient à jour, au fur et à mesure de son exécution, deux ensembles de Q-marquages : *OPEN* l'ensemble des Q-marquages à parcourir et *CLOSED* l'ensemble des Q-marquages parcourus. Pour chacun des Q-marquages, le prédicat $\Delta(m, \sigma)!$ indique s'il existe un Q-marquage *successeur* au Q-marquage m à partir de l'événement σ ; le cas échéant, la procédure $\Delta(m, \sigma)$ (définie à la section 2.6) calcule le Q-marquage successeur. La section qui suit introduit les éléments de base nécessaires pour la définition du prédicat $\Delta(m, \sigma)!$ et la procédure $\Delta(m, \sigma)$.

2.2 Types abstraits de données

Les définitions qui suivent permettent d'expliciter les opérations de l'algorithme. Nous utilisons une notation ensembliste pour représenter les marquages et les opérations.

Définition 2.1. Soit $(x_1, x_2, ..., x_n)$ un n-uplet. L'opérateur π_i dénote la projection telle que

$$\pi_i(x_1, x_2, ..., x_n) := x_i.$$

Définition 2.2. Soit V_P un ensemble de *paramètres* avec $|V_P| \leq |Q|$. Un élément de V_P représente un paramètre du SEDP.

Si $|V_P| > 1$, l'introduction de paramètres additionnels peut permettre la simplification de sous-expressions de paramètres. Par exemple, si on définit $n_1 + n_2 = n$ alors l'expression $n_1 + n_2$ peut être remplacée par n.

Définition 2.3. Soit V_T un ensemble de *variables de transition* avec $|V_T| \leq |Q|^2$. Une *variable de transition* représente des transitions entre deux états.

$$V_T := \left\{ t_{q,q'} \mid q \in Q \wedge q' \in Q \wedge \exists \sigma \left(\sigma \in \Sigma \wedge \delta(q, \sigma) = q' \right) \right\}.$$

Remarque Plusieurs transitions de l'état q vers l'état q' sont représentées par la même variable de transition.

Définition 2.4. Soit $\mathbb{S}_E$ l'ensemble de toutes les expressions symboliques des états de Q.

$$\mathbb{S}_E := \mathcal{P}(V_P \cup V_T) \times \mathcal{P}(V_T) \times \mathbb{Z}.$$

Exemple L'écriture abstraite $(\{n, t_{4,1}\}, \{t_{1,3}\}, -3) \in \mathbb{S}_E$ représente l'expression infixée $n + t_{4,1} - t_{1,3} - 3$.

Définition 2.5. On appelle *Q-marquage* sur Q une fonction $m : Q \to \mathbb{S}_E$. On dénote par $\mathcal{M}$ l'ensemble des Q-marquages.

Exemple La figure 2.2 montre le système de la figure 2.1 avec les variables de transition entre les états correspondants. Donc $V_T = \{t_{1,2}, t_{1,3}, t_{2,4}, t_{3,4}, t_{4,1}\}$.

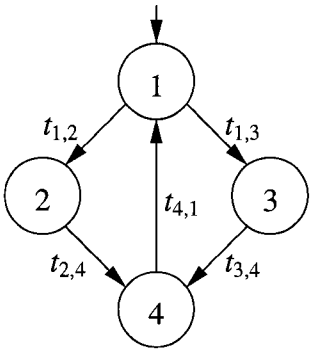


Figure 2.2 – Exemple de variables de transition

Avec $V_P = \{n\}$,

$$\begin{aligned} m : Q &\to \mathbb{S}_E \\ 1 &\mapsto (\{n, t_{4,1}\}, \{t_{1,3}\}, 0) \\ 2 &\mapsto (\emptyset, \emptyset, 0) \\ 3 &\mapsto (\{t_{1,3}\}, \{t_{3,4}\}, 0) \\ 4 &\mapsto (\{t_{3,4}\}, \{t_{4,1}\}, 0) \end{aligned}$$

en est un Q-marquage.

Par la suite, pour des raisons de concision, nous utilisons une notation sous forme de vecteur pour les Q-marquages en supposant les états de Q ordonnés ou numérotés de 1 à $|Q|$.

Notation Soit $e \in \mathbb{S}_E$. On écrit $[\![e]\!]$ pour dénoter l'interprétation de l'expression algébrique elle-même. Cette notation est utilisée par la suite pour composer des expressions en employant les opérateurs usuels $(+, -, \sum)$ de l'arithmétique. Par la suite, par abus de langage, l'expression symbolique poura être confondue à son interprétation.

Exemple $m = [(\{n, t_{4,1}\}, \{t_{1,3}\}, 0), (\emptyset, \emptyset, 0), (\{t_{1,3}\}, \{t_{3,4}\}, 0), (\{t_{3,4}\}, \{t_{4,1}\}, 0)]$ est la représentation abstraite du Q-marquage $m = [n + t_{4,1} - t_{1,3}, 0, t_{1,3} - t_{3,4}, t_{3,4} - t_{4,1}]$.

Définitions 2.6. Soit m un Q-marquage.

- $m(q)$ est *l'expression symbolique* de l'état $q \in Q$, c'est-à-dire la valeur de la fonction m en q.
- $m(q)_P$ est *l'ensemble des paramètres* de l'expression symbolique, c'est-à-dire

$$m(q)_P := \pi_1(m(q)) \setminus V_T.$$

- $m(q)_T^+$ est *l'ensemble des variables de transition positives* de l'expression symbolique, c'est-à-dire

$$m(q)_T^+ := \pi_1(m(q)) \setminus V_P.$$

- $m(q)_T^-$ est *l'ensemble des variables de transition négatives* de l'expression symbolique, c'est-à-dire

$$m(q)_T^- := \pi_2(m(q)).$$

- $m(q)_C$ est *la partie constante* de l'expression symbolique, c'est-à-dire (si l'on admet seulement la constante 0 alors cette partie n'est pas utilisée)

$$m(q)_C := \pi_3(m(q)).$$

- $m(q)_T^\pm$ est *la paire des ensembles de variables de transition en partie positive et en partie négative* de l'expression symbolique, c'est-à-dire

$$m(q)_T^\pm := (m(q)_T^+, m(q)_T^-).$$

- $m(q)_T$ est *l'ensemble de toutes les variables de transition* de l'expression symbolique, c'est-à-dire

$$m(q)_T := m(q)_T^+ \cup m(q)_T^-.$$

- $m(q)_V$ est *l'ensemble des paramètres et des variables de transition* de l'expression symbolique, c'est-à-dire

$$m(q)_V := m(q)_P \cup m(q)_T.$$

2.3 Invariants

Durant l'exécution de l'algorithme, les nœuds du graphe d'accessibilité sont générés à la volée. Les *invariants* permettent de s'assurer que chacun des nœuds introduits dans le graphe est correct. Ces invariants définissent donc, une par une, les contraintes qui font que les expressions symboliques et les marquages soient dans une forme spécifique.

1. Les paramètres n'apparaissent pas en partie négative dans un Q-marquage

$$\left(\bigcup_{q\in Q}\pi_2(m(q))\right)\cap V_P = \emptyset. \tag{I1}$$

   Cet invariant est toujours vrai implicitement par définition de $\mathbb{S}_E$.

2. Au moins un paramètre apparaît dans un Q-marquage comme terme positif (existence).

$$\left(\bigcup_{q\in Q} m(q)_P\right) \neq \emptyset$$

   ou

$$\bigcup_{q\in Q} m(q)_P = V_P \tag{I2}$$

   si les simplifications sur les paramètres ne sont pas permises.

3. Chaque paramètre apparaît au plus une fois dans un Q-marquage (unicité).

$$\left(\bigcup_{q_1\in Q}\bigcup_{q_2\in Q\setminus\{q_1\}} m(q_1)_P \cap m(q_2)_P\right) = \emptyset. \tag{I3}$$

4. Chaque variable de transition apparaît au plus une fois comme terme positif dans un Q-marquage (unicité).

$$\left(\bigcup_{q_1\in Q}\bigcup_{q_2\in Q\setminus\{q_1\}} m(q_1)_T^+ \cap m(q_2)_T^+\right) = \emptyset. \tag{I4}$$

5. Chaque variable de transition apparaît au plus une fois comme terme négatif dans un Q-marquage (unicité).

$$\left(\bigcup_{q_1 \in Q} \bigcup_{q_2 \in Q \setminus \{q_1\}} m(q_1)_T^- \cap m(q_2)_T^-\right) = \emptyset. \tag{I5}$$

6. Une variable de transition apparaît dans un terme positif si et seulement si elle apparaît dans un terme négatif d'un même Q-marquage (correspondance).

$$\bigcup_{q \in Q} m(q)_T^+ = \bigcup_{q \in Q} m(q)_T^-. \tag{I6}$$

7. Une expression symbolique d'un Q-marquage avec des variables de transition positives, avec une constante positive et sans variables de transition négatives ne peut pas inclure tous les paramètres (pas de dépassement).

$$m(q)_C \geq 0 \wedge m(q)_T^- = \emptyset \wedge m(q)_T^+ \neq \emptyset \Rightarrow m(q)_P \neq V_P, \text{ pour tout } q \in Q. \tag{I7}$$

8. Une expression symbolique d'un Q-marquage avec des variables de transition négatives, avec une constante négative et aucune variable de transition positive doit inclure au moins un paramètre (pas de soupassement).

$$m(q)_C \leq 0 \wedge m(q)_T^- \neq \emptyset \wedge m(q)_T^+ = \emptyset \Rightarrow m(q)_P \neq \emptyset, \text{ pour tout } q \in Q. \tag{I8}$$

9. Le nombre total de jetons reste toujours constant (conservation et conséquence de I2, I3, I4, I5, et I6)

$$\sum_{q \in Q} [\![m(q)]\!] = \sum_{q \in Q} [\![m_0(q)]\!]$$

ou

$$\sum_{q \in Q} [\![m(q)]\!] = \sum_{n \in V_P} n \tag{I9}$$

si les simplifications ne sont pas permises.

Notation On écrit $m \models \mathrm{I}$ si le Q-marquage m satisfait l'invariant I et $m \not\models \mathrm{I}$ si le Q-marquage m ne satisfait pas l'invariant I. Dans le cas où l'invariant I peut s'écrire

$$Pred(q), \text{ pour tout } q \in Q$$

où *Pred* est un prédicat sur Q, on note $m(q) \models \mathrm{I}$ si et seulement si $Pred(q)$ a la valeur *vraie* et $m(q) \not\models \mathrm{I}$ dans le cas contraire.

2.4 Simplifications

Lors de la génération d'un graphe d'accessibilité, il est avantageux d'obtenir des expressions symboliques concises dans les Q-marquages. La création d'un nouveau marquage nécessite donc des simplifications d'expressions symboliques.

Définition 2.7. Une simplification est *correcte* si et seulement si
- une expression simplifiée représente la même valeur que l'expression de départ (non simplifiée) d'un point de vue algébrique ;
- le marquage obtenu satisfait tous les invariants.

Les simplifications introduites dans cette section sont correctes. Dans les preuves, seuls les invariants concernant les variables de transition (I4 à I6 et I7 à I9) sont examinés, car les simplifications ne concernent que les variables de transition. Seules les variables qui sont directement concernées par la simplification vont être supprimées.

2.4.1 Simplification par rapport à une seule place

Des sous-expressions de la forme $t_{q,q'} - t_{q,q'}$ peuvent être retirées de $m(p)$ sans en changer la valeur. De manière générale

$$\pi_1(m(p)) := \pi_1(m(p)) \setminus V_{S1}(m(p)) \quad \pi_2(m(p)) := \pi_2(m(p)) \setminus V_{S1}(m(p)), \tag{S1}$$

où V_{S1} est une fonction sur $\mathbb{S}_E$ qui donne l'ensemble des variables de transition qui appa-

raissent à la fois en partie positive et en partie négative d'une expression symbolique

$$\begin{aligned} V_{S1} : \mathbb{S}_E &\rightarrow V_T \\ e &\mapsto m(e)_T^+ \cap m(e)_T^-. \end{aligned}$$

Exemple À partir du Q-marquage $m = [n - t_{1,2}, 0, 0, t_{1,2}]$ de l'exemple de la figure 2.2, l'occurrence de l'événement e (sous l'hypothèse que $t_{1,2} = 1$) conduit au marquage $[n - t_{1,2} + t_{1,2}, 0, 0, 0]$. La sous-expression $t_{1,2} - t_{1,2}$ peut être éliminée de la place 1 et on obtient alors le marquage $[n, 0, 0, 0]$.

Proposition 2.4.1. *La simplification S1 est correcte.*

Preuve. Soient m' un Q-marquage avant l'application de la simplification S1, m son Q-marquage prédecesseur (le cas où m' est le Q-marquage initial est trivial puisqu'il ne contient pas alors de variable de transition et la simplification n'est pas nécessaire) et m'' le Q-marquage après l'application de la simplification S1.

Par définition de V_{S1}, l'expression $(V_{S1}(m'(p)), V_{S1}(m'(p)), 0)$ est toujours d'interprétation nulle. Ainsi

$$[\![m''(p)]\!] = [\![(\pi_1(m'(p)) \setminus V_{S1}(m'(p)),\ \pi_2(m'(p)) \setminus V_{S1}(m'(p)),\ \pi_3(m'(p))]\!].$$

Puisque les variables de transition sont retirées par paire, c'est-à-dire le terme $t_{q,q'}$ dans la partie positive et le terme $-t_{q,q'}$ dans la partie négative, alors les invariants I4 à I6 restent satisfait après la simplification.

