Table des matières

Introduction générale

Ce manuscrit présente mes travaux de thèse, financés par le dispositif de Conventions Industrielles de Formation par la REcherche (CIFRE). Elle a été réalisée au sein de l'entreprise Thales Avionics en collaboration avec l'équipe VERTICS du LAAS-CNRS. Le référent industriel est Eric Jenn (Thales avionics) et les directeurs de thèses sont Bernard Berthomieu(LAAS-CNRS) et François Vernadat(LAAS-CNRS).

Thales Avionics S.A. est une société œuvrant dans le domaine des systèmes avioniques. Elle compte plus de 6000 salariés qui permettent à la société de fournir des systèmes complets d'électronique de vol. Thales Avionics est un fournisseur important des plus grand avionneurs mondiaux, Airbus, Boeing, Sukhoi, ...

L'équipe VERTICS, pour Vérification de Systèmes Temporisés Critiques, travaille autour des techniques de description formelle des systèmes informatiques critiques. Elle est spécialisée dans les modèles temporisés et notamment la vérification par model-checking des réseaux de Petri temporels (TPN pour "Time Petri Net").

Mes travaux se situent aux croisements de ces thématiques : ils portent sur la vérification formelle par model-checking dans le contexte des systèmes avioniques. La thèse est organisée en deux parties. La première traite des spécificités avioniques et présente un exemple de vérification par model-checking sur une fonction avionique, la deuxième partie propose une méthode de réduction par symétries pour les TPN.

La première partie se consacre aux systèmes avioniques qui sont les systèmes électroniques, électriques et informatiques embarqués dans un avion et indispensables à son fonctionnement. Nous ne nous intéressons qu'aux systèmes informatiques qui sont temps réel et critiques. Ces systèmes doivent garantir un très haut niveau de sûreté de fonctionnement. Cette exigence les soumet à de fortes contraintes règlementaires qui en structurent la réalisation. En conséquence, l'utilisation des méthodes formelles s'inscrit dans des processus industriels guidés par un objectif élevé de sûreté de fonctionnement.

Le premier chapitre présente les outils théoriques et pratiques utilisés dans cette thèse pour la vérification par model-checking. L'objectif du model-checking est la vérification de propriétés d'un système. Ces propriétés sont généralement spécifiées par des formules logiques. En particulier, les logiques temporelles, linéaires ou à branchements, permettent d'exprimer des propriétés sur les états/transitions passés, présents ou futurs qu'un système peut/doit atteindre. Le comportement du système est formellement modélisé, via des automates, réseaux de Petri, algèbres de processus, ... Nous nous concentrons sur les modèles temporisés, les automates temporisés et les réseaux de Petri temporels. La principale limite à l'utilisation du model-checking est liée au problème de l'*explosion combinatoire* : le nombre d'états augmente de façon exponentielle en fonction de la complexité du système. Il existe plusieurs techniques de lutte contre l'explosion. Nous donnons un panorama des techniques référencées dans cette thèse. Les expérimentations réalisées l'ont été grâçe au langage FIACRE et à la boite à outils TINA qui sont finalement présentés dans ce chapitre.

Le second chapitre présente comment les méthodes de vérification formelles, et en particulier le model-checking, peuvent être utilisées pour la vérification des systèmes avioniques.

Après un rappel sur les contraintes règlementaires, liées à la problématique de certification et structurantes pour les systèmes avioniques, il donne des exemples de cas d'utilisation tirés de la littérature. Sans donner de règles générales, ces exemples dessinent un cadre dans lequel l'utilisation des méthodes formelles optimise les activités de vérification d'un système avionique.

Nous présentons dans ce chapitre une application de FIACRE/TINA à la modélisation et vérification formelle par model-checking de contraintes temporelles dans un contexte avionique. Précisément, nous vérifions dans cette étude de cas des propriétés non fonctionnelles qui émergent de la nature localement synchrone et globalement asynchrone des architectures avioniques. Cet exemple de modélisation et vérification formelle est la première contribution de cette thèse. Il a fait l'objet d'une publication lors de la 25-ième édition du symposium IEEE "Software Reliability Engineering Workshops (ISSREW)" sous le titre :

> *Model-Checking Real-Time Properties of an Auto Flight Control System Function* [1]

Cette contribution montre qu'une des armes contre l'explosion combinatoire est la modélisation. Le choix des abstractions dans la modélisation a un fort impact sur la taille de l'espace d'états. Lorsqu'elles sont bien choisies, les abstractions permettent de vérifier les propriétés effectivement.

Le reste de la thèse se concentre sur une méthode de lutte contre l'explosion combinatoire qui exploite les symétries structurelles. C'est l'objet de la deuxième partie et la principale contribution de cette thèse.

La deuxième partie propose une méthode de réduction par symétrie pour les TPN. Le concept de symétrie est utilisé dans divers domaines scientifiques ou artistiques. Ce concept permet d'étudier les propriétés d'un objet modulo certaines transformations qui laissent invariantes ces propriétés. Nous reprenons l'idée que les symétries peuvent être utilisées pour la réduction des espaces d'états. Les symétries sont formalisées en mathématique par la théorie des groupes. Le premier chapitre donne les éléments de théorie des groupes nécessaires à la compréhension de cette thèse. Nous utilisons dans ce manuscrit les notions de groupe de permutations et d'actions de groupe. Ces notions donnent un cadre formel abstrait dans lequel s'inscrit notre méthode de construction des réseaux symétriques. L'action d'un groupe sur un ensemble est un concept puissant très utilisé en théorie des groupes. En particulier, les actions de groupes permettent de faire le lien entre la notion géométrique d'orbite et celle plus algébrique de stabilisateur. Pour ce qui nous concerne, les actions permettent de calculer les symétries des réseaux de Petri et sont fondamentales pour l'exploitation des symétries pour la construction d'espace d'états réduit.

Le deuxième chapitre est un état de l'art sur les méthodes de réduction par symétrie dans le model-checking. La méthode de réduction par symétrie exploite les symétries d'un système pour minimiser le nombre d'états à explorer. Cette méthode construit un quotient de l'espace d'états par la relation d'équivalence induite par les symétries du système. Lorsque les applications ont des structures symétriques, ce qui est généralement le cas pour des applications de taille importante, la réduction par symétrie peut contribuer à la réduction de l'explosion combinatoire. Le facteur de réduction est proportionnel à la taille du groupe lorsque le système est totalement symétrique.

Deux problèmes se posent alors pour l'utilisation des symétries. Le premier est d'identifier les symétries d'un système, soit automatiquement, soit à partir d'annotations du modèle. Le deuxième problème est l'exploitation de ces symétries pour la réduction de l'espace d'états. Dans ce chapitre nous présentons les méthodes existantes pour les modèles non temporisés et temporisés.

Le troisième chapitre décrit la deuxième contribution de cette thèse. Nous proposons

une nouvelle méthode pour la construction de réseaux de Petri avec symétries. Nous définissons un opérateur de composition symétrique qui étant donné un groupe G de permutations, k réseaux de Petri composants et des spécifications de synchronisation est capable de construire un réseau dont le groupe de symétries est isomorphe à G, sans contrainte sur les synchronisations. Cette dernière propriété démarque notre approche des constructions existantes basées sur les annotations, où des contraintes sur les synchronisations restreignent la classe des réseaux que l'on peut construire. Notre construction est précise dans le sens où la structure du groupe calculée est connue ce qui permet de choisir une stratégie de réduction adaptée. Elle est générale car elle supporte des groupes de permutations et des synchronisations arbitraires. Elle est plus efficace que les méthodes de détection automatiques mais moins que les méthodes par annotation.

Cet opérateur de composition fait face à quatre problèmes principaux. Le premier est la taille exponentielle des groupes. L'opérateur est capable de calculer un groupe de symétries de N isomorphe à G à partir de l'ensemble générateur de G. La taille d'un ensemble générateur est bornée par $\log |G|$, mais en pratique une dizaine de générateurs est souvent suffisant. Le deuxième problème est lié à l'aspect compositionnel de la construction. Les k composants utilisés peuvent être localement symétriques. Il n'est pas possible d'ignorer ces symétries. Notre opérateur de composition permet de "lier" ces groupes de symétries par la définition d'action de sous-groupe de G sur les composants. Le troisième problème est du aux synchronisations. Les synchronisations modifient la structure du réseau et ces modifications peuvent briser les symétries. Notre opérateur de composition est capable de calculer l'ensemble des synchronisations nécessaires pour que N soit symétrique à partir d'un échantillon de synchronisations. Le dernier problème est lié à la structure des systèmes que l'on souhaite modéliser. Ces systèmes ne sont pas en général totalement symétriques. L'opérateur de composition offre une granularité suffisamment fine pour permettre de construire des réseaux symétriques dont certains sous réseaux n'ont pas de symétrie.

Le quatrième chapitre donne la troisième contribution de cette thèse. Il s'agit d'une méthode qui permet d'exploiter les symétries pour construire le quotient d'un graphe des classes d'un TPN. Nous montrons que les symétries d'un TPN N induisent des symétries sur le graphe d'états de N mais aussi des symétries sur le graphe des classes de N. Ce résultat prouve l'existence d'une relation d'équivalence sur les classes du graphe des classes de N. Après avoir remarqué qu'aucune méthode de réduction n'existait pour les TPN, nous discutons de l'applicabilité des méthodes de réduction existantes pour les modèles non temporisés pour constater que la plupart de ces méthodes, en tout cas les méthodes efficaces, ne s'appliquent pas aux TPN.