Montrons par l'absurde, qu'après la simplification S1, $m''(p) \models$ I7, c'est-à-dire supposons que m satisfait tous les invariants et que $m''(p) \not\models$ I7. Les Q-marquages m et m'' sont liés par les relations suivantes :

$$[\![m(p_0)]\!] = e_1 + t_{q,q'},\ [\![m(p_1)]\!] = e_2 - t_{q,q'}$$

et

$$[\![m''(p)]\!] = e_1 + e_2$$

où $p \in \{p_0, p_1\}$. Puisque $m''(p) \not\models$ I7, alors

$$[\![m''(p)]\!] = \left(\sum_{n \in V_P} n\right) + (t_1 + t_2 + ... + t_L) + C.$$

avec $\{t_1, t_2, ..., t_L\} \subset V_T$ et $C \in \mathbb{Z}$.

Ainsi tous les paramètres sont dans e_1 et e_2 et dans le Q-marquage m le seul terme négatif dans les places p_0 et p_1 est $-t_{q,q'}$, sinon dans le Q-marquage m'' on n'a pas de dépassement dans la place p. Toujours sous l'hypothÃ¨se de dÃ©passement, la variable de transition t_1 apparaît en tant que terme positif dans le Q-marquage m soit à la place p_0, soit à la place p_1. La variable t_1 doit aussi apparaître dans un terme négatif, par I6, dans une place $p_2 \notin \{p_0, p_1\}$. Puisque $m(p_2) \models$ I8, alors $-t_1$ doit apparaître dans une expression ayant un terme positif constitué d'une variable de transition, car tous les paramètres sont dans $m(p_0)$ et $m(p_1)$. On poursuit le même raisonnement avec cette nouvelle variable et ainsi de suite, ce qui conduit à une contradiction puisque l'ensemble des places est fini.

Procédons également par l'absurde pour montrer que $m''(p) \models$ I8, en considérent le Q-marquage m tel que précédemment introduit, et en supposant que celui-ci satisfait tous les invariants et que $m''(p) \not\models$ I8. Comme $m''(p) \not\models$ I8, on peut écrire

$$[\![m''(p)]\!] = -t_1 - t_2 - ... - t_L + C$$

avec $\{t_1, t_2, ..., t_L\} \subset V_T$ et $C \in \mathbb{Z}$, ce qui signifie que e_1 et e_2 ne contiennent que des variables en terme négatif (et pas de paramètre). Alors $[\![m(p_1)]\!] = e_2 - t_{q,q'}$ ne contient que des variables en terme négatif, c'est-à-dire $m(p_1) \not\models$ I8. Ceci est une contradiction et $m''(p) \models$ I8. ∎

2.4.2 Simplification par rapport à deux places

Si dans un Q-marquage m, $m(q)$ et $m(q')$ contiennent une sous-expression commune à un signe près, alors elle peut être remplacée par la variable de transition $t_{q,q'}$ sous les conditions qu'il existe une telle variable de transition entre q et q' (de manière symétrique

$t_{q',q}$ entre q' et q) et que $t_{q,q'}$ soit absente de m ou apparaisse dans $m(q)$ et $m(q')$.

Exemple À partir du marquage $m = [n_1 - t_{1,2}, 0, 0, n_2 + t_{1,2}]$ de l'exemple de la figure 2.2, l'occurrence de l'événement e conduit au marquage $[n_1 - t_{1,2} + t_{4,1}, 0, 0, n_2 + t_{1,2} - t_{4,1}]$. La sous-expression $t_{4,1} - t_{1,2}$ apparaît dans $m(1)$ et la sous-expression $-(t_{4,1} - t_{1,2})$ apparaît dans $m(4)$. Ainsi, $t_{4,1} - t_{1,2}$ peut être remplacée par $t_{4,1}$ puisqu'il existe une transition de l'état 4 vers l'état 1. On obtient alors le marquage $[n_1 + t_{4,1}, 0, 0, n_2 - t_{4,1}]$.

Malheureusement, comme le montre l'exemple suivant, une telle simplification peut résulter en un marquage qui viole l'invariant I8 et parfois I7. Un changement de variable est alors nécessaire.

Exemple À partir cette fois-ci du marquage $m = [n - t_{1,2}, 0, 0, t_{1,2}]$ et de l'occurrence de l'événement e, le marquage obtenu après la simplification est $[n + t_{4,1}, 0, 0, -t_{4,1}]$ et il ne satisfait pas les invariants I7 et I8. Posons $t'_{4,1} = n + t_{4,1}$. Alors $-t_{4,1} = n - t'_{4,1}$ et le marquage devient $[t'_{4,1}, 0, 0, n - t'_{4,1}]$, c'est-à-dire $[t_{4,1}, 0, 0, n - t_{4,1}]$ à un renommage près.

De manière générale, si $t_{q,q'} \in V_{S2S3}(m(q), m(q'))$ ou $t_{q,q'} \notin m(p)_T$ pour tout $p \in Q$,

$$\begin{aligned}
\pi_1(m(q)) &:= \pi_1(m(q)) \setminus V_{S2S3}(m(q), m(q')) \\
\pi_2(m(q)) &:= (\pi_2(m(q)) \setminus V_{S2S3}(m(q), m(q'))) \cup \{t_{q,q'}\} \\
\pi_1(m(q')) &:= (\pi_1(m(q')) \setminus V_{S2S3}(m(q), m(q'))) \cup \{t_{q,q'}\} \\
\pi_2(m(q')) &:= \pi_2(m(q')) \setminus V_{S2S3}(m(q), m(q'))
\end{aligned} \tag{S2}$$

où

$$\begin{aligned}
V_{S2S3} : \mathbb{S}_E \times \mathbb{S}_E &\to V_T \\
(e_1, e_2) &\mapsto (e_{1_T}^+ \cap e_{2_T}^-) \cup (e_{1_T}^- \cap e_{2_T}^+).
\end{aligned}$$

est l'ensemble des variables de transition communes à e_1 et e_2 au signe près et si le mar-

quage résultant satisfait I7 et I8, ou

$$\begin{aligned}
\pi_1(m(q)) &:= (\pi_1(m(q)) \setminus V_{S2S3}(m(q), m(q'))) \cup m(q')_P \\
\pi_2(m(q)) &:= (\pi_2(m(q)) \setminus V_{S2S3}(m(q), m(q'))) \cup \{t_{q,q'}\} \\
\pi_1(m(q')) &:= (\pi_1(m(q')) \setminus (m(q')_P \cup V_{S2S3}(m(q), m(q')))) \cup \{t_{q,q'}\} \\
\pi_2(m(q')) &:= \pi_2(m(q')) \setminus V_{S2S3}(m(q), m(q'))
\end{aligned} \tag{S3}$$

autrement et si $m(q')_P \neq \emptyset$.

Proposition 2.4.2. *La simplification S2 est correcte.*

Preuve. Par définition de V_{S2S3}, $[\![m(q)]\!] = e_1 - e$ et $[\![m(q')]\!] = e_2 + e$, où e est la sous-expression commune c'est-à-dire $e_T = V_{S2S3}(m(q), m(q'))$. Puisque m satisfait les invariants I4 à I6, les variables de e n'apparaissent que dans $m(q)$ et $m(q')$. Le retrait de e jetons de q et l'ajout de e jetons dans q' est équivalent au retrait de $t_{q,q'}$ jetons de q et à l'ajout de $t_{q,q'}$ jetons dans q'. Ainsi e est remplacée par $t_{q,q'}$ dans $m(q)$ et $m(q')$, puisque $t_{q,q'} \notin m(p)_T$ pour tout $p \in Q, p \neq q$ et $p \neq q'$, les invariants I4 à I6 restent satisfaits. Comme le nouveau marquage satisfait les invariants I7 et I8 la simplification S2 est correcte. ∎

Proposition 2.4.3. *Si $m(q')_P \neq \emptyset$ alors la simplification S3 est correcte.*

Preuve. La preuve est similaire à celle de la proposition 2.4.2. Mais comme les invariants I8 et parfois I7 ne sont pas satisfaits, alors e_1 ne contient que des termes négatifs, c'est-à-dire que $m \not\models$ I8 avec $[\![m(q)]\!] = e_1 - t_{q,q'}$, et parfois e_2 contient que des termes positifs et tous les paramètres, c'est-à-dire que $m \not\models$ I7 avec $[\![m(q')]\!] = e_2 + t_{q,q'}$, car le marquage initial satisfait l'invariant I9.

Écrivons $[\![m(q')]\!] = e'_2 + e''_2 + t_{q,q'}$, où e'_2 contient les paramètres et e''_2 contient les variables de transition. Posons $t'_{q,q'} = e'_2 + t_{q,q'}$ (changement de variable). Alors $t_{q,q'} = t'_{q,q'} - e'_2$ et $e_1 - t_{q,q'} = e_1 + e'_2 - t'_{q,q'}$. Ainsi, par renommage de $t'_{q,q'}$ par $t_{q,q'}$, $[\![m(q)]\!] = e_1 + e'_2 - t_{q,q'}$ et $[\![m(q')]\!] = e''_2 + t_{q,q'}$ et tous les invariants, en particulier I7 et I8, sont satisfaits. ∎

2.5 Variable de transition biaisée et alias

Définition 2.8. Une variable de transition t_{q_1,q_2} entre les états q_1 et q_2 de Q est dite *biaisée* dans un Q-marquage m par rapport à la place q si et seulement si

$$(\exists q \mid q \in Q \setminus \{q_1, q_2\} : t_{q_1,q_2} \in m(q)_T).$$

Dans le cas contraire, on dit que t_{q_1,q_2} est *non biaisée* dans le Q-marquage m.

Lors de la génération d'un graphe d'accessibilité, la variable t_{q_1,q_2} apparaît pour la première fois dans un Q-marquage aux places q_1 en partie négative et q_2 en partie positive. Après une transition sur un événement σ_1 entre les places q_2 et q_3, le terme t_{q_1,q_2} en partie positive de la place q_2 peut changer de place pour se retrouver en partie positive à la place q_3. Ce changement de place introduit un *biais* dans le Q-marquage et t_{q_1,q_2} est la variable biaisée. Les biais peuvent apparaître autant pour le terme positif que négatif d'une variable de transition et ceci pour une variable de transition biaisée ou non.

Exemple Relativement à l'exemple de la figure 2.2, la variable de transition $t_{1,3}$ est *non biaisée* dans le Q-marquage $[(\{n\}, \{t_{1,3}\}, 0), (\emptyset, \emptyset, 0), (\{t_{1,3}\}, \emptyset, 0), (\emptyset, \emptyset, 0)]$, mais elle est *biaisée* dans le Q-marquage $[(\{n\}, \{t_{1,3}\}, 0), (\emptyset, \emptyset, 0), (\emptyset, \emptyset, 0), (\{t_{1,3}\}, \emptyset, 0)]$ par rapport à la place 4.

Définition 2.9. Soit t_{q_1,q_2} une variable de transition biaisée dans un Q-marquage m par rapport à une place q suite à la transition de l'état p vers l'état q. La variable de transition $t_{p,q}$ est un *alias* pour la variable de transition t_{q_1,q_2} et l'*ensemble des alias* de la variable t_{q_1,q_2} dans le Q-marquage m est noté $m.A_{t_{q_1,q_2}}$:

$$t_{p,q} \in m.A_{t_{q_1,q_2}}.$$

Exemple La variable $t_{3,4}$ est un alias pour $t_{1,3}$, c'est-à-dire que $t_{3,4} \in m.A_{t_{1,3}}$, si $m = [(\{n\}, \{t_{1,3}\}, 0), (\emptyset, \emptyset, 0), (\emptyset, \emptyset, 0), (\{t_{1,3}\}, \emptyset, 0)]$.

La procédure `Update_Alias` met à jour les ensembles des alias d'un Q-marquage m', ayant pour prédecesseur le Q-marquage m, avec l'alias $t_{q,q'}$ (voir l'algorithme 2.2).

```
1 procedure Update_Alias(
2   in  m,     // Q-marquage prédecesseur de m'
3       t_{q,q'},  // variable de transition devenant un alias
4       m',    // Q-marquage dont les alias sont mis à jour
5 ) out m'
6 )
7 for each t_{q1,q2} ∈ (m(q)_T ∩ m'(q')_T) \ {t_{q,q'}} do
8   m'.A_{t_{q1,q2}} := m'.A_{t_{q1,q2}} ∪ {t_{q,q'}}
```

Algorithme 2.2 – Mise à jour des alias

Remarque Lorsqu'il n'y a pas d'ambiguïté sur le Q-marquage, celui-ci est omis de l'expression $t_{p,q} \in m.A_{t_{q_1,q_2}}$ et on écrit simplement $t_{p,q} \in A_{t_{q_1,q_2}}$.

L'invariant suivant indique que si $t_{p,q}$ est un alias pour t_{q_1,q_2} alors $[\![m(p)]\!] = 0$. Pour une place $p \in Q$,

$$\left(\exists q, q_1, q_2 \mid q, q_1, q_2 \in Q \wedge q \notin \{q_1, q_2\} : t_{p,q} \in m.A_{t_{q_1,q_2}}\right) \Rightarrow [\![m(p)]\!] = 0. \qquad \text{(I10)}$$

Ainsi pour maintenir cet invariant, un *alias* $t_{p,q} \in m.A_{t_{q_1,q_2}}$ doit être enlevé de l'ensemble $m.A_{t_{q_1,q_2}}$ dès que l'expression associée à $m(p)$ devient différente de 0. Si l'expression associée à $m(p)$ change de 0 à une expression différente de 0 alors la procédure `Apply_Alias` (voir l'algorithme 2.3) doit être appelée avec les paramètres m et p.