Cela s'explique par la dimension temporelle des TPN, capturée par un domaine de tir, c'est à dire un ensemble de contraintes sur les dates de tir des transitions. Ce domaine de tir est généralement implémenté par une matrice de différences indicée par des transitions sensibilisées. En conséquence, en considérant l'état global comme un vecteur (ie: en "linéarisant" la matrice), chaque transition contribue par plusieurs indices dans ce vecteur. Cela invalide l'hypothèse faite par les méthodes pour les modèles non temporisés où il y a une bijection entre les éléments permutés du modèle et les indices du vecteur d'états. Par exemple, dans le cas des réseaux de Petri, l'état est un vecteur de marquages indicé par les places.

La méthode que nous proposons repose sur la définition d'un ordre total sur les transitions de même intervalle statique et sensibilisées par un marquage. Comme les symétries préservent les intervalles statiques, nous pouvons ordonner les transitions équivalentes par symétrie. Cet ordre total exploite un invariant sur les différences entre les transitions dans l'algorithme de calcul incrémental des classes d'états. Il est alors possible d'abstraire l'information temporelle associée à ces transitions par leur position dans l'ordre ce qui rend applicable les méthodes des modèles non temporisés. Parmi les méthodes disponibles nous implémentons celle dite du calcul du représentant canonique. Le calcul du repré-

sentant canonique suppose l'existence d'un ordre total sur les classes équivalentes. Nous montrons l'existence de cet ordre qui se déduit de celui sur les transitions équivalentes. Finalement, nous définissons une classe de réseaux, dite classe des réseaux à composants disjoints (RCD), dans laquelle le calcul du représentant canonique est applicable. Cette définition généralise des résultats existants dans le contexte du model-checking non temporisé et des automates temporisés. Nous donnons des conditions suffisantes sur l'opérateur de composition symétrique qui permettent d'obtenir des réseaux à composants disjoints.

Les principaux résultats de ce chapitre ont fait l'objet d'une publication lors de la 30-ième édition du symposium ACM "Symposium on Applied Computing" sous le titre :

Symmetry reduced state classes for time petri nets [2]

Le cinquième et dernier chapitre présente l'implémentation d'un prototype pour l'opérateur de composition symétrique avec lequel ont été construits tous les réseaux illustrés dans cette thèse. Il présente aussi une première version d'une implémentation dans TINA de la méthode de réduction. Des résultats expérimentaux encourageants sont présentés.

Première partie

Model-checking dans un contexte avionique

Chapitre 1

Vérification formelle par model-checking

Sommaire

Le model-checking [3, 4] est une approche automatisée permettant de vérifier qu'un modèle de système est conforme à ses spécifications. Le comportement du système est formellement modélisé, via des automates, réseaux de Petri, algèbres de processus, . . . et les spécifications, exprimant les propriétés attendues du système, sont formellement exprimées par exemple via des formules de logiques temporelles.

En pratique, les propriétés du système sont souvent classées en deux grandes catégories informelles. Les propriétés de *sûreté* énonce qu'une situation particulière ne peut être atteinte. Les propriétés de *vivacité* énonce quelque chose de mauvais (ou de bon) qui finira par se produire [5].

Le model-checking permet de décider si une structure M satisfait une propriété ϕ, exprimée comme une formule logique. La satisfaction de ϕ par M est notée $M \models \phi$. Par extension, le système satisfait ses spécifications si le modèle formel de son "comportement" satisfait les formules logiques exprimant les propriétés attendues. Suivant les types de logiques considérés : le comportement correspond à toutes les séquences issues de l'état initial (logiques linéaires) ou au graphe des états accessibles (logiques arborescentes).

La Figure 1.1, tirée de [6], donne un aperçu simplifié du cycle d'utilisation du model-checking. Le cycle peut se diviser en trois phases :

1. Modélisation formelle du comportement du système
2. Expression formelle des propriétés attendues
3. Si une propriété n'est pas satisfaite, un contre-exemple est produit qui décrit un scénario possible d'erreur (i.e de violation de la propriété).
   L'analyse de celui-ci aide à apporter les corrections nécessaires que ce soit sur la modélisation du comportement du système ou sur l'expression formelle des propriétés attendues.

Ce cycle est répété jusqu'à ce que toutes les formules, c'est à dire toutes les spécifications, soient vérifiées.

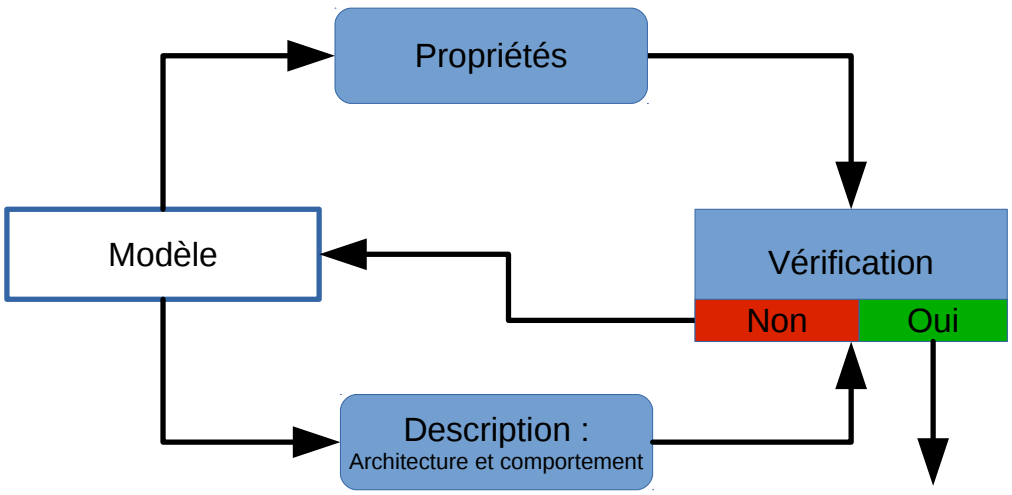


FIGURE 1.1: Cycle d'utilisation du model-checking

La figure précédente est volontairement abstraite. Nous verrons, un peu plus loin, que les algorithmes de vérification ne procèdent pas directement sur la modélisation (implicite)

du comportement (via réseaux de Petri, Automates, ...) mais nécessite une représentation explicite - y compris symbolique et/ou partielle - de celui-ci à travers son graphe d'états qui sera encodé sous forme de structure de Kripke. La nécessité d'expliciter le comportement du système sous forme de structure de Kripke dont la taille explose combinatoirement limite fortement l'applicabilité de ces méthodes.

Nous verrons que, suivant le cas, il n'est pas toujours indispensable de construire la totalité de cet espace d'états et que la vérification de la propriété peut être menée de front avec la construction de l'espace à vérifier ; on parle alors de vérification à la volée.

Le cas des systèmes temporels est singulier puisque leurs espaces d'états sont généralement infinis. Afin de pouvoir leur appliquer les techniques de vérification exhaustive - type model-checking - il est nécessaire de disposer d'abstractions finies de leurs espaces d'états. Les abstractions les plus connues pour les automates temporisés et pour les réseaux de Petri sont présentées dans les sections 1.2.2 et 1.2.3, respectivement .

La section 1.3 présente quelques pistes pour contourner les problèmes d'explosion combinatoire.

1.1 Logiques temporelles

Les logiques temporelles permettent d'exprimer des propriétés sur les états/transitions passés, présents ou futurs qu'un système peut/doit atteindre. A cet effet, elles offrent des opérateurs temporels spécifiques, des modalités, tels que *finalement* ou *jamais*, qui permettent de décrire des ordres entre les évènements/états sans pour autant introduire le temps explicitement. Pour ces logiques, le temps est une donnée abstraite qui n'est pas quantifiée par opposition aux logiques temporelles temporisées qui permettent de manipuler explicitement le temps.

D'un point de vue plus théorique, on peut distinguer deux grandes familles de logiques temporelles. Ces deux familles, arborescentes et linéaires, sont distinguées par la nature de la relation d'ordre associée au temps que l'on considère. Cet ordre capture l'antériorité temporelle entre les états/événements et :

— Si l'ordre est total, le comportement du système est vu comme un ensemble de séquences d'exécution et la spécification se fait via les logiques temporelles linéaires. La logique LTL (Linear time Logic) 1.1.2 constitue un exemple de ces logiques.

— Si l'ordre est partiel, le comportement du système est vu comme un graphe et la spécification se fait via les logiques temporelles arborescentes. la logique CTL (Conditional/Computer Tree Logic) 1.1.3 constitue un exemple de ces logiques.

Les pouvoirs d'expression des logiques arborescentes et linéaires sont incomparables. Les propriétés de sûreté et de vivacité sont expressibles tant en LTL qu'en CTL. Les propriétés d'équité, qui énoncent que sous certaines conditions quelque chose aura lieu un nombre infini de fois, ne sont expressibles qu'en LTL. Les propriétés de potentialité, qui permette par exemple d'énoncer le fait d'être réinitialisable, ne sont expressibles qu'en CTL. Sans vouloir être exhaustif, mentionnons la logique CTL* qui offre le pouvoir d'expression de LTL et de CTL.

1.1.1 Définitions

Structure de Kripke

La sémantique des formules de logiques temporelles est définie sur les *structures de Kripke* (KS). Nous noterons AP un ensemble de propositions atomiques.

Définition 1.1. *Une* structure de Kripke*(KS) est un quadruplet $\langle S, R, s_0, \nu \rangle$, où :*

*S est un ensemble fini d'*états *;*

R est une relation de transitions *sur S ; et*

*$s_0 \in S$ est l'*état initial.

I une valuation $S \mapsto 2^{AP}$, qui à chaque état $s \in S$ associe un sous-ensemble de propositions atomiques.

La taille de KS est définie par $|S| + |R|$.

Chemins d'exécution

Soit $M = \langle S, R, _s 0, I\rangle$ une KS sur un AP. Un chemin fini est une séquence non vide d'états $s_0, \ldots, s_{n-1}$ de S telle que $(s_i, s_{i+1}) \in R$ pour tout $i \in [0, n-2]$. La longueur de π est n, notée $|\pi| = n$. Un chemin infini est une séquence infinie $\pi = (s_0, s_1, \ldots)$ d'états de S telle que $(s_i, s_{i+1}) \in R$ pour tout $i \geq 0$. On note π_i l'état à la position i dans π et π^i est la queue de π qui commence à π_i ($\pi^i = (s_i, s_{i+1}, \ldots)$). Un chemin de M est *maximal* si il ne peut pas être étendu (il est infini ou se termine sur un état sans successeur).

1.1.2 LTL

La logique linéaire est composée de quatre opérateurs temporels de base permettant de décrire des propriétés d'un chemin. Deux de ces opérateurs permettent de tous les construire. Le premier, noté $\bigcirc$ (*next*), requiert que la propriété soit vraie à l'état suivant ; le deuxième, noté U (*until*), requiert qu'une propriété soit vraie jusqu'à ce qu'une autre le devienne : la formule est vraie $\phi_0\ U\ \phi_1$ si la propriété ϕ_0 est vraie à partir de l'état initial et reste vraie jusqu'à ce que ϕ_1 le devienne. Les deux opérateurs restants sont le $\Diamond$ (*eventually*) qui requiert qu'une propriété soit vérifiée dans un état futur et le $\Box$ (*always*) qui requiert qu'une propriété soit vérifiée dans tous les états du chemin.

Syntaxe

Soit p une propriété sur l'ensemble AP, alors une formule LTL est une formule de la forme :

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc\ \phi_1 \mid \phi_1\ U\ \phi_2 \mid \Diamond\phi \mid \Box\phi$$

Sémantique

La sémantique des formules de LTL se définit sur l'ensemble des chemins infinis π d'une KS, comme indiqué dans la Table 1.1.

$M \models \phi$ si et seulement si, pour tout chemin infini π dans KS on a $\pi \models \phi$ avec :

$\pi \models p$	iff	$p \in I(\pi_0)$
$\pi \models \neg\phi$	iff	$\pi \not\models \phi$
$\pi \models \phi_1 \vee \phi_2$	iff	$\pi \models \phi_1 \vee \phi \models \phi_2$
$\pi \models \bigcirc\phi$	iff	$\lvert\pi\rvert > 1$ et $\pi^1 \models \phi$
$\pi \models \phi_1\ U\ \phi_2$	iff	il existe $k,\ 0 \leq k < \lvert\pi\rvert$, avec $\pi^k \models \phi_2$ et pour tout $i,\ 0 \leq i < k,\ \pi^i \models \phi_1$

TABLE 1.1: Sémantique de LTL

Comme discuté précédemment la sémantique des opérateur $\Diamond$ et $\Box$ se déduit des règles de la Figure 1.1. L'opérateur $\Diamond$ se déduit de U lorsque la première propriété est toujours vraie :

$$\pi \models \Diamond\phi \quad \triangleq \quad true\ U\ \phi$$

L'opérateur $\Box$ se déduit de $\Diamond$ par négation :

$$\Box\phi \quad \triangleq \quad \neg\Diamond\neg\phi$$

La sémantique est définie par rapport à un chemin. Par extension, une structure de Kripke K d'origine s_0 satisfait une formule ϕ ssi pour tout chemin π dans K d'origine s_0 on a $\pi \models \phi$.

Model-checking de LTL

Établir la satisfaction d'une formule revient à s'assurer que que toutes les séquences d'exécution satisfont cette formule. L'algorithme, basé sur les automates, réduit le problème du model-checking à celui de l'inclusion de langages.

En substance, le problème du model-checking revient à vérifier qu'un système A satisfait sa spécification S c'est à dire $L(A) \subseteq L(S)$. Dans cette approche, d'abord proposée par [7], on vérifie qu'aucun des mots acceptés par la négation de l'automate représentant la formule n'est accepté par l'automate représentant le système. En résumé, soit $L(\bar{S})$ le complément de $L(S)$, l'acceptation de langage peut s'écrire $L(A) \cap L(\bar{S}) = \emptyset$. On peut alors appliquer un algorithme de recherche des composants fortement connexes [8] et décider si l'intersection est vide en temps linéaire ($O(|S| + |R|)$). Cet algorithme peut être implémenté *à la volée*. Une exploration à la volée s'arrête dès que la formule est fausse, on évite ainsi d'explorer tous les états et la détection d'erreur est plus rapide. D'autres algorithmes existent pour le model-checking de formule LTL.

1.1.3 CTL

La logique temporelle arborescente CTL est définie à partir d'un ensemble de variables propositionnelles AP, des connecteurs classiques de la logique propositionnelle et de deux connecteurs binaires EU et AU notés de façon infixe.

Ces modalités permettent de spécifier qu'une certaine condition doit être maintenue jusqu'à la réalisation d'une certaine propriété. Cette contrainte doit être vérifiée pour un chemin (EU) ou pour tous les chemins (AU).

Syntaxe

Soit p un élément de l'ensemble AP. Alors une formule CTL est une formule de la forme :

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid EX\ \phi \mid AX\ \phi \mid \phi_1\ EU\ \phi_2 \mid \phi_1\ AU\ \phi_2$$

Sémantique

Soit M une structure de Kripke. La sémantique des formules de CTL se définit par rapport à un sommet de M, en considérant les chemins ayant pour origine ce sommet. Elle est présentée dans la Figure 1.2

La sémantique des formules de CTL est ainsi définie par rapport à un état particulier de la structure de Kripke. Par extension, une structure de Kripke K satisfait une formule ϕ - noté $K \models \phi$ ssi pour tout états s dans K on a $K, s \models \phi$. Une variante consiste aussi à noter $K \models \phi$ ssi $K, s_0 \models \phi$. Nous définissons quelques abréviations usuelles de la CTL dans la Table 1.3.

La Figure 1.2 illustre des exemples de satisfaction de modalités de CTL ; Les bulles noires matérialisent les états satisfaisant p tandis que les blanches satisfaisant $\neg p$.

$$
\begin{array}{rl}
M, s \models p & \text{ssi } p \in I(s) \\
M, s \models \neg\phi & \text{ssi } NON(M, s \models \phi) \\
M, s \models \phi_1 \vee \phi_2 & \text{ssi } M, s \models \phi_1 \text{ OU } M, s \models \phi_2 \\
M, s \models \mathbf{EX}\ \phi_1 & \text{ssi } \exists \pi\ tq\ \pi_o = s \text{ et } \pi^1 \models \phi_1, \\
M, s \models \mathbf{AX}\ \phi_1 & \text{ssi } \forall \pi\ tq\ \pi_o = s \text{ et } \pi^1 \models \phi_1, \\
M, s \models \phi_1\ \mathbf{EU}\ \phi_2 & \text{ssi } \exists \pi\ tq\ \pi_o = s, \\
 & \exists k : 0 \leq k < |\pi|, \text{ avec } M, \pi^k \models \phi_2, \\
 & \forall i : 0 \leq i < k \Rightarrow M, \pi^i \models \phi_1 \\
M, s \models \phi_1\ \mathbf{AU}\ \phi_2 & \text{ssi Si } \pi \text{ est un chemin maximal d'origine } \pi_o = s \text{ alors} \\
 & \exists k : 0 \leq k < |\pi|, \text{ avec } M, \pi^k \models \phi_2, \\
 & \forall i : 0 \leq i < k \Rightarrow M, \pi^i \models \phi_1
\end{array}
$$

TABLE 1.2: Sémantique de CTL

Model-checking de CTL

L'approche *sémantique*, qui interprète inductivement les formules sur la structure, est la première solution apportée au problème du model-checking [9, 10] . Schématiquement, l'algorithme calcule l'ensemble des états qui satisfont une formule f en étiquetant chaque état s par l'ensemble des sous formules de f satisfaites dans s. Itérativement, l'algorithme évalue ainsi la formule complète avec une complexité linéaire dans la longueur $|f|$ de la formule et la taille du graphe d'état ($\mathcal{O}(|f| \cdot (|S| + |R|))$).