La procédure `Apply_Alias` cherche dans chaque ensemble d'alias lié à m, la plus longue chaîne d'alias $\langle t_{p_0,p_1}, t_{p_1,p_2}, ..., t_{p_{n-1},p_n} \rangle$ telle que $p_0 = p$. Les alias de cette chaîne sont ensuite enlevés pour être mis dans le Q-marquage m. La variable dont l'ensemble d'alias est examiné, et qui se trouve — assurément — dans la position p_n est déplacée dans la place p_0. L'éventualité que l'invariant I7 puisse être violé est détectée et les paramètres sont aussi déplacés de la place p_n vers la place p_0 le cas échéant. Pendant ces opérations, les expressions associées à certaines places peuvent changer de 0 à une expression différente de 0. Pour ces places il faut refaire les mêmes étapes, ce qui justifie la présence de la variable *File*.

```
1  procedure Apply_Alias(in m, p)
2    File := ⟨p⟩
3    repeat
4      p0 := head_and_remove(File)
5      for each m.A_{t_{q1,q2}} such that there exists t_{p0,p1} ∈ m.A_{t_{q1,q2}}
6        let T := {t_{p0,p1}, t_{p1,p2}, ..., t_{pn-1,pn}} ⊆ m.A_{t_{q1,q2}}
7        where n ≥ 1 is greater as possible
8        for each t_{pi-1,pi} ∈ T
9          // ajouter le terme -t_{pi-1,pi} dans l'expression de p_{i-1}
10         π2(m(p_{i-1})) := π2(m(p_{i-1})) ∪ {t_{pi-1,pi}}
11         /* si l'expression associée à m(p_i) change de 0
12          * à une expression différente de 0, appliquer les
13          * alias pour p_i */
14         if [[m(p_i)]] = 0 then File := File ^ ⟨p_i⟩
15         // ajouter le terme +t_{pi-1,pi} dans l'expression de p_i
16         π1(m(p_i)) := π1(m(p_i)) ∪ {t_{pi-1,pi}}
17         // enlever l'alias t_{pi-1,pi} de l'ensemble m.A_{t_{q1,q2}}
18         m.A_{t_{q1,q2}} := m.A_{t_{q1,q2}} \ {t_{pi-1,pi}}
19       /* déplacer le terme associé à la variable biaisée t_{q1,q2}
20        * de la place courante p_n vers la place p_0 */
21       if t_{q1,q2} ∈ π1(m(p_n)) then
22         π1(m(p_0)) := π1(m(p_0)) ∪ {t_{q1,q2}}   π1(m(p_n)) := π1(m(p_n)) \ {t_{q1,q2}}
23       else
24         π2(m(p_0)) := π2(m(p_0)) ∪ {t_{q1,q2}}   π2(m(p_n)) := π2(m(p_n)) \ {t_{q1,q2}}
25       /* si l'invariant I7 est violé, déplacer tous les
26        * paramètres de la place courante p_n vers p_0 */
27       if m(p_n) ⊭ I7 then
28         π1(m(p_0)) := π1(m(p_0)) ∪ m(p_n)_P   π1(m(p_n)) := π1(m(p_n)) \ m(p_n)_P
29   until File = ⟨⟩
30 end.
```

Algorithme 2.3 – Application d'un alias

Comme le montre les exemples suivants, les alias permettent de s'approcher ou même d'atteindre un marquage dans lequel il n'y a pas de variables biaisées, c'est-à-dire que les variables occupent leurs places respectives.

Exemple À partir du marquage $[n-t_{13}, 0, t_{13}, 0]$ de l'exemple de la figure 2.2, l'occurrence de l'événement d (avec $t_{13} = 1$) conduit au marquage $[n - t_{13}, 0, 0, t_{13}]$. La variable t_{13} est biaisée et t_{34} est un alias de t_{13}, c'est-à-dire que $t_{34} \in A_{t_{13}}$. L'occurrence de l'événement b (avec $n - t_{13} = 1$) conduit au marquage $[0, 0, n - t_{13}, t_{13}]$ et l'expression de la place 3 est différente de 0. L'alias t_{34} doit être retiré de $A_{t_{13}}$. La variable t_{34} doit donc être ajoutée dans les places 3 et 4, respectivement comme terme négatif et comme terme positif. La variable t_{13} (dans la place 4) est déplacée dans la place 3. Ceci conduit au marquage

$$[0, 0, n - t_{13} + t_{13} - t_{34}, t_{34}],$$

c'est-à-dire au marquage suivant après simplification

$$[0, 0, n - t_{34}, t_{34}].$$

Exemple Considérons les marquages suivants suite à des occurrences successives d'événements à partir de l'exemple de la figure 2.2 :

$$[n - t_{13}, 0, t_{13}, 0]$$
$$\downarrow d, t_{13}=1$$
$$[n - t_{13}, 0, 0, t_{13}] \quad t_{34} \in A_{t_{13}}$$
$$\downarrow b, n-t_{13}>1$$
$$[n - t_{13} - t_{13}, 0, t_{13}, t_{13}]$$

L'expression dans la place 3 est différente de 0 et t_{34} est un alias t_{13}. La variable t_{13} dans la place 4 est déplacée dans la place 3 (la place source de l'alias) et la variable t_{34} est ajoutée comme terme négatif dans la place 3 et comme terme positif dans la place 4. Ceci conduit au marquage

$$[n - t_{13} - t_{13}, 0, t_{13} + t_{13} - t_{34}, t_{34}],$$

c'est-à-dire

$$[n - t_{13}, 0, t_{13} - t_{34}, t_{34}]$$

après simplification (et renommage).

Les deux premiers exemples illustrent, entre autres, le déplacement de la variable biaisée dans la place source de l'alias considéré.

Exemple Considérons les marquages suivants suite à des occurrences successives d'événements à partir de l'exemple de la figure 2.2 :

$$[n - t_{13}, 0, t_{13}, 0]$$
$$\downarrow d, t_{13}=1$$
$$[n - t_{13}, 0, 0, t_{13}] \quad t_{34} \in A_{t_{13}}$$
$$\downarrow a, n-t_{13}=1$$
$$[0, n - t_{13}, 0, t_{13}] \quad t_{34}, t_{12} \in A_{t_{13}}$$
$$\downarrow e, t_{13}>1$$
$$[t_{41}, n - t_{13}, 0, t_{13} - t_{41}]$$

L'expression dans la place 1 est différente de 0. La variable t_{12} est ajoutée dans les places 1 et 2, respectivement comme terme négatif et comme terme positif, et la variable t_{13} (dans la place 2) est déplacée dans la place 1. Ceci conduit au marquage suivant

$$[t_{41} - t_{12} - t_{13}, n + t_{12}, 0, t_{13} - t_{41}].$$

L'invariant I7 est violé, le paramètre est aussi déplacé dans la place 1 pour obtenir le marquage

$$[n + t_{41} - t_{12} - t_{13}, t_{12}, 0, t_{13} - t_{41}] \quad t_{34} \in A_{t_{13}}.$$

Après simplification (sous-expression $t_{41} - t_{13}$ et renommage), ceci conduit au marquage suivant

$$[t_{41} - t_{12}, t_{12}, 0, n - t_{41}].$$

Exemple Considérons toujours la structure réplicable de la figure 2.2 et la suite des marquages suivant :

$$[n - t_{12}, t_{12}, 0, 0]$$
$$\downarrow c, t_{12} > 1$$
$$[n - t_{12}, t_{12} - t_{24}, 0, t_{24}]$$
$$\downarrow e, t_{24} > 1$$
$$[n + t_{41} - t_{12}, t_{12} - t_{24}, 0, t_{24} - t_{41}]$$
$$\downarrow b, n + t_{41} - t_{12} = 1$$
$$[0, t_{12} - t_{24}, n + t_{41} - t_{12}, t_{24} - t_{41}] \quad t_{13} \in A_{t_{12}}, t_{13} \in A_{t_{41}}$$
$$\downarrow e, t_{24} - t_{41} = 1$$
$$[t_{24} - t_{41}, t_{12} - t_{24}, n + t_{41} - t_{12}, 0]$$

L'expression dans la place 1 est différente de 0. La variable t_{13} est ajoutée comme terme positif et comme terme négatif. Les variables t_{12} et t_{41} sont déplacées de la place 3 vers la place 1. Ceci conduit au marquage

$$[t_{24} - t_{41} - t_{13} + t_{41} - t_{12}, t_{12} - t_{24}, n + t_{13}, 0] \quad t_{41} \in A_{t_{24}}.$$

La variable t_{24} est maintenant biaisée. L'invariant I7 est violé, le paramètre n est aussi déplacé dans la place 1 pour obtenir le marquage

$$[n + t_{24} - t_{12} - t_{13}, t_{12} - t_{24}, t_{13}, 0]$$

qui après simplification devient (la variable t_{24} disparaît avec son alias)

$$[n - t_{12} - t_{13}, t_{12}, t_{13}, 0].$$

Exemple Considérons la structure réplicable de la figure 2.3 et les Q-marquages suivants suite à des occurrences successives d'événements :

$$[n_1 - t_{12}, t_{12}, 0, 0, n_2, 0]$$
$$\downarrow b, t_{12}=1$$
$$[n_1 - t_{12}, 0, t_{12}, 0, n_2, 0] \quad t_{23} \in A_{t_{12}}$$
$$\downarrow c, t_{12}=1$$
$$[n_1 - t_{12}, 0, 0, t_{12}, n_2, 0] \quad t_{23}, t_{34} \in A_{t_{12}}$$
$$\downarrow d, t_{12}=1$$
$$[n_1 - t_{12}, 0, 0, 0, n_2 + t_{12}, 0] \quad t_{23}, t_{34}, t_{45} \in A_{t_{12}}$$
$$\downarrow f, n_2+t_{12}>1$$
$$[n_1 - t_{12}, 0, t_{53}, 0, n_2 + t_{12} - t_{53}, 0]$$

Puisque l'expression dans la place 3 est différente de 0 et que $t_{34} \in A_{t_{12}}$, la variable biaisée dans la place 5 est déplacée dans la place 3 (il faut noter que ceci est fait à la fin dans l'algorithme 2.3) et la variable t_{34} (alias de t_{12}) est ajoutée comme terme négatif dans la place 3 et comme terme positif dans la place 4. Ceci conduit au marquage

$$[n_1 - t_{12}, 0, t_{53} + t_{12} - t_{34}, t_{34}, n_2 - t_{53}, 0] \quad t_{23}, t_{45} \in A_{t_{12}}.$$

Mais comme l'expression dans la place 4 est différente de 0 et que $t_{45} \in A_{t_{12}}$, la variable t_{45} (alias de t_{12}) est ajoutée comme terme négatif dans la place 4 et comme terme positif dans la place 5. Ceci conduit au marquage

$$[n_1 - t_{12}, 0, t_{12} + t_{53} - t_{34}, t_{34} - t_{45}, n_2 + t_{45} - t_{53}, 0] \quad t_{23} \in A_{t_{12}}.$$

La variable t_{12} est toujours biaisée, mais par rapport à une seule place (la place 3) et il n'y a plus qu'un seul alias dans $A_{t_{12}}$ (l'alias t_{23}).

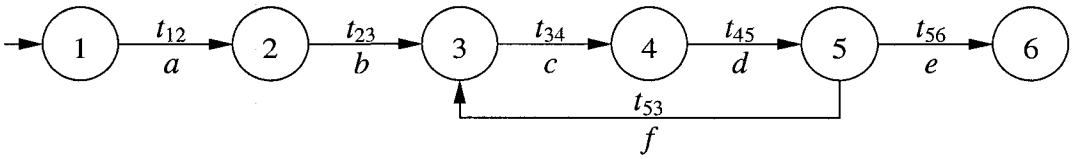


Figure 2.3 – Structure réplicable linéaire et retour en arrière

2.6 Génération d'un graphe d'accessibilité

La boucle principale de l'algorithme 2.1 sélectionne répétitivement un nœud dans l'ensemble *OPEN* et calcule ses successeurs suivant son Q-marquage. Le prédicat $\Delta(m,\sigma)!$ indique si la transition σ est définie à partir du nœud m. Sa valeur de retour est donnée par

$$\Delta(m,\sigma)! = \begin{cases} 1 & \text{si } \sigma \in \Sigma_s \text{ et } \Sigma_{q|\delta(q,\sigma)!}\, [\![m(q)]\!] = \Sigma_q\, [\![m_0(q)]\!] \quad \text{(C1)} \\ & \text{ou si } \sigma \in \Sigma \text{ et } (\exists q \mid q \in Q : \delta(q,\sigma)! \wedge [\![m(q)]\!] \neq 0) \quad \text{(C2)} \\ 0 & \text{autrement} \end{cases}$$

où

$$\delta(q,\sigma)! := (\exists q' \mid q' \in Q : \delta(q,\sigma) = q')\,.$$

Une transition $\sigma \in \Sigma_s$ est définie à partir d'un nœud m si tous les processus présents au départ (dans le nœud racine) sont prêts à effectuer la transition dans la structure réplicable. Dans le cas $\sigma \in \Sigma$, il doit exister au moins un processus de la structure réplicable prêt à effectuer la transition.

La procédure $\Delta(m,\sigma)$ calcule les nœuds successeurs du nœud courant. Elle considère deux cas pour ce calcul : $\sigma \in \Sigma_s$ et $\sigma \in \Sigma$. Dans le premier cas ($\sigma \in \Sigma_s$), tous les processus (jetons) se déplacent de l'état de départ vers l'état d'arrivée. L'algorithme 2.4 donne le détail de ce traitement. Le Q-marquage m' est un successeur du Q-marquage m.

```
1   for each q' ∈ Q
2     π1(m'(q')) := ⋃_{q|δ(q,σ)=q'} π1(m(q))
3     π2(m'(q')) := ⋃_{q|δ(q,σ)=q'} π2(m(q))
4     π3(m'(q')) := Σ_{q|δ(q,σ)=q'} π3(m(q))
```

Algorithme 2.4 – Procédure $\Delta(m,\sigma)$ pour $\sigma \in \Sigma_s$

Dans le second cas ($\sigma \in \Sigma$), la condition C2 peut trouver plusieurs états pour lesquels la transition sur σ peut être effectuée. Pour chaque état q, la procédure examine deux cas, suivant la valeur de $m(q)$: $m(q) > 1$ et $m(q) = 1$. Cette distinction est essentielle pour

faire apparaître *explicitement* les 0 dans les Q-marquages, ce qui par ailleurs sert aussi à l'évaluation de la condition C2.

Dans le premier cas ($m(q) > 1$), $t_{q,q'}$ jetons sont (symboliquement) déplacés de la place q à la place q'. Si avant ce déplacement, l'expression $m(q')$ est nulle, alors les alias éventuels doivent être appliqués (voir l'algorithme 2.5).

1 $m' := m$
2 $\pi_2(m'(q)) := \pi_2(m'(q)) \cup \{t_{q,q'}\}$
3 $\pi_1(m'(q')) := \pi_1(m'(q')) \cup \{t_{q,q'}\}$
4 **if** $[\![m(q')]\!] = 0$ **begin** `Apply_Alias(`m', q'`)` **end**

Algorithme 2.5 – Premier sous-cas du calcul de $\Delta(m, \sigma)$ pour $\sigma \in \Sigma$

Remarque Puisque $\pi_1(m'(q'))$ et $\pi_2(m'(q))$ sont des ensembles, il n'est pas possible de représenter les sous-expressions $t_{q,q'} + t_{q,q'}$ et $-t_{q,q'} - t_{q,q'}$. Cette situation survient lorsque $t_{q,q'} \in \pi_1(m(q'))$ et $t_{q,q'} \in \pi_2(m(q))$. Ainsi, les expression associées à $m(q)$ et $m(q')$ sont respectivement $e_1 - t_{q,q'}$ et $e_2 + t_{q,q'}$. Les expressions associées à $m'(q)$ et $m'(q')$ devraient respectivement être alors $e_1 - (t_{q,q'} + t_{q,q'})$ et $e_2 + (t_{q,q'} + t_{q,q'})$. En substituant $t_{q,q'} + t_{q,q'}$ par t et en renommant t par $t_{q,q'}$, les expressions associées à $m'(q)$ et $m'(q')$ deviennent respectivement $e_1 - t_{q,q'}$ et $e_2 + t_{q,q'}$. Ceci signifie que $m = m'$ et qu'il existe une boucle sur m pour la transition concernée. Ainsi, dans ce cas particulier les opérations de substitution et de renommage sont transparents et conformes à la représentation abstraite choisie pour les Q-marquages. Il n'est alors pas nécessaire de distinguer le cas où $t_{q,q'} \in \pi_1(m(q'))$ et $t_{q,q'} \in \pi_2(m(q))$ du cas où $t_{q,q'} \notin \pi_1(m(q'))$ et $t_{q,q'} \notin \pi_2(m(q))$ puisque dans les deux cas on obtient les mêmes expressions, c'est-à-dire $e_1 - t_{q,q'}$ et $e_2 + t_{q,q'}$.

Le second cas, pour lequel on considère symboliquement que $[\![m(q)]\!] = 1$, est légèrement plus complexe. Ce cas n'est possible que si $\pi_2(m(q)) \neq \emptyset \vee |\pi_1(m(q))| = 1$. Tous les jetons sont déplacés de la place q vers la place q' et les alias sont mis à jour pour les variables de transition (biaisés ou pas) qui vont changer de position (voir l'algorithme 2.6). Ces opérations sont conformes aux résultats théoriques attendus. Pour les variables de transition ainsi déplacées, la procédure `Update_Alias` met à jour les alias avec l'alias $t_{q,q'}$ (voir l'algorithme 2.2).

1 $m' := m$
2 $\pi_1(m'(q')) := \pi_1(m'(q')) \cup \pi_1(m'(q))$
3 $\pi_2(m'(q')) := \pi_2(m'(q')) \cup \pi_2(m'(q))$
4 $\pi_3(m'(q')) := \pi_3(m'(q')) + \pi_3(m'(q))$
5 $\pi_1(m'(q)) := \emptyset$
6 $\pi_2(m'(q)) := \emptyset$
7 $\pi_3(m'(q)) := 0$
8 `Update_Alias(`$m, m', t_{q,q'}$`)`
9 **`if`** $[\![m(q')]\!] = 0$ **`begin Apply_Alias(`**m', q'**`) end`**

Algorithme 2.6 – Deuxième sous-cas du calcul de $\Delta(m, \sigma)$ pour $\sigma \in \Sigma$

Remarque Si $t_{q,q'} \in \pi_2(m(q))$ et $t_{q,q'} \in \pi_1(m(q'))$, alors les expressions associées à $m(q)$ et $m(q')$ sont respectivement $e_1 - t_{q,q'}$ et $e_2 + t_{q,q'}$. Les expressions associées à $m'(q)$ et $m'(q')$ sont respectivement 0 et $e_1 - t_{q,q'} + e_2 + t_{q,q'} = e_1 + e_2$ (par simplification).

Tous les cas mentionnés précédemment constituent la procédure $\Delta(m, \sigma)$ qui calcule l'ensemble de Q-marquages successeurs de m (voir l'algorithme 2.7).

```
1  input m, σ; output Δ // marking set
2  Δ := ∅
3  if σ ∈ Σ_s
4    for each q' ∈ Q
5      π1(m'(q')) := ⋃_{q|δ(q,σ)=q'} π1(m(q))
6      π2(m'(q')) := ⋃_{q|δ(q,σ)=q'} π2(m(q))
7      π3(m'(q')) := Σ_{q|δ(q,σ)=q'} π3(m(q))
8    Δ := Δ ∪ {m'}
9  else  //  σ ∈ Σ
10   for each q such that C2 is true
11     q' := δ(q, σ)
12     begin // premier cas : [[m(q)]] > 1
13       m' := m
14       π2(m'(q)) := π2(m'(q)) ∪ {t_{q,q'}}
15       π1(m'(q')) := π1(m'(q')) ∪ {t_{q,q'}}
16       if [[m(q')]] = 0 begin Apply_Alias(m', q') end
17       Δ := Δ ∪ {m'}
18     end
19     if π2(m(q)) ≠ ∅ ∨ |π1(m(q))| = 1 begin // deuxième cas : [[m(q)]] = 1
20       begin
21         m' := m
22         π1(m'(q')) := π1(m'(q')) ∪ π1(m'(q))
23         π2(m'(q')) := π2(m'(q')) ∪ π2(m'(q))
24         π3(m'(q')) := π3(m'(q')) + π3(m'(q))
25         π1(m'(q)) := ∅
26         π2(m'(q)) := ∅
27         π3(m'(q)) := 0
28         Update_Alias(m, m', t_{q,q'})
29         if [[m(q')]] = 0 begin Apply_Alias(m', q') end
30         Δ := Δ ∪ {m'}
31       end
32     end
```

Algorithme 2.7 – Calcul de $\Delta(m, \sigma)$

Chapitre 3

Implémentation de l'algorithme

L'algorithme de génération d'un graphe d'accessibilité est implémenté à l'aide d'une méthode orientée objet. Cette implémentation est faite avec le langage objet Java.

3.1 Diagrammes de classes

3.1.1 Diagramme de classes des données du problème

La figure 3.1 donne le diagramme de classes qui représentent les données du problème à traiter. Ces données sont essentiellement le *système à événements discrets paramétré* et le *Q-marquage initial.*

Ces classes constituent le paquetage Java `udes.pdes`. En particulier, les interfaces `IState` et `IEvent` représentent respectivement les états et les événements du SED. Les objets de la classe `ParameterizedDiscreteEventSystem` utilisent le bibliothèque `JGraphT` [1] pour gérer en interne les états et les événements. Les classes `State` et `Event` fournissent une implémentation par défaut respectivement pour les interfaces `IState` et `IEvent`. Le marquage initial (classe `InitialMarking`) associe un paramètre (interface `IParameter`) à chaque état du SED.

1. `http://jgrapht.sourceforge.net/`

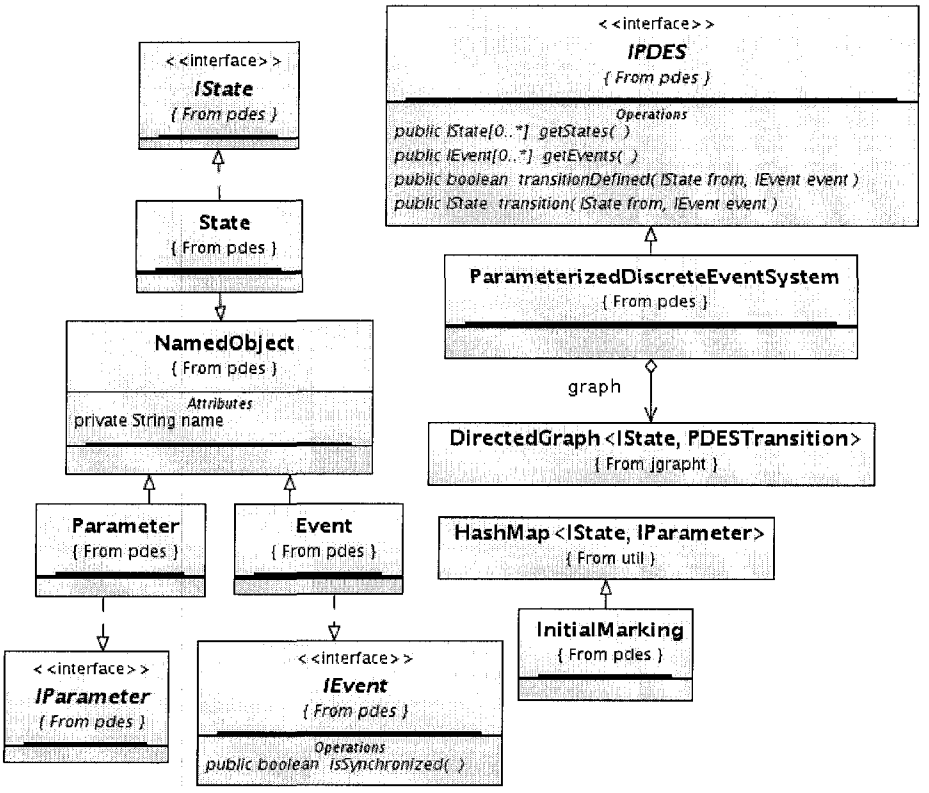


Figure 3.1 – Diagramme de classes des données du problème

3.1.2 Diagramme de classes de la génération d'un graphe d'accessibilité

La figure 3.2 donne le diagramme des classes de l'implémentation de l'algorithme de génération d'un graphe d'accessibilité proprement dit. La classe `ReachabilityGraph` implémente, d'une part, les structures de données (en utilisant les paquetages de `JGraphT`) représentant le graphe d'accessibilité (marquages et transitions) et, d'autre part, l'algorithme de génération :

1. La classe `Marking` implémente les Q-marquages basés sur les expressions symboliques implémentées par la classe `Expression`. Les transitions dans le graphe sont implémentées par la classe `Arc`. La classe `Node` sert principalement d'artifice pour la gestion des invariants et est un conteneur pour les Q-marquages.

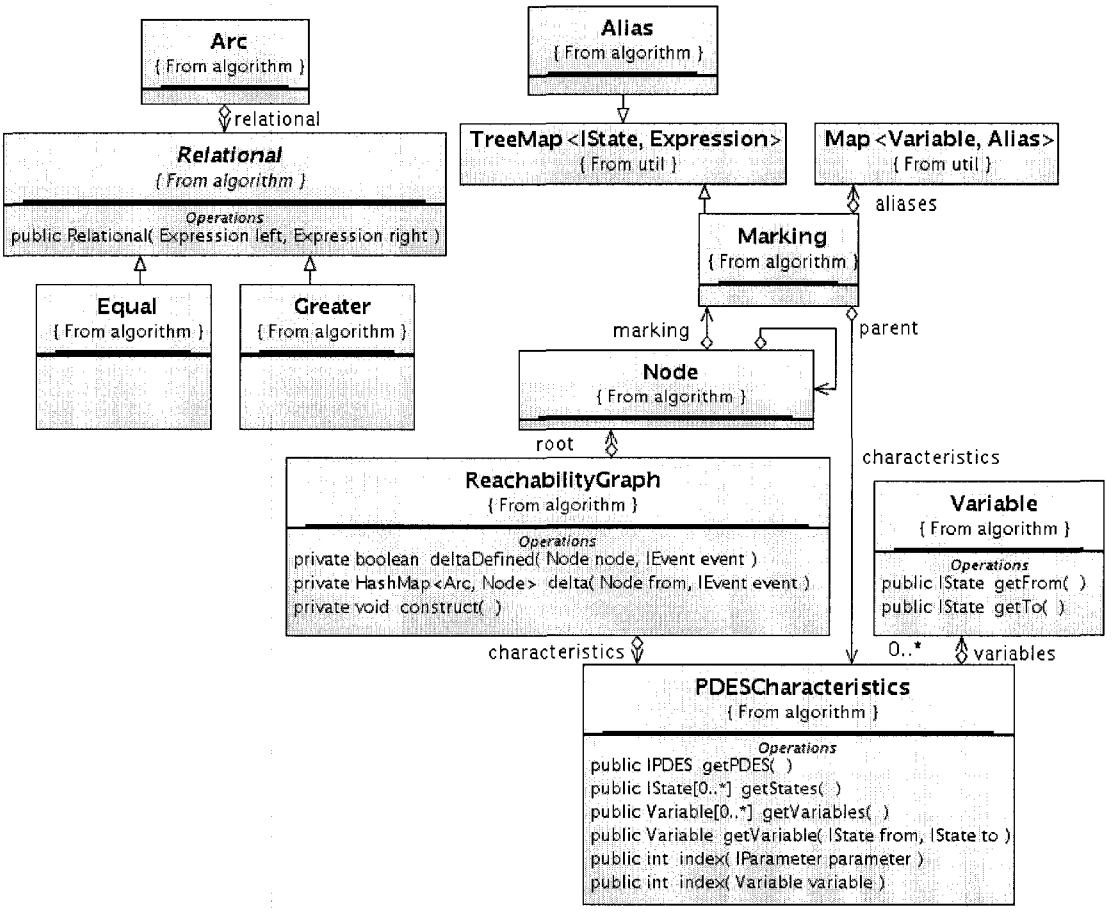


Figure 3.2 – Diagramme de classes de l'algorithme

2. Les procédures $\Delta(m, \sigma)!$, $\Delta(m, \sigma)$ et `Generate_Accessibility_Graph` du chapitre 2 sont implémentées respectivement par la méthode `deltaDefined`, la méthode `delta` et la méthode `construct`.