1.2 Modèles temporels

1.2.1 Systèmes de transitions temporisés

Dans le contexte des applications critiques, la prise en compte des contraintes temps réel est indispensable pour la vérification formelle des comportements. Plusieurs formalismes permettent la description de modèles temps réel, en particulier les automates temporisés (Timed automata) et les réseaux de Petri temporels (Time Petri Net) que nous présenterons dans cette section. Généralement les espaces d'états calculés à partir de ces modèles sont infinis. Pour pouvoir appliquer les techniques de model-checking, il est donc nécessaire d'introduire des abstractions permettant d'en construire des représentations finies. Les abstractions considérées doivent bien sûr préserver les classes de propriétés que l'on cherche à vérifier. Nous présenterons deux de ces abstractions, le graphe des régions et le graphe de classe respectivement associées aux automates temporisés et aux réseaux de Petri temporels.

En préambule, nous définissons les *systèmes de transitions temporisés* [11] qui permettent de définir la sémantique des formalismes temporisés. Dans ce qui suit, on considère un alphabet Σ et $\mathbb{R}_{\geq} \subset \mathbb{R}$ l'ensemble des réels positifs ou nul.

Définition 1.2 (Système de Transitions Temporisé)**.** *Un* système de transitions temporisé *est un système de transitions $S = (Q, q_0, \rightarrow)$, où Q est l'ensemble des états, $q_0 \in Q$ est l'état initial et la relation de transition $\rightarrow$ est composée de transitions de délai $q \xrightarrow{d} q'$,*

$$
\begin{array}{llll}
EF\ \phi & \equiv True\ \mathbf{EU}\ \phi & AG\ \phi & \equiv \neg\, EF \neg\, \phi \\
AF\ \phi & \equiv True\ \mathbf{AU}\ \phi & EG\ \phi & \equiv \neg\, AF \neg\, \phi
\end{array}
$$

TABLE 1.3: Abréviations CTL

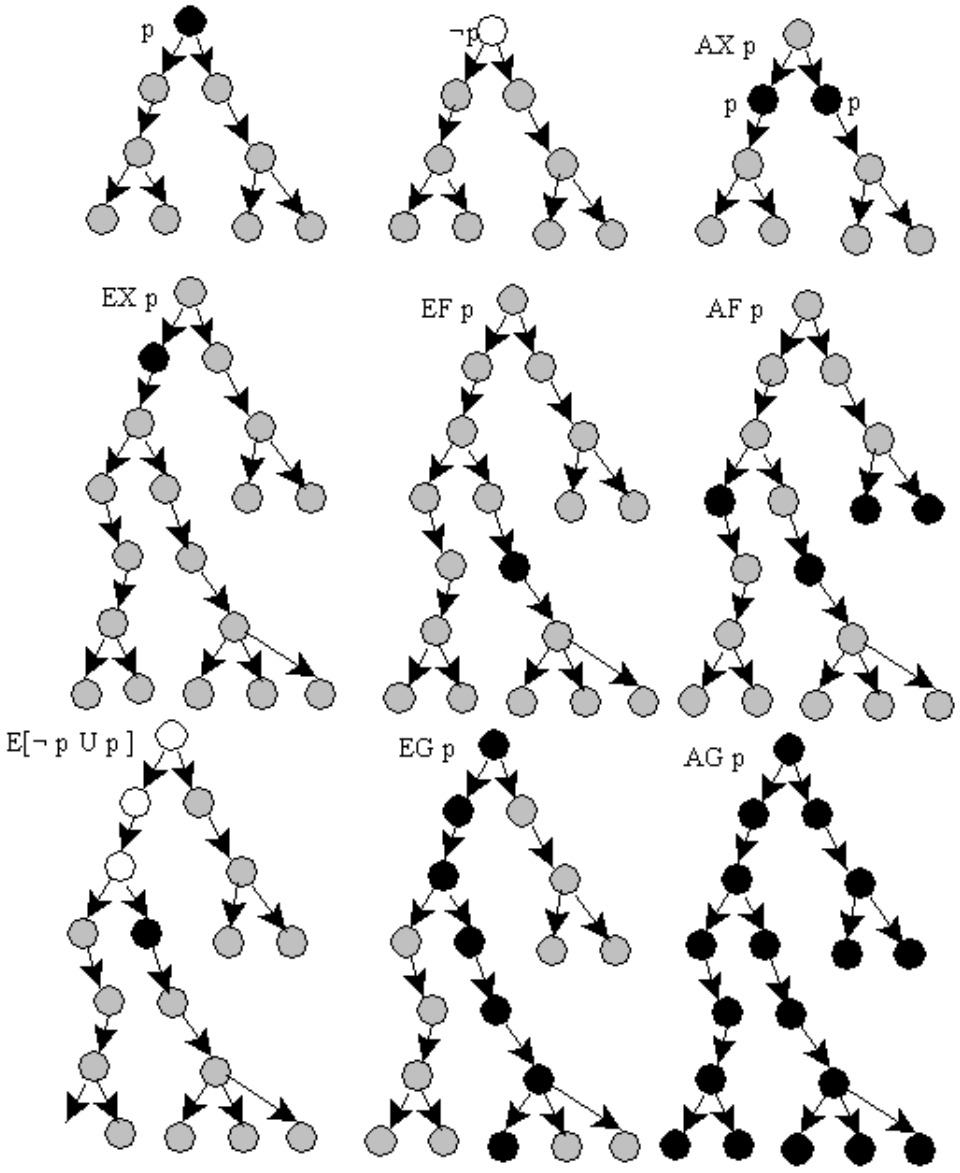


FIGURE 1.2: Exemples de satisfaction de modalités CTL

avec $d \in \mathbb{R}^+$*, et de transitions discrètes* $q \xrightarrow{a} q'$*, avec* $a \in \Sigma$*. La relation* $\rightarrow$ *doit vérifier les propriétés suivantes :*

Déterminisme temporel *si* $q \xrightarrow{d} q'$ *et* $q \xrightarrow{d} q''$*, alors* $q' = q''$ *;*

0-délai $q \xrightarrow{0} q$ *;*

Additivité *si* $q \xrightarrow{d} q'$ *et* $q' \xrightarrow{d'} q''$*, alors* $q \xrightarrow{d+d'} q''$ *;*

Continuité *si* $q \xrightarrow{d} q'$*, alors pour tout* d' *et* d'' *appartenant à* $\mathbf{R}^+$ *tel que* $d = d' + d''$*, il existe* $q'' \in Q$ *tel que* $q \xrightarrow{d'} q'' \xrightarrow{d''} q'$

Une *exécution temporisée* d'un système de transitions $S = (Q, q_0, \rightarrow)$ est séquence $\rho = q_0 \xrightarrow{d_0} q_1 \xrightarrow{a_1} q_2 \xrightarrow{d_2} \dots q_n \xrightarrow{d_n} \dots q_{n+1} \xrightarrow{a_{n+1}} \dots$ telle que pour tout indice $i \geq 1$, les transitions (q_i, d_i, q_{i+1}) et (q_i, a_i, q_{i+1}) appartiennent à S. Le mot associé à une telle exécution est le mot $\delta = (a_{i+1}, d_i)_{1 \leq 0 \leq |\rho|}$, appelé *mot de délai*. Les conditions d'acceptation décrivant les exécutions acceptées sont décrites par un ensemble d'états finals $Q_f \subseteq Q$.

On peut vouloir comparer des systèmes de transitions, à cet effet nous rappelons les définitions de *simulation* et *bisimulation*.

Définition 1.3 (Simulation, Bisimulation)**.** *Soit* $S_1 = \langle Q_1, q_0^1, \rightarrow_1 \rangle$ *et* $S_2 = \langle Q_2, q_0^2, \rightarrow_2 \rangle$ *deux systèmes de transitions temporisés sur l'alphabet* Σ*. Une relation* $\mathcal{R} \subseteq Q_1 \times Q_2$ *est une* simulation *de* S_1 *par* S_2 *si elle vérifie les conditions suivantes :*

1. $(q_0^1, q_0^2) \in \mathcal{R}$
2. *Pour tout* $(q_1, q_2) \in \mathcal{R}$*, pour tout* $\sigma \in \Sigma \cup \mathbb{R}_+$*, pour tout* $q_1' \in Q_1$ *tel que* $(q_1, \sigma, q_1') \in \rightarrow_1$*, il existe un état* $q_2 \in Q_2$ *tel que* $(q_2, \sigma, q_2') \in \rightarrow_2$ *et* $(q_2, q_2') \in \mathcal{R}$

Dans ce cas, S_2 *simule* S_1 *: tout comportement de* S_1 *est un comportement de* S_2

Une relation $\mathcal{R} \subseteq Q_1 \times Q_2$ *est une* bisimulation *de* S_1 *par* S_2 *si c'est une simulation de* S_1 *par* S_2 *et si la réciproque* $\mathcal{R}^{-1}$ *est une simulation de* S_2 *par* S_1*. Dans ce cas, on dit que* S_1 *et* S_2 *sont* bisimilaires*.*

1.2.2 Automates temporisés

Les automates temporisés [12, 13] constituent une extension des automates finis obtenue en ajoutant un ensemble fini d'horloges prenant leurs valeurs dans $R_{\geq}$. Des contraintes sont définies sur ces horloges, celles-ci permettent de garder temporellement une transition ou de décrire un invariant temporel associé à un état.