3.2 Aspects de codification

3.2.1 Implémentation de la notation ensembliste

Au chapitre 2, une notation à l'aide d'ensembles pour les expressions symboliques a été introduite. Pour l'implémenter, des *tableaux de bits* sont utilisés. La classe Java correspon-

dante utilisée est la classe `BitSet`. Avec cette implémentation, l'appartenance d'un élément (paramètre ou variable de transition) à $\pi_1(m(q))$ ou $\pi_2(m(q))$ est signifiée par la position à 1 ou 0 d'un bit désigné. La classe `Expression` de la figure 3.3 implémente les différentes façons d'accéder aux parties d'un Q-marquage.

La figure 3.4 illustre l'idée derrière l'utilisation des tableaux de bits. Les expressions e^+ et e^- désignent les tableaux de bits respectivement de la partie positive et de la partie négative de l'expression symbolique e.

3.2.2 Implémentation des invariants

Les propriétés exprimées par les invariants I1 à I9 s'expriment de façon convenable avec les opérations sur les tableaux de bits. Voici les expressions pour les invariants I2, I3 et I6.

– Invariant I2 :

$$OR_{q \in Q}\ m(q)_P \neq \vec{0};$$

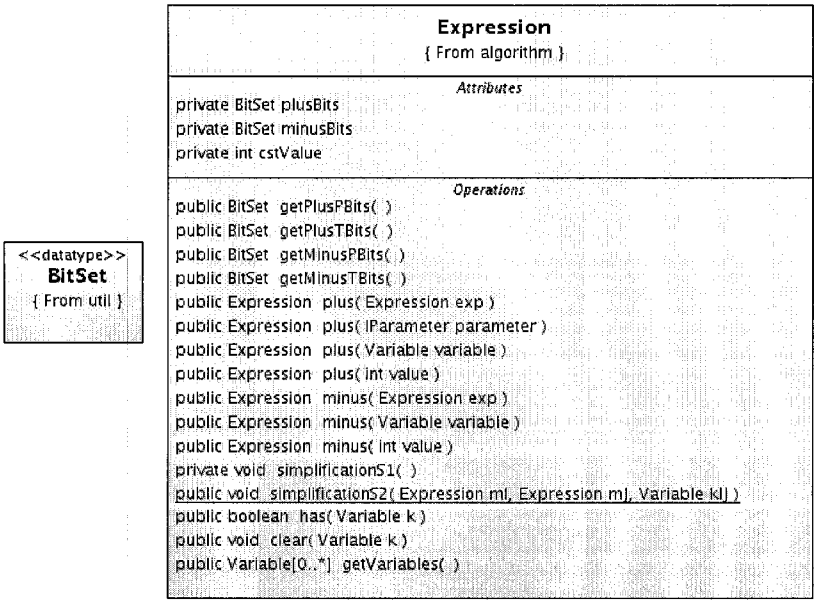


Figure 3.3 – Diagramme de classes des expressions symboliques

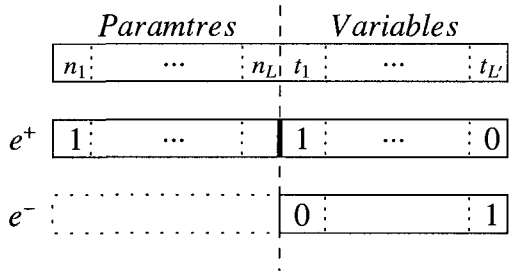


Figure 3.4 – Représentation d'une expression

- Invariant I3 :

$$m(q_1)_T^+ \; AND \; m(q_2)_T^+ = \vec{0} \text{ pour tout } q_1, q_2 \in Q \text{ tels que } q_1 \neq q_2 \; ;$$

- Invariant I6 :

$$(\, OR_{q \in Q} \; m(q)_T^+) \; XOR \; (\, OR_{q \in Q} \; m(q)_T^-) = \vec{0}.$$

3.2.3 Implémentation de la mise à jour des alias

Les alias des variables de transition sont mis à jour lors du déroulement de l'algorithme lorsque la variable mise en cause passe d'une position non biaisée ou biaisée à une autre position biaisée. Cette mise à jour s'effectue suivant la procédure `Update_Alias` (voir l'algorithme 3.1). Pour garder l'uniformité avec les Q-marquages, les alias sont implémentés de la même façon c'est-à-dire comme des fonctions des états ou places à valeurs dans les expressions symboliques. L'ajout de l'alias $t_{q,q'}$ à l'ensemble des alias $A_{t_{q_1,q_2}}$ de la variable t_{q_1,q_2} est implémenté comme l'ajout du terme négatif $-t_{q,q'}$ à l'expression $A_{t_{q_1,q_2}}(q)$ et l'ajout du terme positif $t_{q,q'}$ à l'expression $A_{t_{q_1,q_2}}(q')$.

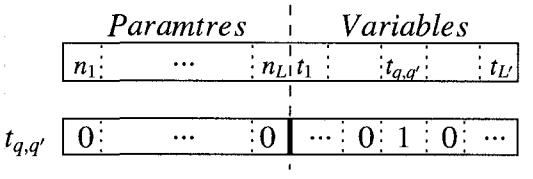


Figure 3.5 – Masque de la variable de transition $t_{q,q'}$

```
1  procedure Update_Alias(
2    in  m,     // Q-marquage predecesseur de m'
3        m',    // Q-marquage dont les alias sont mis a jour
4        t_{q,q'},  // variable de transition devenant un alias
5  )
6  for each t_{q1,q2} ∈ V_T \ {t_{q,q'}} do
7    if (t_{q1,q2} AND (m(q)^+ OR m(q)^-)) ≠ 0⃗ ∧ (t_{q1,q2} AND (m'(q')^+ OR m'(q')^-)) ≠ 0⃗
8      m'.A_{t_{q1,q2}}(q)^- := m'.A_{t_{q1,q2}}(q)^- OR t_{q,q'}
9      m'.A_{t_{q1,q2}}(q')^+ := m'.A_{t_{q1,q2}}(q')^+ OR t_{q,q'}
```

Algorithme 3.1 – Procédure `Update_Alias`

Dans la procédure `Update_Alias`, $t_{q,q'}$ est un masque qui indique la position de la variable de transition correspondante dans les tableaux de bits (voir la figure 3.5).

3.2.4 Implémentation du prédicat $\Delta(m, \sigma)!$

La valeur de retour du prédicat $\Delta(m, \sigma)!$ est donnée par la procédure `Delta_Defined` (voir l'algorithme 3.2).

```
1  procedure Delta_Defined(in m, σ; out Δ!)
2  if σ ∈ Σ_s
3    Δ! := XOR_{q∈Q ∧ δ(q,σ)!} (m(q)^+ OR m(q)^-) = OR_{q∈Q} m_0(q)^+
4          ∧ Σ_{q∈Q ∧ δ(q,σ)!} m(q)_C = 0
5  else  //  σ ∈ Σ
6    Δ! := (∃q | q ∈ Q : δ(q,σ)! ∧ (m(q)^+ ≠ 0⃗ ∨ m(q)_C ≠ 0))
```

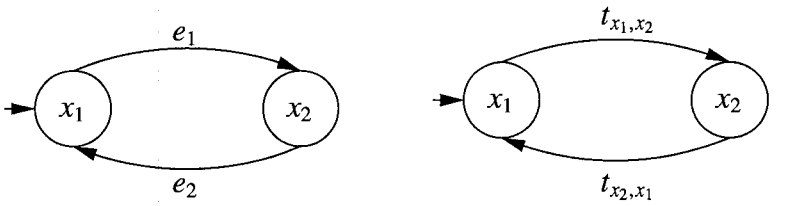
Algorithme 3.2 – Procédure `Delta_Defined`

3.2.5 Implémentation du calcul $\Delta(m, \sigma)$

Les successeurs d'un Q-marquage m pour la transition σ sont calculés par la procédure `Delta` (voir l'algorithme 3.3).

```
1  procedure Delta(in m, σ; out Δ)
2    Δ := ∅
3    if σ ∈ Σ_s
4      for each q' ∈ Q
5        m'(q') := (0⃗, 0⃗, 0)
6        for each q such that δ(q, σ) = q'
7          m'(q')⁺ := m'(q')⁺ OR m(q)⁺;  m'(q')⁻ := m'(q')⁻ OR m(q)⁻
8          m'(q')_C := m'(q')_C + m(q)_C
9      Δ := Δ ∪ {m'}
10   else  //  σ ∈ Σ
11     for each q such that C2 is true
12       q' := δ(q, σ)
13       begin // premier cas : [[m(q)]] > 1
14         m' := m
15         m'(q)⁻ := m'(q)⁻ OR t_{q,q'};  m'(q')⁺ := m'(q')⁺ OR t_{q,q'}
16         if m(q') = (0⃗, 0⃗, 0) begin Apply_Alias(m', q') end
17         Δ := Δ ∪ {m'}
18       end
19       if m(q)⁻ ≠ 0⃗ ∨ |m(q)⁺| = 1 begin // deuxième cas : [[m(q)]] = 1
20         begin
21           m' := m
22           m'(q')⁺ := m'(q')⁺ OR m'(q)⁺;  m'(q')⁻ := m'(q')⁻ OR m'(q)⁻
23           m'(q')_C := m'(q')_C + m'(q)_C
24           m'(q) := (0⃗, 0⃗, 0)
25           Update_Alias(m, m', t_{q,q'})
26           if m(q') = (0⃗, 0⃗, 0) begin Apply_Alias(m', q') end
27           Δ := Δ ∪ {m'}
28         end
29       end
```

Algorithme 3.3 – Procédure `Delta`



(a) Système à deux états S_1 (b) Variables de transition dans S_1

Figure 3.6 – Exemple simple

3.3 Illustration à l'aide d'exemples

L'algorithme implémenté a été exécuté sur plusieurs exemples.

Exemple 1 Considérons le système minimal à deux états S_1 de la figure 3.6a.

Lorsque ce système est mis en entrée de l'algorithme avec comme Q-marquage initial $m_0 = (n, 0)$, où n est un paramètre, le graphe d'accessibilité de la figure 3.7 est produit. Il possède 4 nœuds et 12 arcs. Il a été généré en 56 millisecondes.

Exemple 2 Considérons le système S_2 de la figure 3.8.

Avec le Q-marquage initial $m_0 = (n, 0, 0)$, où n est un paramètre, le graphe d'accessibilité généré par l'algorithme a 15 nœuds et 70 arcs. Il a été généré en 1 178 millisecondes.

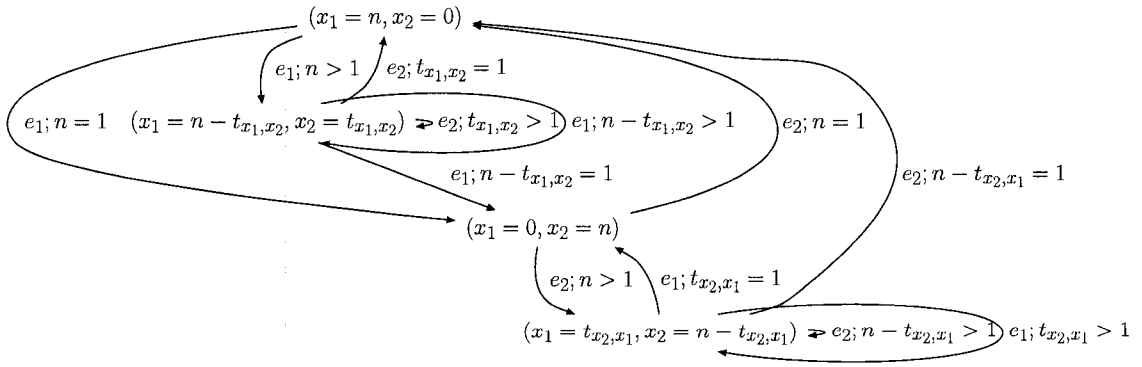


Figure 3.7 – Graphe d'accessibilité du système S_1

Exemple 3 Considérons le système S_3 de la figure 3.9a.