Définitions

Soit X un ensemble de variables appelées *horloges*. Une *valuation* d'horloges sur X est une fonction $v : X \to \mathbb{R}_{\geq}$ qui à chaque horloge associe une valeur de temps. On notera $\lfloor v \rfloor$ la partie entière d'une valuation et $\lceil v \rceil$ sa partie fractionnaire. L'ensemble des valuations de X est noté $\mathbb{R}_{\geq}^X$. Étant donnée une valeur $t \in \mathbb{R}_{\geq}$ et une valuation $v \in \mathbb{R}_{\geq}^X$, on définit la valuation $v + t$ par $(v + t)(x) = v(x) + t$, pour tout $x \in X$. Soit R un sous-ensemble de X, la remise à zéro des horloges de R est définie comme :

$$(v[R \leftarrow 0])(x) = \begin{cases} 0 & \text{si } x \in R \\ v(x) & \text{sinon} \end{cases}$$

Une *contrainte d'horloge* φ est une contrainte de la forme :

$$\varphi ::= x \sim c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \top$$

où $c \in \mathbb{Q}$, $\sim \in \{<, \leq, =, \geq, >\}$ et $\top$ représente le booléen vrai. On note $v \models \varphi$ lorsque la valuation v satisfait φ et $[\![\varphi]\!]_X$ l'ensemble des valuations de X qui satisfont φ. L'ensemble des contraintes d'horloges est noté $\mathcal{C}(X)$ et l'ensemble des contraintes de la forme $\sim \in \{<, \leq\}$ est noté $\mathcal{LC}(C)$.

Définition 1.4 (Automates temporisés)**.** *Un automate temporisé sur* $\mathbb{R}_{\geq}$ *est un tuple* $\mathcal{A} = \langle \Sigma, X, L, T, Inv, l_0, L_f \rangle$*, où :*

Σ *est un ensemble fini d'actions ;*

X *est un ensemble fini d'horloges ;*

L *est un ensemble fini d'états de contrôle ;*

$T \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ *est un ensemble de transitions,*

$Inv : L \to \mathcal{LC}(X)$ *est une fonction qui à chaque état associe des contraintes d'horloges décrivant un invariant temporel.*

$l_0 \in L$ *est l'état de contrôle initial ;*

$L_f \subseteq L$ *est l'ensemble d'états de contrôle finals*

Sémantique

La sémantique d'un automate temporisé est définie par un système de transitions temporisé infini.

Définition 1.5 (Sémantique d'un automate temporisé)**.** *La sémantique d'un automate temporisé $\mathcal{A}$ est définie par le système de transition temporisé $[\![\mathcal{A}]\!] = \langle Q, s_0, \rightarrow \rangle$, avec $\Sigma = \{0\}$, où :*

1. $Q = \{(s, v) \in L \times \mathbb{R}_{\geq}^{X} \mid v \models Inv(s)\}$ *est l'ensemble des états ;*
2. $\rightarrow \subset Q \times \mathbb{R}_{\geq} \times Q$ *est la relation de transition entre états définie par :*
   (a) $(s, v) \xrightarrow{d} (s', v')$ *ssi* (s, v) *et* (s', v') *appartiennent à* Q, $s = s'$ *et* $d \in \mathbb{R}_{\geq}$ *tel que* $v' = v + d$. *La transition est une* transition de délai *;*
   (b) $(s, v) \xrightarrow{\sigma} (s', v')$ *ssi* (s, v) *et* (s', v') *appartiennent à* Q *et il existe* $t = (s, \varphi, \sigma, r, s') \in T$ *tel que* $v \models \varphi$, $v' \models v[r \rightarrow 0]$. *La transition est une* transition d'action.

L'exemple 1.2.1 donne un exemple d'automate temporisé modélisant une minuterie simple.

Exemple 1.2.1. *Soit l'automate temporisé* $\mathcal{A} = \langle \Sigma, X, L, T, Inv, l_0, L_f, L_r \rangle$ *avec :*

- $\Sigma = \{abandon?, \text{début}, temporisation!\}$;
- $X = \{x\}$;
- $L = \{Repos, Armée\}$;
- $T = \left\{ \begin{array}{l} (Repos,\ \top,\ \text{début},\ x := 0,\ Armée), \\ (Armée,\ x = 3,\ \text{temporisation}\,!, \emptyset,\ Repos), \\ (Armée,\ \top,\ \text{abandon}\,?, \emptyset,\ Repos) \end{array} \right\}$;
- $Inv(Repos) = \top, Inv(Armée) = x \leq 3$;
- $l_0 = Repos$; *et*
- $L_f = \{Repos\}$

Cet automate, représenté dans la Figure 1.3 modélise une minuterie simple. Initialement, la minuterie est désactivée, dans l'état Repos. La minuterie est déclenchée dès qu'elle se trouve dans l'état Armée. La transition étiquetée début *remet à zéro l'horloge* x *et change l'état de la minuterie de Repos à Armée. Deux transitions sont possibles depuis l'état Armée. La première est tirée sur réception d'un message d'*abandon *(le* ? *indique généralement une réception de message). La deuxième transition est tirée lorsque la valeur de* x *est* 3. *Dans ce cas, un message* temporisation *est émis (le* ! *indique généralement une émission de message).*

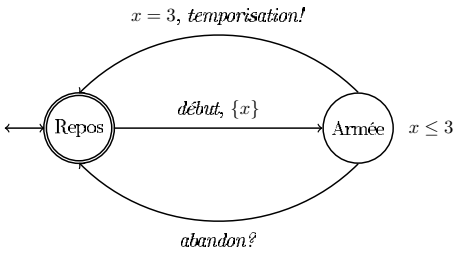


FIGURE 1.3: Exemple d'automate temporisé

La modélisation des systèmes est faite le plus souvent par décompositions successives en éléments de plus en plus simple. Des opérations de compositions sont définies sur les automates temporisés qui permettent de supporter cette démarche. Si cette facilité de

modélisation par composants est indispensable pour la modélisation des systèmes, elle n'ajoute pas d'expressivité sur le plan théorique : on peut toujours construire un automate produit ayant le même comportement (au sens de la bisimulation) que la composition parallèle considérée.

Abstractions

Graphe des régions La sémantique d'un automate temporisé est un système de transitions infini. Pour permettre d'appliquer les techniques de model-checking, il convient donc d'obtenir une abstraction finie de cet espace infini, cette abstraction finie doit bien sûr préserver la classe de propriétés envisagée.

Les régions sont des classes d'équivalences d'une relation définie sur l'ensemble des valuations d'horloges. Cette relation d'équivalence est basée sur la propriété suivante : Deux états $(l, v), (l, v')$ tels que v et v' ont la même partie entière sur chaque horloge et tels que leurs parties fractionnaires sont dans le même ordre offrent le même comportement.

Nous notons K la constante maximale en valeur absolue apparaissant dans une contrainte d'horloge de l'automate. Soit la relation $\approx$ telle que $v \approx v'$ si et seulement si les conditions ci-dessous sont satisfaites :

- Pour toute horloge $x \in X$, soit $v(x)$ et $v'(x)$ sont strictement supèrieures à K, soit $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$;
- pour toutes horloges $x, x' \in X$, si la valeur absolue $|v(x) - v(x')| \leq K$ alors :
  - $\lceil v(x) \rceil = 0$ ssi $\lceil v'(x) \rceil = 0$
  - $\lceil v(x) \rceil \leq \lceil v(x') \rceil$ ssi $\lceil v'(x) \rceil \leq \lceil v'(x') \rceil$

La relation $\approx$ est une relation d'équivalence et ses classes d'équivalences sont appelées *régions*. Nous notons $\mathcal{R}$ l'ensemble des régions. La relation $\approx$ satisfait les propriétés suivantes :

$$v \approx v' \Rightarrow \begin{cases} \text{pour toute contrainte } g \text{ de } \mathcal{A},\ v \models g \Leftrightarrow v' \models g \\ \text{pour tout } d \in \mathbb{R}_{\geq}, \text{ il existe } d' \in \mathbb{R}_{\geq} \text{ tel que } v + d \approx v' + d' \end{cases}$$

La Figure 1.4 montre l'ensemble des régions pour les valuations possibles de deux horloges x et y avec $K = 3$.

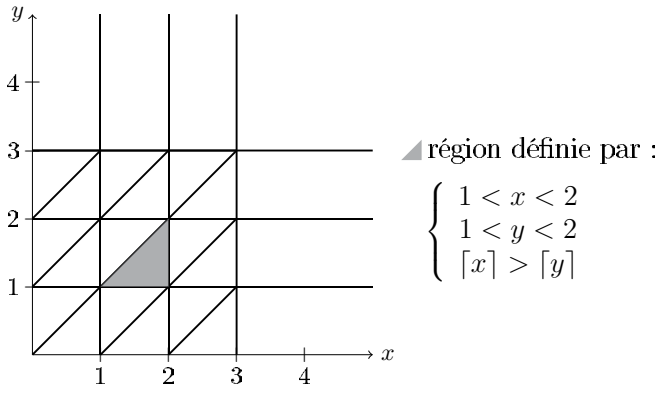


FIGURE 1.4: Exemple de régions

Définition 1.6 (Graphe des régions)**.** *Le* graphe des régions *d'un automate $\mathcal{A}$ est un système de transitions fini $\mathcal{R}(\mathcal{A}) = (\Gamma, \gamma_0, \rightarrow)$ où :*

1. $\Gamma = \{(l, b) \mid l \in L, b \in \mathcal{R}\}$ *;*

2. γ_0 *est l'état initial* (l_0, b_0) *où* b_0 *est la région qui contient la valuation* 0 *; et*
3. $\rightarrow \subseteq \Gamma \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ *et* $((l,b),\sigma,(l',b'))) \in \rightarrow$ *ssi* $(l,b) \neq (l',b')$ *et*
   (a) *soit* $\sigma \in \Sigma$ *et* $(l,v) \xrightarrow{\sigma} (l',v')$ *est une transition de* $[\![\mathcal{A}]\!]$ *un certain* $v \in b$ *et un certain* $v' \in b'$ *;*
   (b) *soit* σ *est le symbole* ϵ *et il existe* $d \in \mathbf{R}_{\geq}$ *tel que* $(l,v) \xrightarrow{d} (l,v')$ *est une transition de* $[\![\mathcal{A}]\!]$ *pour un certain* $v \in b$ *et un certain* $v' \in b$.

De plus, l'ensemble des états finals, noté Γ_f *est défini par :*

$$\Gamma_f = \{(l,b) \mid l \in F, b \in \mathcal{R}\}$$

Le graphe des régions est bisimilaire en temps abstrait à la sémantique de l'automate. Soit la relation $\simeq \subseteq (L \times R^X_{\geq}) \times \Gamma$ définie, pour tout état (l,v) et (l,b), par :

$$(l,v) \simeq (l,b) \Leftrightarrow [v] = b \quad \text{où } [v] \text{ désigne la région de } v$$

Cette relation définie une *bisimulation*, entre le système de transitions (généralement infini) et le graphe de régions (fini) qui lui est associé.

Cette bisimulation est dite *à temps abstrait* dans la mesure où les transitions discrètes sont préservées et la valeur exacte du délai des transitions d'écoulement de temps est abstraite.

C'est une propriété fondamentale de l'automate des régions, et des abstractions finies des sémantiques généralement infinies des modèles temporisés.

Le *problème du vide* - consistant à déterminer si le langage accepté par l'automate est vide ou non - est décidable pour la classe des automates avec contraintes d'horloges diagonales, eg. des contraintes de la forme $x - y \sim c$ où c est une constante et $\sim \in \{<, \leq, =, \geq, >\}$ ([12, 13]).

Zones et analyse avant des automates temporisés Pour réduire l'explosion des régions une abstraction plus grossière est considérée : une zone consiste à regrouper des régions convexes.

Il y a principalement deux familles d'algorithmes qui permettent d'analyser des systèmes temporisés à la volée. La première, appelée *analyse arrière*, consiste à calculer itérativement les prédécesseurs des états que l'on cherche à atteindre et à tester si un état initial se trouve dans l'ensemble calculé. La seconde, appelée *analyse avant*, consiste à calculer itérativement les successeurs des états initiaux et à tester si l'état que l'on cherche à atteindre est calculé. Nous présenterons uniquement l'analyse avant, implémentée dans l'outil **Uppaal**.

L'analyse avant repose sur une représentation symbolique des valuations appelée *zone*. Une zone est un ensemble de valuations défini par une conjonction de contraintes simples $x \sim c$ ou $x - y \sim x$ où $\sim \in \{<, \leq, =, \geq, >\}$. Dans l'analyse avant, les objets manipulés seront des paires (l, Z) où l est un état de contrôle et Z une zone. Plusieurs opérations peuvent être effectuées sur les zones, nous en citons trois :

— Le futur d'une zone Z, défini par $\overrightarrow{Z} = \{v + t \mid v \in Z \wedge t \in \mathbb{R}_{\geq}\}$;

— l'intersection de Z et Z', définie par $Z \cap Z' = \{v \mid v \in Z \cap Z'\}$;

— la remise à zéro de r de Z, définie par $[r \leftarrow 0]Z = \{[r \leftarrow 0]v \mid v \in Z\}$

Ces opérations, définies grâce des formules du premier ordre, produisent des zones. La structure de donnée principalement utilisée pour représenter les zones est la *matrice de différence des bornes*, où *Difference Bound Matrice* (DBM) en anglais.

Partant des états initiaux, l'analyse avant calcule d'abord les successeurs en un pas, puis ceux en deux pas, etc ... et teste si les états recherchés se trouvent dans l'ensemble calculé. Un pas de l'analyse avant se calcule simplement à l'aide des zones. Si $t = l \xrightarrow{g,a,r} l'$

est une transition de l'automate temporisé et si Z est une zone, l'ensemble des successeurs de (l, Z) en un pas en tirant t est la paire (l', Z') où $Z' = [r \leftarrow 0](g \cap \overrightarrow{Z})$.

Pour garantir la terminaison de l'algorithme, il est nécessaire d'appliquer un opérateur d'*extrapolation* (appelé aussi *normalisation*). Soit k la constante maximale apparaissant dans une contrainte de l'automate. L'extrapolation d'une zone Z par rapport à k est la plus petite zone contenant S définie par des contraintes utilisant des constantes entre $-k$ et $+k$. Les valeurs d'horloges supèrieures à k sont abstraites, il suffit de retenir qu'elles ont dépassé k. Cet opérateur d'extrapolation garantit que le graphe des zones est fini.

L'extrapolation préserve l'accessibilité des états de contrôles.

1.2.3 Réseau de Petri temporel

Les réseaux de Petri temporel (TPN) sont une extension des réseaux de Petri [14] où chaque transition est associée à un intervalle temporel qui est l'intervalle dans lequel la transition va être tirée.

Définitions

Nous commençons par définir les réseaux de Petri.

Définition 1.7 (Réseau de Petri)**.** *Un réseau de Petri est un tuple* $\langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$ *où :*

- P *est un ensemble fini non vide de places,*
- T *est un ensemble fini non vide de transition,*
- $\mathbf{Pre} : T \to P \to \mathbb{N}$ *est la fonction de précondition,*
- $\mathbf{Post} : T \to P \to \mathbb{N}$ *est la fonction de postcondition*

Un réseau de Petri est généralement marqué, c'est à dire équipé d'un marquage initial. Dans toute cette section, nous ne considérons que des réseaux marqués.

Définition 1.8 (Réseau de Petri marqué)**.** *Un réseau de Petri* marqué *est un réseau de Petri associé à un marquage initial* m_0 *qui à chaque place associe un marquage, une fonction de* P *dans* $\mathbf{N}$ *qui à chaque place associe une valeur dans* $\mathbb{N}$

Soit Σ_ϵ un ensemble d'étiquettes, contenant l'étiquette vide ϵ. Un réseau de Petri étiqueté est un réseau de Petri associé à une fonction d'étiquetage qui à chaque transition du réseau associe une étiquette de Σ_ϵ.

Définition 1.9 (Réseau de Petri étiqueté)**.** *Un réseau de Petri* étiqueté *est un réseau de Petri associé à une fonction* $\lambda : T \to \Sigma_\epsilon$ *qui à chaque transition associe une étiquette dans* Σ_ϵ*.*

Un *Réseau de Petri Temporel* (ou TPN) [15] est un réseau de Petri où chaque transition t est associée à deux réels a et b, tels que $a \leq b$, l'intervalle $[a, b]$ est appelé *intervalle statique*. Cet intervalle spécifie des dates de tirs possibles de t. Si t reste sensibilisée, sans interruption, après qu'elle ait été sensibilisée alors :

- $a (0 \leq a)$ est le temps minimum qui doit s'écouler, à partir de l'instant où t a été sensibilisée, avant que t ne puisse être tirée.
- $b (a \leq b)$ est la durée maximale pendant laquelle t peut rester sensibilisée sans tirée.

Les dates a et b, pour une transition t, sont relatives à la date où t a été sensibilisée. Soit τ cette date. Alors t ne peut être tirée avant $\tau + a$ et doit être tirée avant $\tau + b$, à moins qu'elle ne soit désensibilisée avant. La Figure 1.5 montre un TPN.

Nous donnons maintenant une définition des TPN. Nous noterons $\mathbb{I} \subseteq \mathbb{IR}_{\geq}$ l'ensemble des intervalles réels non vides à bornes rationnelles non négatives.