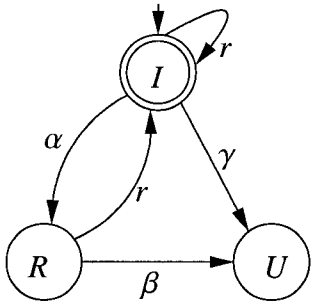
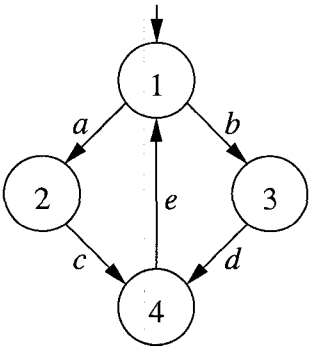
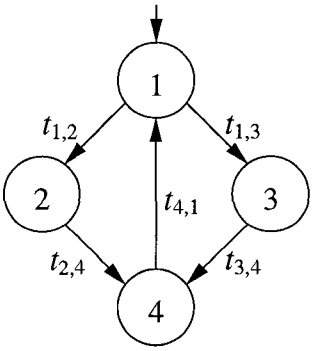


Figure 3.8 – Système à trois états S_2



(a) Système à quatre états S_3



(b) Variables de transition dans S_3

Figure 3.9 – Structure réplicable S_3

Avec le Q-marquage initial $m_0 = (n, 0, 0, 0)$, où n est un paramètre, le graphe d'accessibilité généré par l'algorithme a 80 nœuds et 547 arcs. Il a été généré en 1 276 millisecondes.

Exemple 4 Considérons le système S_4 de la figure 3.10.

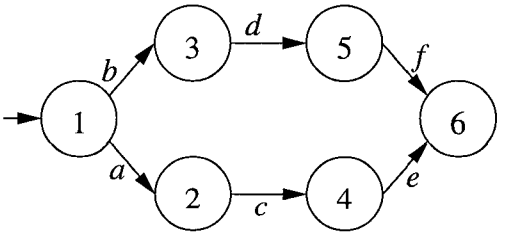


Figure 3.10 – Structure réplicable S_4

Avec le Q-marquage initial $m_0 = (n, 0, 0, 0)$, où n est un paramètre, le graphe d'accessibilité généré par l'algorithme a 169 nœuds et 984 arcs. Il a été généré en 12 983

millisecondes.

Ces résultats sont résumés dans le tableau 3.1.

Système	États	Transitions	Graphe d'accessibilité		
			Nœuds	Arcs	Temps (ms)
S_1	2	2	4	12	56
S_2	3	5	15	70	1 178
S_3	4	5	80	547	1 276
S_4	6	6	169	984	12 983

Tableau 3.1 – Résultats de données tests

Chapitre 4

Intégration dans un algorithme de synthèse

Barbeau et al. décrivent dans [2] un algorithme pour la synthèse hors ligne d'un contrôleur pour un système à événements discrets. Celui-ci peut être aisément adapté pour intégrer l'algorithme de génération d'un graphe d'accessibilité décrit au chapitre 2.

4.1 Description de l'algorithme de synthèse

L'algorithme calcule, non pas à partir du comportement global du système, mais à partir du comportement libre de chacune de ses composantes, un contrôleur optimal respectant une spécification (ou comportement légal). Il a ceci de particulier qu'il calcule pour ce faire, et en même temps, l'espace d'états nécessaire pour générer le contrôleur optimal. Il ne requiert donc pas l'espace d'états complet du système à contrôler. De plus, l'algorithme détecte les *mauvais* états le plus tôt possible et effectue un retour en arrière (*backtracking*) sur les chemins d'événements incontrôlables.

Entre autres hypothèses, pour expliciter cet algorithme, nous supposons que le problème à traiter est de type *NSC (nonblocking supervisory control)*, c'est-à-dire que c'est le système qui détecte les tâches complètes, et nous modifions les éléments issus de [2] pour refléter cette hypothèse de travail.

Le système considéré est un *générateur* $G = (\Sigma, X, \delta, x_0, X_m)$ qui est un automate fini déterministe, où Σ est un ensemble fini d'événements, X un ensemble fini d'états, $\delta : \Sigma \times X \to X$ une fonction de transition, $x_0 \in X$ un état initial et $X_m \subseteq X$ un ensemble d'états marqués. Le *comportement clos* et le *comportement marqué* de G sont définis respectivement par

$$\begin{aligned} L(G) &:= \{w \in \Sigma^* \mid \delta(w, x_0)!\} \text{ et} \\ L_m(G) &:= \{w \in \Sigma^* \mid \delta(w, x_0) \in X_m\}. \end{aligned}$$

L'ensemble X_m représente les *tâches complètes* du système. Au vue du contrôle, l'alphabet Σ est divisé en deux sous-ensembles disjoints : Σ_u, l'ensemble des événements incontrôlables, et Σ_c, l'ensemble des événements contrôlables. Le rôle du contrôle est de gouverner le comportement de G pour qu'il respecte un *langage légal* L défini sur l'alphabet Σ. Soit H l'automate fini déterministe $(\Sigma, Y, \xi, y_0, Y_m)$ tel que $L = L_m(H)$.

Considérons l'intersection de $L(G)$ et $L(H)$, respectivement les comportements clos de G et H. Soit $R = (\Sigma, X \times Y, \gamma, \langle x_0, y_0 \rangle, X_m \times Y_m)$ tel que pour tout $\sigma \in \Sigma$, $x \in X$ et $y \in Y$,

$$\gamma(\sigma, \langle x, y \rangle) = \begin{cases} \langle \delta(\sigma, x), \xi(\sigma, y) \rangle & \text{si } \delta(\sigma, x)! \text{ et } \xi(\sigma, y)! \\ \text{indéfini} & \text{autrement.} \end{cases}$$

On montre aisément que $L(R) = L(H) \cap L(G)$ et $L_m(R) = L \cap L_m(G)$. Un état $\langle x, y \rangle$ de R représente l'évolution de la *boucle de rétroaction* constituée par le système et le contrôleur. Il faut noter que $\delta(\sigma, x)$ peut être défini pour un certain $\sigma \in \Sigma$, tandis que $\xi(\sigma, y)$ n'est pas défini (voir figure 4.1).

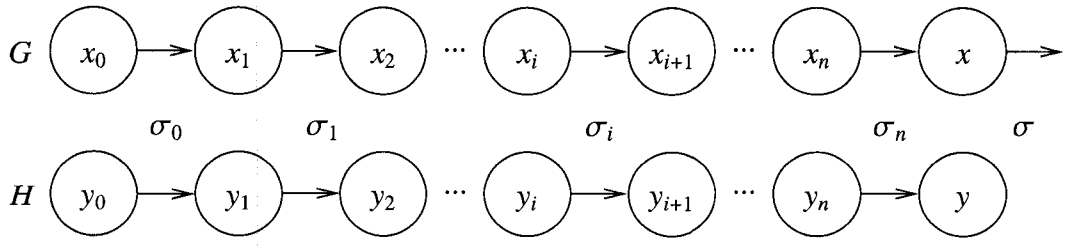


Figure 4.1 – Dernière transition contrôlable de $\langle x, y \rangle$

Dans ce cas particulier, le contrôleur a pour but de désactiver l'événement σ s'il est contrôlable ou d'empêcher le système d'atteindre l'état x en désactivant la dernière transition contrôlable σ_i à partir de la fin de la trace représentée par la figure 4.1.

Soit $s, s' \in X \times Y$. Soit $s.lct$ l'ensemble des dernières transitions contrôlables – *latest controllable transitions* – sur tous les chemins menant à s, c'est-à-dire que l'on peut empêcher la boucle de rétroaction d'atteindre l'état s en désactivant σ' à l'état s' pour tous $[\sigma', s'] \in s.lct$. L'*inactivation* d'une transition à partir de l'état $\langle x, y\rangle$ sur l'événement σ suivant que celui-ci est contrôlable ou pas se formalise ainsi (figure 4.2) :

$$\begin{aligned} &\varphi(\langle x, y\rangle) = \varphi(\langle x, y\rangle) \cup \{\sigma\} && \text{si } \sigma \in \Sigma_c \\ &\varphi(\langle x', y'\rangle) = \varphi(\langle x', y'\rangle) \cup \{\sigma'\} \text{ pour tout } [\sigma', \langle x', y'\rangle] \in \langle x, y\rangle.lct && \text{si } \sigma \in \Sigma_u. \end{aligned}$$

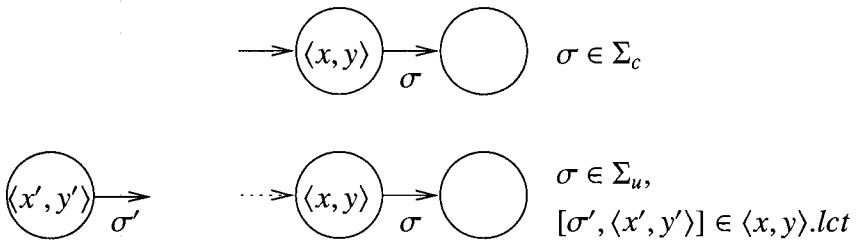


Figure 4.2 – Inactivation de la transition sur l'événement σ

Pour l'implémentation de $s.lct$, deux expressions sont introduites. Soit $s.pre \subseteq \mathcal{P}(X \times Y)$ l'ensemble des états à partir desquels s est directement accessible par un événement incontrôlable. Soit $s.pct \subseteq \mathcal{P}(\Sigma_c \times (X \times Y))$ l'ensemble

$$\{[\sigma, t] \mid \gamma(\sigma, t) = s \wedge \sigma \in \Sigma_c\},$$

c'est-à-dire que s est directement accessible à partir de l'état t sur une transition contrôlable. Si $[\sigma', s'] \in s.lct$ alors il existe une séquence d'états $s_1, s_2, ..., s_n$ avec $n \geq 2$, $s_1 = s'$ et $s_n = s$, tel que $s_{i-1} \in s_i.pre$, $3 \leq i \leq n$, et $[\sigma', s'] \in s_2.pct$.

Soit $s = \langle x, y\rangle \in X \times Y$. Dénotons $s.x$ et $s.y$ respectivement les états $x \in X$ et $y \in Y$. Barbeau et al. utilisent aussi pour leur approche de synthèse les prédicats suivants :

$$Marked(s) = \begin{cases} 1 & \text{si } s.x \in X_m \\ 0 & \text{autrement.} \end{cases}$$

$$Coreachable(s) = \begin{cases} 1 & \text{si } Marked(s) \text{ ou} \\ & \text{si } (\exists w \mid w \in \Sigma^* : \gamma(w, s)! \wedge Marked(\gamma(w, s))) \\ 0 & \text{autrement.} \end{cases}$$

$$Sink(s) = \begin{cases} 1 & \text{si } (\exists \sigma \mid \sigma \in \Sigma_u : \delta(\sigma, s.x)! \wedge \neg\xi(\sigma, s.y)!) \text{ ou} \\ & \text{si } (\exists \sigma \mid \sigma \in \Sigma_u : \delta(\sigma, s.x)! \wedge \xi(\sigma, s.y)! \\ & \qquad \wedge Sink(\langle \delta(\sigma, s.x), \xi(\sigma, s.y) \rangle)) \\ 0 & \text{autrement.} \end{cases}$$

Le premier prédicat (*Marked*) est basé sur la définition de l'intersection de deux langages et est utilisé pour détecter les tâches complètes. Le second prédicat (*Coreachable*) formalise la notion d'état *coaccessible*, c'est-à-dire les états à partir desquels on peut arriver par un chemin (éventuellement vide) à un état marqué. Le troisième prédicat détermine si un état est illégal (*sink*) au vue de la spécification et exprime récursivement la propriété de la contrôlabilité.

Durant la synthèse d'un contrôleur, un indicateur est associé à chaque état visité $s \in X \times Y$, indiquant si s est un état illégal (*sink*), coaccessible (*coreachable*) ou indéfini (*undefined*). L'algorithme de synthèse fait usage des procédures auxilaires suivantes :

- `Mark_State` qui détermine si un état représente une tâche complète ;
- `Initialize_New_State` qui initialise les attributs d'un nouvel état ;
- `Sink` qui fixe l'indicateur d'un état à *sink* (illégal) et détermine aussi parmi ses prédécesseurs les états qui deviennent illégaux ;
- `Coreachable` qui fixe l'indicateur d'un état à *coreachable* (coaccessible) et détermine ceux de ses prédécesseurs qui deviennent coaccessibles ;
- `Undefined` qui vérifie si l'indicateur d'un état peut être défini à *undefined* (indéfini) et détermine parmi ses prédécesseurs ceux qui peuvent potentiellement avoir l'indicateur *indéfini*.

Les algorithmes 4.1, 4.2, 4.3, 4.4 et 4.5 correspondent à ces procédures.

```
1 procedure Mark_State(s)
2   s.marked ← s.x in X_m;
3 end.
```

Algorithme 4.1 – Procédure `Mark_State`

```
1 procedure Initialize_New_state(s)
2   Mark_State(s); s.pct ← {}; s.pre ← {};
3   s.status ← undefined; OPEN ← OPEN ∪ {s};
4 end.
```

Algorithme 4.2 – Procédure `Initialize_New_State`

```
1 procedure Sink(s)
2   s.status ← sink; Delete_Bound_Transitions(s);
3   for each s' in s.pre such that s'.status ≠ sink do Sink(s');
4
5   for each [σ, s'] in s.pct such that s'.marked = false
6                       and s'.status = coreachable do
7     Undefined(s');
8 end.
```

Algorithme 4.3 – Procédure `Sink`

Quand un état s est déclaré *sink*, la procédure `Delete_Bound_Transitions` supprime toutes ses transitions entrantes et sortantes. De ce fait certains des prédecesseurs de s peuvent devenir illégaux ou non coaccessibles ; ceux-là sont alors marqués comme tels. En particulier, la procédure `Undefined` appelée vérifie si les prédecesseurs non marqués et coaccessibles le restent après suppression des transitions.