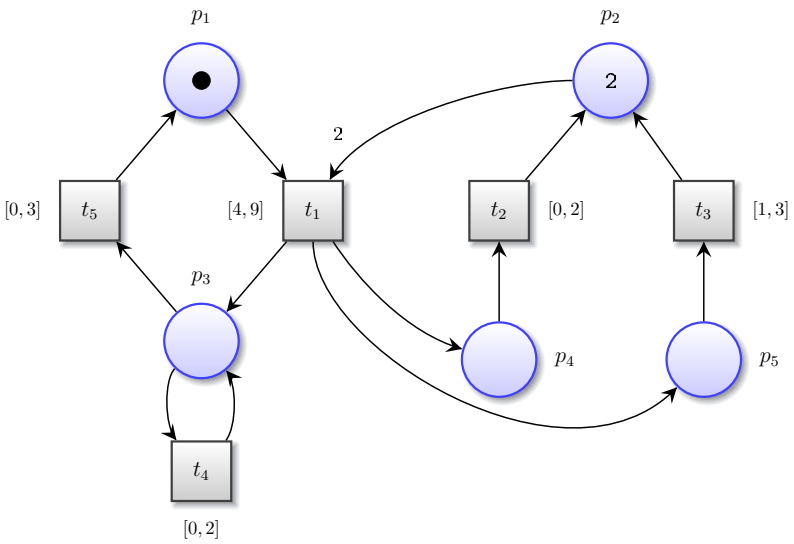


FIGURE 1.5: Exemple de TPN

Définition 1.10 (Réseau de Petri temporel)**.** *Un TPN est un tuple $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s, \rangle$ où :*

- *P est un ensemble fini non vide de places,*
- *T est un ensemble fini non vide de transitions,*
- *$\mathbf{Pre} : T \to P \to \mathbb{N}$ est la fonction de précondition,*
- *$\mathbf{Post} : T \to P \to \mathbb{N}$ est la fonction de postcondition*
- *m_0 est le marquage initial (eg: une fonction $P \to \mathbb{N}$)*
- *$\mathbf{I_s} : T \to \mathbb{I}$ est une fonction appelée intervalle statique.*

Sémantique

Une transition $t \in T$ est *sensibilisée* par un marquage m si et seulement si $m \geq \mathbf{Pre}(t)$. On note $\mathbb{M}$ l'ensemble des marquages d'un réseau et $\mathcal{E}(m)$ l'ensemble des transitions sensibilisées par $m \in \mathbb{M}$.

Pour $i \in \mathbb{I}$, $\downarrow i$ désigne sa borne inférieure, et $\uparrow i$ sa borne supèrieure (ou ∞ si i n'est pas borné). Pour tout $\theta \in \mathbb{R}_{\geq}$ on définit $i \dot{-} \theta = \{x - \theta | x \in i \wedge x \geq \theta\}$. Cette opération modélise l'écoulement du temps.

Un *état* d'un TPN est une paire $s = (m, I)$ où M est un marquage et $I : T \to \mathbb{I}$ est une fonction partielle, appelée fonction *d'intervalle dynamique* associant un intervalle temporel à chaque transition sensibilisée par m.

Définition 1.11 (Graphe d'états)**.** *La sémantique d'un TPN $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$, aussi appelé* graphe d'états*, est donnée par un système de transitions temporisé $SG = \langle S, s_0, \to \rangle$ où :*

- *S est l'ensemble des états du TPN ;*
- *$s_0 = (m_0, I_0)$ est l'état initial, où m_0 est le marquage initial et I_0 la fonction intervalle statique restreinte aux transitions sensibilisées par m_0 ;*
- *$\to \subseteq S \times (T \cup \mathbb{R}_{\geq}) \times S$ est la relation de transition entre états, définie comme suit pour tout $t \in T$ et $\theta \in \mathbb{R}_{\geq}$:*
  - *(i) $(m, I) \xrightarrow{t} (m', I')$ si et seulement si:*
    1. *$m \geq \mathbf{Pre}(t)$*

2. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$

3. $0 \in I(t)$

4. $(\forall k)(m' \geq \mathbf{Pre}(k) \Rightarrow$
   $I'(k) = si\ k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)\ alors\ I(k)\ sinon\ I_s(k))$

(ii) $(m, I) \xrightarrow{\theta} (m, I')$ *si et seulement si:*

5. $(\forall k)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k) \wedge I'(k) = I(k) \dot{-} \theta)$

Une transition t peut être tirée depuis l'état (m, I) si t est sensibilisée par m et *immédiatement* franchissable(ie: $\downarrow(I(t)) = 0$). Dans l'état destination, les transitions k qui sont restées sensibilisées pendant le tir de t (t exclue) conservent leurs intervalles ; ces transitions k sont dites *persistantes*. Les autres transitions, dites *nouvellement sensibilisées*, sont associées à leurs intervalles statiques.

Une transition continue de θ est possible depuis (m, I) si et seulement si θ est plus petit que $\uparrow I(k)$, pour tout $k \in \mathcal{E}(m)$.

Comme pour les automates temporisés, l'espace d'état d'un TPN est généralement infini, même lorsque le réseau de Petri sous-jacent est borné. Il est impossible de l'exploiter à des fins de vérification. Plusieurs abstractions existent et sont implémentées dans l'outil TINA [16], présenté à la Section 1.4.

Abstraction des classes d'états

Un état du graphe d'états peut être vu comme une paire (m, D), où m est un marquage et D et un ensemble de vecteurs, appelé *domaine de tir*. Chaque vecteur du domaine de tir a une entrée par transition sensibilisée par m. La projection des vecteurs de D sur l'entrée i correspond à l'intervalle dynamique de la i-ème transition sensibilisée. Un domaine de tir décrit l'ensemble des solutions d'un système d'inéquations linéaires où chaque variable correspond exactement à une transition sensibilisée dans m.

Nous donnons deux constructions différentes pour le graphe des classes d'états.

Construction géométrique Soit $\mathcal{N}$ un TPN, SG son graphe d'états, π un chemin du graphe et $\sigma = (t_0, \theta_0, \ldots, t_n, \theta_n)$ la séquence des transitions étiquetées de π. On note σ_T la projection de σ sur les transitions de $\mathcal{N}$, σ_T est appelée *séquence de tir*.

Soit C_σ l'ensemble des états atteints par l'ensemble des chemins π' ayant la même séquence de tir que σ_T. Tous les états de C_σ ont le même marquage (le marquage d'un état ne dépend que de la séquence de tir). Le domaine de tir de C_σ est la réunion des domaines de tirs des états qu'il contient.

Si un marquage dépend uniquement de la séquence de tirs, des séquences de tirs différentes peuvent emmener au même marquage. Parmi celles ci, des séquences plus petites (en nombre de transitions tirées) peuvent résulter en le même domaine de tir que d'autres, plus longues. La relation $\cong$ capture ses propriétés.

Soit $\cong$ la relation telle que $C_\sigma \cong C_{\sigma'}$ si et seulement si C_σ et $C_{\sigma'}$ ont le même marquage et le même domaine de tir. Si $C_\sigma \cong C_{\sigma'}$ alors tout chemin possible depuis un état de C_σ l'est aussi depuis un état de $C_{\sigma'}$.

Le graphe des classes d'états de [17], ou SCG est l'ensemble des ensembles d'états C_σ, pour tout séquence σ tirable, considérés modulo la relation $\cong$, muni de la relation de transition $C_\sigma \xrightarrow{t} X$ si et seulement si $C_{\sigma.t} \cong X$. La classe initiale est la classe d'équivalence de l'ensemble des états constitué du seul état initial.

Le *SCG* est obtenu comme suit. Les classes d'états sont représentées par des paires (m, D), où m est un marquage et D un domaine de tir décrit par un système d'inéquations linéaires $A\underline{\phi} \leq \underline{w}$ (et plus en un ensemble de vecteur). Chaque variable ϕ_t de $\underline{\phi}$ est associée par bijection à la transition t, nécessairement sensibilisée par m. On a $(m, D) \cong (m', D')$ si et seulement si $m = m'$ et les domaines D et D' ont même ensemble de solutions.

Algorithme 1.2.1 Calcul du *SCG*

Pour toute séquence σ tirable, L_σ peut être calculée comme suit. Calculer le plus petit ensemble $\mathcal{C}$ de classes contenant L_ϵ et tel que, lorsque $L_\sigma \in \mathcal{C}$ et $\sigma.t$ est tirable alors $(\exists X \in \mathcal{C})(X \cong L_{\sigma.t})$.

— La classe initiale est L_ϵ est (m_0, D_0), où D_0 est le domaine défini par le système d'inégalité $\{\downarrow I_s(t) \leq \phi_t \leq \uparrow I_s(t) \mid t \in \mathcal{E}(m_0)\}$.

— Si σ est tirable et $L_\sigma = (m, D)$, alors $\sigma.t$ est tirable si et seulement si:

1. $m \geq \mathbf{Pre}(t)$ (t est sensibilisée m)
2. Le système $D \wedge F$ est satisfiable, avec $F = \{\phi_t \leq \phi_i \mid i \neq t \wedge i \in \mathcal{E}(m)\}$

— Si $\sigma.t$ est tirable et que $L_\sigma = (m, D)$ alors $L_{\sigma.t} = (m', D')$ avec :

$m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$

D' est construit à partir de D en trois étapes :

1. Les contraintes de F sont ajoutées D ;
2. Pour chaque k sensibilisée dans m' on introduit une nouvelle variable ϕ'_k, telle que: $\begin{cases} \phi'_k = \phi_k - \phi_t & \text{si } k \text{ est persistente} \\ \downarrow I_s(k) \leq \phi'_k \leq \uparrow I_s(k) & \text{si } k \text{ est nouvellement sensibilisée} \end{cases}$
3. Les (anciennes) variables ϕ_i sont éliminées.

L_σ est la classe d'équivalence pour la relation $\cong$ de l'ensemble C_σ des états atteints en tirant les transitions de σ depuis s_0. L'équivalence $\cong$ est vérifiée en mettant les systèmes représentant les domaines de tir sous forme canonique. Ces systèmes sont des systèmes de différences, calculer leur forme canonique se ramène à un problème de plus court chemin entre toutes paires de sommets, résolu en temps polynômial en utillisant, par exemple, l'algorithme de Floyd/Warshall.