La procédure de synthèse est donnée par l'algorithme 4.6.

```
1  procedure Coreachable(s)
2    s.status ← coreachable;
3    for each s' in s.pre such that s'.status = undefined do
4      Coreachable(s');
5    for each [σ, s'] in s.pct such that s'.status = undefined do
6      Coreachable(s');
7  end.
```

Algorithme 4.4 – Procédure Coreachable

```
1  procedure Undefined(s)
2    if not Marker_State_Reachable_From(s) then
3      s.status ← undefined;
4    for each s' in s.pre such that s'.marked = false and
5      s'.status = coreachable do Undefined(s');
6    for each [σ, s'] in s.pct such that s'.marked = false and
7      s'.status = coreachable do Undefined(s');
8  end.
```

Algorithme 4.5 – Procédure Undefined

```
1  procedure Derive_Controller(in C_1, ..., C_n, H, Σ_u, out (S, φ))
2    CLOSED ← {}; OPEN ← {}; Initialize_New_State(s_0);
3
4    repeat
5      select s in OPEN;
6      OPEN := OPEN − {s};
7      for each σ in Σ such that δ(σ, s.x)! do
8        x' ← δ(σ, s.x);
9        if not ξ(σ, s.y)! then
10         if σ in Σ_u then {Sink(s); break}
11       else
12         s' ← ⟨x', ξ(σ, s, y)⟩;
13         if s' in CLOSED and s'.status = sink then
14           if σ in Σ_u then {Sink(s); break}
15         else
```

```
16              γ(σ, s) = s' ;
17              if s' not in CLOSED and s' not in OPEN then
18                Initialize_New_State(s');
19              if σ in Σ_u then
20                s'.pre ← s'.pre ∪ {s};
21              else
22                s'.pct ← s'.pct ∪ {[σ, s]};
23              if s.marked = false and s' in CLOSED and
24                s'.status = coreachable then Coreachable(s);
25        if s.marked = true and s.status = undefined then
26          Coreachable(s);
27        CLOSED ← CLOSED ∪ {s};
28      until OPEN = {};
29
30      while exists s in CLOSED such that s.status = undefined then
31        Sink(s);
32      if s_0.status = coreachable then
33        S ← (Σ, CLOSED_reachable, γ, s_0, (CLOSED_m)_reachable);
34      else
35        Error("The solution is empty.");
36      for each s in CLOSED_reachable do
37        for each σ in Σ − Σ_u do
38          if not γ(σ, s)! then φ(s) ← φ(s) ∪ {σ};
39  end.
```

Algorithme 4.6 – Procédure de synthèse

4.2 Spécification et prédicats

Contrairement à l'algorithme de Barbeau et al., pour lequel la spécification est donnée sous la forme d'un langage légal (défini par un automate H), l'intégration de l'algorithme de génération d'un graphe d'accessibilité dans celui de Barbeau et al. exige une spécification donnée sous la forme de prédicats.

Définition 4.1 (Notation). On note $Pred(X)$ l'ensemble des prédicats $\{true, false\}^X$ sur l'ensemble des états X.

Un prédicat $P \in Pred(X)$ représente généralement la spécification à satisfaire. Il peut se présenter sous différentes formes.

Exemples Le prédicat suivant est donné sous la forme de mauvais états :

$$P \Leftrightarrow (\forall i, j \mid 1 \leq i, j \leq 3 \wedge i \neq j : \neg(q[i] = U_i \wedge q[j] = U_j)).$$

Le prédicat suivant est donné sous une forme linéaire :

$$P \Leftrightarrow \#U + 2 \times \#R \leq 10.$$

Dans notre cas, les prédicats sont donnés sous la forme de marquages légaux, c'est-à-dire des marquages pour lesquels le nombre de jetons dans une place donnée est arbitraire ou limité à une valeur fixe comme 0 et 1.

Exemple La figure 4.3 représente un système à trois états I (*Idle*), R (*Request*) et U (*Use*) et dont les événements sont donnés par les ensembles $\Sigma_s = \{r\}$ et $\Sigma = \{\alpha, \beta, \gamma\}$. Les événements contrôlables de ce système sont ceux de l'ensemble $\Sigma_c = \{r, \beta\}$. La condition de l'*exclusion mutuelle* pour les processus suivant ce cycle s'exprime en écrivant le prédicat P sous la forme d'un ensemble de marquages légaux :

$$P = \{(\_, \_, 0), (\_, \_, 1)\}.$$

Dans cette écriture, on remarque l'utilisation du caractère « _ » comme caractère de remplacement (*wildcard*) pour exprimer le fait que les valeurs aux positions concernées ne sont pas restreintes.

Définition 4.2. Un prédicat $P \in Pred(X)$ est dit *non bloquant* pour P si et seulement si

$$(\forall x \mid x \in X \wedge P(x) : (\exists v \mid v \in \Sigma^* : \delta(x, v)! \wedge \delta(x, v) \in X_m \wedge (\forall u \mid u \leq v : P(\delta(x, u))))).$$

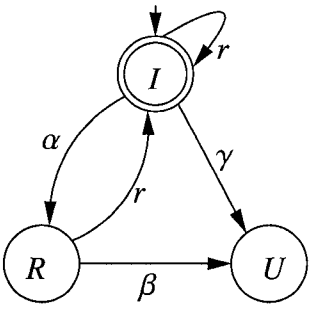


Figure 4.3 – Comportement d'un utilisateur d'une ressource partagée

Définition 4.3. Une fonction SFBC f pour P est *non bloquante* si et seulement si $Re(G|f)$ est un prédicat non bloquant, où $Re(G|f)$ est satisfait pour un état x si ce dernier est accessible depuis x_0 dans G sous le contrôle de f [35].

4.3 Modification de la procédure de synthèse

L'algorithme 4.6 qui effectue la synthèse du contrôleur teste la spécification pour déterminer si un état est mauvais ou pas aux lignes 9 à 10. L'intégration de la génération d'un graphe d'accessibilité doit se faire à ce niveau. Avec le prédicat $\xi(\sigma, s.y)!$, il se produit deux éventualités :

1. la transition conduit à un marquage légal dans le graphe d'accessibilité et, dans ce cas, l'algorithme poursuit sans changement à la ligne 12 ;
2. la transition conduit à un marquage illégal dans le graphe d'accessibilité, c'est-à-dire qu'apparaît dans une place, pour une première fois, un paramètre ou une variable de transition (nécessairement positive) alors que le prédicat P contient le nombre de jetons à 0 ou 1 pour cette place. Il faut alors substituer le paramètre ou la variable de transition par la valeur permise par P et cela pour tous les marquages accessibles par un retour en arrière via l'ensemble *lct* si l'événement concerné est incontrôlable.

Exemple Considérons l'exemple de la figure 4.3 ainsi que la spécification de marquages légaux $P = \{(\_, \_, 0), (\_, \_, 1)\}$. À un point de l'exécution de `Derive_Controller`, l'espace d'états exploré est celui de la figure 4.4.

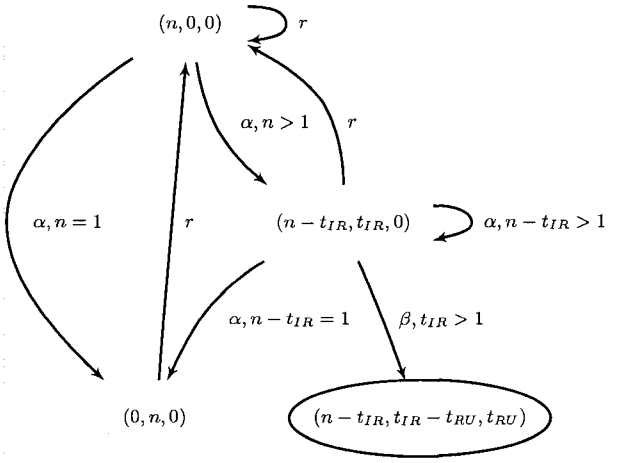


Figure 4.4 – Parcours à la volée de l'espace d'états

Tous les marquages sont acceptables par rapport à P excepté celui entouré d'une ellipse. En effet, posons $m = [n - t_{IR}, t_{IR} - t_{RU}, t_{RU}]$. Pour une valeur arbitraire de $t_{RU} \geq 2$, m ne respecte pas la condition de l'exclusion mutuelle exprimée par la spécification P. Le marquage doit alors être resserré pour que t_{RU} ne prenne que les valeurs permises par P. Ceci fait, on obtient le graphe de la figure 4.5.

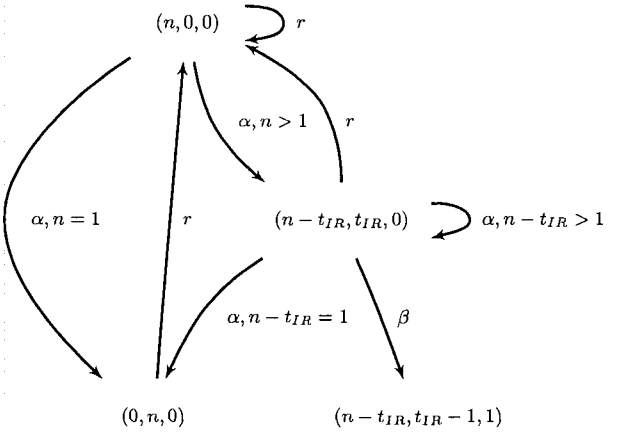
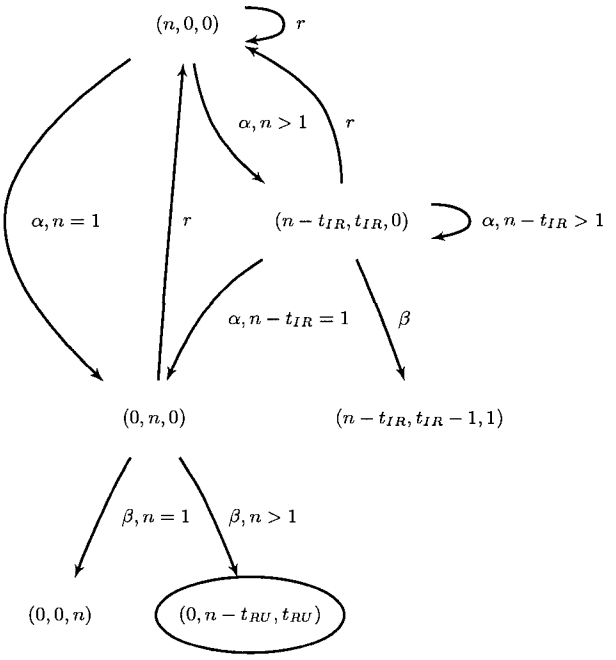
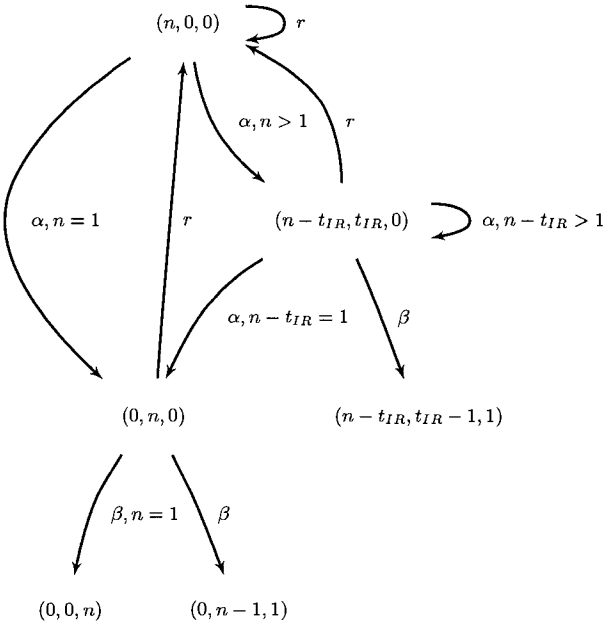


Figure 4.5 – Espace d'états après resserrement

De même en poursuivant l'exécution à partir du marquage $[0, n, 0]$, on obtient le graphe de la figure 4.6a, et après resserrement celui de la figure 4.6b.



(a) Avant resserrement



(b) Après resserrement

Figure 4.6 – Espace d'états à l'itération suivante

Dans le cas où $\Sigma_c = \{\alpha\}$, l'espace d'états obtenus à une certaine itération de l'algorithme est décrit par la figure 4.7.

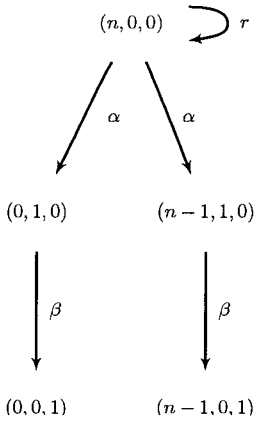


Figure 4.7 – Espace d'états après resserrement pour $\Sigma_c = \{\alpha\}$

Pour parvenir à ce résultat, il faut modifier la définition de la fonction de contrôle φ et adjoindre aux événements (contrôlables) interdits une condition nécessaire ψ (pour le déclenchement de la transition) et une condition suffisante ϕ issue de P. Il n'est alors plus nécessaire d'indiquer pour quel Q-marquage un événement est interdit, puisque c'est la condition nécesaire qui joue ce rôle. De plus les traitements relatifs à φ se font à la volée dans la procédure `Sink`, les lignes 36 à 38 de l'algorithme 4.6 sont alors supprimées. Ainsi,

$$\begin{aligned} &[\sigma, \psi_m, \phi_P] && \text{si } \sigma \in \Sigma_c; \\ &[\sigma', \psi_{m'}, \phi_{P'}] && \text{pour tout } [\sigma', m'] \in m.lct \end{aligned}$$

où ψ_m (respectivement $\psi_{m'}$) est la condition de déclenchement de la transition sur l'événement σ (respectivement σ') à partir de m (respectivement m') et ϕ_P (respectivement $\phi_{P'}$) est la condition issue de P (respectivement P', où P' est obtenu de P lors du retour arrière).

Exemple Relativement à la figure 4.6b de l'exemple précédent, φ correspond à la liste d'un seul triplet :

$$[\beta, m(R) \neq 0, m(U) \geq 1] \text{ pour tout } m \in \mathcal{M}.$$

Si β avait été incontrôlable et α contrôlable, le graphe d'accessibilité aurait été celui de la figure 4.7 et la fonction de rétroaction aurait été donnée par la liste d'un seul triplet :

$$[\alpha, m(I) \neq 0, m(R) \geq 1] \text{ pour tout } m \in \mathcal{M}.$$

Le reste de l'algorithme de synthèse se poursuit sans modification. Les indicateurs des états sont mis à jour aux lignes 24 et 26. La coaccessibilité des Q-marquages à la fin de l'exécution de l'algorithme est donnée par la valeur *coreachable* de leurs indicateurs.

Conclusion

Dans ce mémoire, nous avons amélioré et donné une forme concrète à un algorithme pour la génération d'un graphe d'accessibilité. En particulier, des structures de données concrètes ont été explicitées et une implémentation avec une méthode orientée objet a été donnée. Nous avons fourni une preuve partielle de l'algorithme. Nous avons également illustré sur quelques exemples son fonctionnement.

Nous avons montré comment adapter un algorithme de synthèse de contrôleurs. Cette adaptation permet la synthèse d'un contrôleur dans une forme fermée pour le système à composants décrit par le graphe d'accessibilité. Nous avons fourni une notation pour la spécification d'états légaux utilisant un caractère de remplacement pour décrire des expressions symboliques prenant des valeurs arbitraires. Une représentation pour la fonction de contrôle φ a également été donnée. Cette représentation exprime avec des inégalités simples les conditions d'inhibition des événements contrôlables.

Comme amélioration à ce travail, une preuve formelle et complète de l'algorithme peut être apportée. Une implémentation de l'algorithme de synthèse de contrôleurs intégrant la génération d'un graphe d'accessibilité peut aussi être envisagée. Par ailleurs, les exemples utilisés pour illustrer l'exécution de l'algorithme font usage d'un seul paramètre dans les Q-marquages initiaux et de constantes nulles. Il serait intéressant d'observer le comportement de l'algorithme pour des cas où $|V_P| > 1$, tout comme des cas avec des constantes non nulles à certaines places. Éventuellement certaines parties de l'algorithme seront à adapter pour accomoder le nombre de paramètres et les constantes non nulles. Notamment, une simplification avec les paramètres pour traiter les relations entre paramètres telles que $n = n_1 + n_2$, pourrait être introduite de façon formelle et prouvée. Cette simplification apporterait une meilleure lisibilité des Q-marquages dans le graphe d'accessibilité.

La spécification P de l'exclusion mutuelle évoquée dans ce mémoire limite le nombre de processus dans un certain état à 0 ou 1. Une spécification avec des *valeurs limites* plus grandes pourrait être envisagée. Le resserement effectué sur les Q-marquages qui violent la spécification doit alors être fait en plusieurs étapes, jusqu'à ce que la valeur limite permise soit atteinte.

Annexe A

Exemple d'un processus BPEL

```
1  <process name="purchaseOrderProcess"
2  targetNamespace="http://example.com/ws-bp/purchase"
3  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
4  xmlns:lns="http://manufacturing.org/wsdl/purchase">
5    <documentation xml:lang="EN">
6    A simple example of a WS-BPEL process for handling a purchase
7    order.
8    </documentation>
9
10   <partnerLinks>
11     <partnerLink
12       name="purchasing"
13       partnerLinkType="lns:purchasingLT"
14       myRole="purchaseService" />
15     <partnerLink
16       name="invoicing"
17       partnerLinkType="lns:invoicingLT"
18       myRole="invoiceRequester"
19       partnerRole="invoiceService" />
20     <partnerLink
21       name="shipping"
22       partnerLinkType="lns:shippingLT"
```

```
23        myRole="shippingRequester"
24        partnerRole="shippingService" />
25      <partnerLink
26        name="scheduling"
27        partnerLinkType="lns:schedulingLT"
28        partnerRole="schedulingService" />
29    </partnerLinks>
30
31    <variables>
32      <variable name="PO" messageType="lns:POMessage" />
33      <variable name="Invoice" messageType="lns:InvMessage" />
34      <variable
35        name="shippingRequest"
36        messageType="lns:shippingRequestMessage" />
37      <variable
38        name="shippingInfo"
39        messageType="lns:shippingInfoMessage" />
40      <variable
41        name="shippingSchedule"
42        messageType="lns:scheduleMessage" />
43    </variables>
44
45    <faultHandlers>
46      <catch
47        faultName="lns:cannotCompleteOrder"
48        faultVariable="POFault"
49        faultMessageType="lns:orderFaultType">
50        <reply
51          partnerLink="purchasing"
52          portType="lns:purchaseOrderPT"
53          operation="sendPurchaseOrder"
54          variable="POFault"
55          faultName="cannotCompleteOrder" />
56      </catch>
```

```
57    </faultHandlers>
58
59    <sequence>
60      <receive
61        partnerLink="purchasing"
62        portType="lns:purchaseOrderPT"
63        operation="sendPurchaseOrder"
64        variable="PO"
65        createInstance="yes">
66        <documentation>Receive Purchase Order</documentation>
67      </receive>
68      <flow>
69        <documentation>
70          A parallel flow to handle shipping, invoicing and
71          scheduling
72        </documentation>
73        <links>
74          <link name="ship-to-invoice" />
75          <link name="ship-to-scheduling" />
76        </links>
77
78        <sequence>
79          <assign>
80            <copy>
81              <from>$PO.customerInfo</from>
82              <to>$shippingRequest.customerInfo</to>
83            </copy>
84          </assign>
85          <invoke partnerLink="shipping"
86          portType="lns:shippingPT"
87          operation="requestShipping"
88          inputVariable="shippingRequest"
89          outputVariable="shippingInfo">
90            <documentation>Decide On Shipper</documentation>
```

```
91          <sources>
92            <source linkName="ship-to-invoice" />
93          </sources>
94        </invoke>
95        <receive
96          partnerLink="shipping"
97          portType="lns:shippingCallbackPT"
98          operation="sendSchedule" variable="shippingSchedule">
99          <documentation>Arrange Logistics</documentation>
100         <sources>
101           <source linkName="ship-to-scheduling" />
102         </sources>
103       </receive>
104     </sequence>
105
106     <sequence>
107       <invoke
108         partnerLink="invoicing" portType="lns:computePricePT"
109         operation="initiatePriceCalculation" inputVariable="PO">
110         <documentation>Initial Price Calculation</documentation>
111       </invoke>
112       <invoke
113         partnerLink="invoicing"
114         portType="lns:computePricePT"
115         operation="sendShippingPrice"
116         inputVariable="shippingInfo">
117         <documentation>Complete Price Calculation</documentation>
118         <targets>
119           <target linkName="ship-to-invoice" />
120         </targets>
121       </invoke>
122       <receive
123         partnerLink="invoicing"
124         rolesportType="lns:invoiceCallbackPT"
```

```
125           operation="sendInvoice" variable="Invoice" />
126       </sequence>
127
128       <sequence>
129         <invoke
130           partnerLink="scheduling"
131           portType="lns:schedulingPT"
132           operation="requestProductionScheduling"
133           inputVariable="PO">
134           <documentation>
135             Initiate Production Scheduling
136           </documentation>
137         </invoke>
138         <invoke
139           partnerLink="scheduling"
140           portType="lns:schedulingPT"
141           operation="sendShippingSchedule"
142           inputVariable="shippingSchedule">
143           <documentation>
144             Complete Production Scheduling
145           </documentation>
146           <targets>
147             <target linkName="ship-to-scheduling" />
148           </targets>
149         </invoke>
150       </sequence>
151     </flow>
152     <reply
153       partnerLink="purchasing"
154       portType="lns:purchaseOrderPT"
155       operation="sendPurchaseOrder"
156       variable="Invoice">
157       <documentation>Invoice Processing</documentation>
158     </reply>
```

```
159     </sequence>
160   </process>
```

Bibliographie

[1] W. M. P. van der AALST. « The application of Petri nets to workflow management ». The Journal of Circuits, Systems and Computers, 8(1) :21–66, 1998.

[2] M. BARBEAU, F. KABANZA et R. ST-DENIS. « An efficient algorithm for controller synthesis under full observation ». Journal of Algorithms, 25(1) :144–161, 1997.

[3] L. ben OTHMANE. « Développement d'un système de gestion de workflows distribué ». Mémoire de maîtrise, Université de Sherbrooke, 2000.

[4] H. BHERER, J. DESHARNAIS et R. ST-DENIS. « On the reachability and nonblocking properties for parameterized discrete event systems ». Dans Proceedings of the 8th International Workshop on Discrete Event Systems, pages 113–118, Ann Arbor, MI, 2006.

[5] H. BHERER, J. DESHARNAIS et R. ST-DENIS. « Control of parameterized discrete event systems ». Discrete Event Dynamic Systems : Theory and Applications, À paraître.

[6] P. BRUCKER. Scheduling algorithms. Springer, 2007.

[7] P. BRUCKER et S. KNUST. Complex scheduling. Springer, 2006.

[8] C. G. CASSANDRAS et S. LAFORTUNE. Introduction to discrete event systems. Kluwer Academic Press, 1999.

[9] J. CHEN et Y. YANG. « Key research issues in grid workflow verification and validation ». Dans ACSW Frontiers '06 : Proceedings of the 2006 Australasian Workshops on Grid Computing and e-Research, pages 97–104, Hobart, Australia, 2006.

[10] H. DAVULCU, M. KIFER, C. R. RAMAKRISHNAN et I. V. RAMAKRISHNAN. « Logic based modeling and analysis of workflows ». Dans PODS '98 : Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pages 25–33, New York, NY, 1998.

[11] E. GAMMA, R. HELM, R. JOHNSON et J. VLISSIDES. Design patterns : elements of reusable object-oriented software. Addison-Wesley, 1995.

[12] M. R. GAREY et D. S. JOHNSON. Computers and intractability ; A Guide to the theory of NP-completeness. W. H. Freeman & Co., 1990.

[13] R. E. GOMORY. Solving linear programming problems in integers. American Mathematical Society, 1960.

[14] X. HE et T. MURATA. « The electrical engineering handbook », Chapitre High-level Petri nets - Extension, analysis and applications, pages 459–475. Elsevier Academic Press, 2005.

[15] L. E. HOLLOWAY, B. H. KROGH et A. GIUA. « A survey of Petri net methods for controlled discrete event systems ». Discrete Event Dynamic Systems : Theory and Applications, 7(2) :151–190, 1997.

[16] S. JABLONSKI et C. BUSSLER. Workflow management – modeling concepts, architecture and implementation. International Thomson Publishing, 1996.

[17] R. KUMAR et V. K. GARG. Modeling and control of logical discrete event systems. Kluwer, 1995.

[18] S. LABYAD. « Contrôle des processus d'affaires ». Mémoire de maîtrise, Université de Sherbrooke, 1998.

[19] Y. LI et W. M. WONHAM. « Control of vector discrete-event systems II – controller synthesis ». IEEE Transactions on Automatic Control, 39(3) :512–531, 1994.

[20] S. MUKHERJEE. « Logic-based approaches to workflow modeling and verification », Chapitre 5, pages 167–202. Springer, 2004.

[21] T. MURATA. « Petri nets : properties, analysis and applications ». Proceedings of the IEEE, 77(4) :541–580, 1989.

[22] OASIS. Web Services Business Process Execution Language Version 2.0. 2007.

[23] E. OREN et A. HALLER. « Formal frameworks for workflow modeling ». Rapport technique, DERI - Digital Enterprise Research Institute, 2005.

[24] C. A. PETRI. « Kommunikation mit automaten ». Thèse de doctorat, Institut für instrumentelle Mathematik, 1962.

[25] S. A. REVELIOTIS. Real-time management of resource allocation systems. A discrete event systems approach. Springer, 2005.

[26] D. RIEHLE et H. ZÜLLIGHOVEN. « Understanding and using patterns in software development ». Theory and Practice of Object Systems, 2(1) :3–13, 1996.

[27] N. RUSSELL, A. H. M. ter HOFSTEDE et W. M. P. van der AALST. « Exception handling patterns in process-aware information systems ». Rapport technique, BPM Center Report BPM-06-04, BPMcenter.org, 2006.

[28] N. RUSSELL, A. H. M. ter HOFSTEDE, W. M. P. van der AALST et D. EDMOND. « Workflow data patterns ». Rapport technique, QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.

[29] N. RUSSELL, A. H. M. ter HOFSTEDE, W. M. P. van der AALST et D. EDMOND. « Workflow resource patterns ». Rapport technique, BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

[30] N. RUSSELL, A. H. M. ter HOFSTEDE, W. M. P. van der AALST et N. MULYAR. « Workflow control-flow patterns : a revised view ». Rapport technique, BPM Center Report BPM-06-22, BPMcenter.org, 2006.

[31] C. SCHWINDT. Resource allocation in project management. Springer, 2005.

[32] W. M. P. van der AALST. « Verification of workflow nets ». Dans ICATPN'97 : Proceedings of the 18th International Conference on Application and Theory of Petri Nets, pages 407–426, Toulouse, France, 1997. Springer-Verlag.

[33] W. M. P. van der AALST, A. H. M. ter HOFSTEDE, B. KIEPUSZEWSKI et A. P. BARROS. « Workflow patterns ». Distributed Parallel Databases, 14(1) :5–51, 2003.

[34] WFMC. « Workflow management coalition terminology & glossary ». Rapport technique, Workflow Management Coalition, Document Number WFMC-TC-1011, 1999.

[35] W. M. WONHAM. « Supervisory control of discrete-event systems ». Rapport technique, ECE 1636F/1637S, System Control Group, University of Toronto, 2006.