Deux ensembles d'états C_σ et $C_{\sigma'}$ peuvent être équivalent par $\cong$ mais de contenus différents. Par exemple, l'ensemble C_σ peut contenir les états (m, a) et (m, b) (a et b sont des ensembles de vecteurs) mais la classe de $C_{\sigma'}$ contenir les états (m, a),(m, b_1) et (m, b_2) avec $b = b_1 \cup b_2$. Dans ce cas, on a bien $C_\sigma \cong C_{\sigma'}$ mais $C_\sigma \neq C_{\sigma'}$.

L'exemple 1.2.2 montre le calcul de quelques classes du réseaux de la Figure 1.5.

Exemple 1.2.2. *Soit le réseau de la Figure 1.5. La classe initiale c_0 est décrite comme l'état initial e_0. Le tir de t_1 depuis c_0 conduit à une classe c_1 décrite comme l'état e_1. Le tir de t_2 depuis c_1 conduit à $c_2 = (m_2, D_2)$, avec $m_2 = (p_2, p_3, p_5)$ et D_2 déterminé en 3 étapes :*

1. *on ajoute à D_1 les conditions de tirabilités de t_2, exprimée par le système :*

$$t_2 \leq t_2$$
$$t_2 \leq t_4$$
$$t_2 \leq t_5$$

2. *aucune transition n'est nouvellement sensibilisée et les transitions t_3, t_4, t_5 restent sensibilisées lors du tir de t_2. On ajoute donc les équations $t'_i = t_i - 2$, pour $i \in \{3, 4, 5\}$;*

3. on élimine les variables t_i, ce qui conduit au système :

$$\begin{array}{ll} 0 \leq t'_3 \leq 3 & t'_4 - t'_3 \leq 1 \\ 0 \leq t'_4 \leq 2 & t'_5 - t'_3 \leq 2 \\ 0 \leq t'_5 \leq 3 & \end{array}$$

Un réseau de Petri est *borné* si le marquage de toute place admet une borne supèrieure. Le problème de déterminer si un TPN est borné est indécidable. Néanmoins, des conditions suffisantes peuvent être établies, qui sont les suivantes. Soit $\mathcal{N}$ un TPN, alors $\mathcal{N}$ est borné si il ne contient pas de paire de classes $c = (m, D)$ et $c' = (m', D')$ telles que :

1. c' est accessible depuis c ;
2. $(\exists p \in P)(m'(p) \leq m(p))$;
3. $D = D'$;
4. $(\forall p \in P)(m'(p) > m(p) \Rightarrow m'(p) \geq max_{t \in T}(\mathbf{Pre}(p, t)))$

Le graphe des classes d'états et le graphe d'états contiennent les mêmes marquages et les mêmes séquences de tir [18]. Pour une classe C_σ, si un des états de C_σ est un état puits alors tous les états de C_σ le sont puisque la propriété d'état puits dépend uniquement du marquage, et tous les états d'une classe ont le même marquage. Les *SCG* et les graphes d'états, vus comme des systèmes à transition étiquetés, ont donc exactement les mêmes traces. Comme indiqué en Section 1.1.2, la sémantique d'une formule LTL se définit sur les chemins d'un système de transitions étiqueté. D'où le *SCG* préserve les marquages ainsi que les formules LTL du graphe d'états.

Construction incrémentale L'Algorithme 1.2.1 donne une définition géométrique du graphe des classes. En pratique, la construction du graphe repose sur des procédures plus efficaces. Nous présentons dans cette section une construction incrémentale des domaines de tir, que nous utiliserons dans le Chapitre 4.

Afin de clarifier la notation, les variables des domaines de tir (notées ϕ_i dans l'Algorithme 1.2.1) sont notées comme les transitions auxquelles elles sont associées (eg: ϕ_i est notée i). La borne minimale(resp. maximale) d'un intervalle statique sera notée α_t^s(resp. β_t^s) au lieu de $\downarrow I_s(t)$(resp. $\uparrow I_s(t)$).

Les domaines de tirs peuvent toujours être écrits sous la forme d'un système de contraintes de différences, appelée *forme normale*, c'est à dire un ensemble d'inégalité de la forme, pour tout $i \in T$:

$$\begin{array}{ll} \alpha_i \leq i \leq \beta_i & \\ i - j \leq \gamma_{i,j} & (i \neq j \in T) \end{array}$$

où α_i, β_i et $\gamma_{i,j}$ sont des constantes. Soit D un domaine de tir, alors il existe une *forme canonique* de D, qui peut être obtenue à partir de la forme normale en déterminant les plus grand α_i et les plus petits β_i et $\gamma_{i,j}$ qui préservent l'ensemble des solutions de D. Ainsi, deux domaines de tirs sont égaux si et seulement si ils ont la même forme canonique.

Le Lemme 1.12, tiré de [19], donne une construction incrémentale de domaines de tir en forme canonique de complexité quadratique en temps.

Lemme 1.12 (Calcul incrémental des domaines de tir)**.** *Soit $C = (m, D)$ une classe, et D en forme canonique. Pour toute transition $t \in \mathcal{E}(m)$ il existe une classe unique*

$C' = (m', D')$ *obtenue à partir de* C *en tirant* t *telle que* D' *est aussi en forme canonique, et pour toutes les transitions distinctes* $i, j \in \mathcal{E}(m')$*:*

$$\begin{array}{rclll} \beta'_i & = & \beta^s_i & \textit{si } i \textit{ nouv. sensibilisée,} & \beta'_i = \gamma_{i,t} \textit{ sinon} \\ \alpha'_i & = & \alpha^s_i & \textit{si } i \textit{ nouv. sensibilisée,} & \alpha'_i = \max(0, -\min_{k\in\mathcal{E}(m)}(\gamma_{k,i})) \textit{ sinon} \\ \gamma'_{i,j} & = & \beta'_i - \alpha'_j & \textit{si } i \textit{ ou } j \textit{ nv. sensibilisée,} & \gamma'_{i,j} = \min(\gamma_{i,j}, \beta'_i - \alpha'_j) \textit{ sinon} \end{array}$$

A partir de cette construction incrémentale, il est possible de proposer des invariants sur les coefficients des domaines de tir en forme canonique. Ces invariants nous serons utiles pour la définition de l'ordre sur les transitions de même intervalle statique dans le Chapitre 4.

Lemme 1.13. *Pour toute classe* (m, D) *telle que* D *est en forme canonique, et pour toutes transitions* i, j, k *sensibilisée dans* C *:*

1. $\gamma_{i,j} \leq \beta_i - \alpha_j$
2. $\beta_i \leq \gamma_{i,j} + \beta_j$
3. $\alpha_i \leq \gamma_{i,j} + \alpha_j$
4. $\gamma_{i,j} \leq \gamma_{i,k} + \gamma_{k,j}$
5. $0 \leq \alpha_i \leq \alpha^s_i$
6. $0 \leq \beta_i \leq \beta^s_i$
7. *Si* $C \xrightarrow{t} (m', D')$ *et* D' *en forme canonique et* i *est persistante dans* C' *alors* $\beta'_i \leq \beta_i$

Inclusion des classes d'états

L'inclusion des classes d'états est une abstraction du graphe d'états qui préserve les marquages. Elle comporte un test d'inclusion de classe et ne mémorise pas les classes incluses dans une classe déjà construite. Cette abstraction produit généralement un graphe plus compact que le *SCG*, mais ne préserve pas les traces. Plus précisément, soit $C = (m, D)$ et $C' = (m', D')$ deux classes. On définit la relation $\sqsubseteq$ par :

$$C \sqsubseteq C' \Leftrightarrow m = m' \wedge D \subseteq D'$$

Plutôt que de procéder comme dans l'Algorithme 1.2.1, on construit un ensemble C de classes tel que, lorsque $L_\sigma \in \mathcal{C}$ et $\sigma.t$ est tirable on a $(\exists X \in \mathcal{C})(L_{\sigma.t} \sqsubseteq X)$.

Intuitivement, si une telle classe X existe on ne trouvera pas de nouveaux marquages en mémorisant $L_{\sigma.t}$ car les futurs possibles depuis X le sont aussi depuis $L_{\sigma.t}$. La construction ne préserve plus les séquences de tir du graphe d'états, et donc ses propriétés LTL, mais seulement les marquages.

Cette abstraction est suffisante pour la vérification de toutes les propriétés d'accessibilité.

Autres abstractions

Les classes construites par l'Algorithme 1.2.1 représentent les classes d'équivalence de $\cong$. Deux ensembles C_σ et $C_{\sigma'}$ peuvent être équivalent par $\cong$, mais ne pas contenir le même ensemble d'états. En conséquence, le *SCG* ne peut pas être utilisé pour démontrer l'accessibilité ou non d'un état particulier du TPN.

Les *classes d'états fortes* coïncident exactement avec ces ensembles d'états C_σ considérées ici sans autre forme d'équivalence que l'égalité. Une construction analogue à celle de l'Algorithme 1.2.1 permet de construire un *graphe des classes fortes*, ou *SSCG*, tel qu'un état (m, I) est accessible si et seulement si il existe une classe forte dans le SSCG qui le contient. Le SSCG préserve aussi les traces et donc les propriétés LTL.

Les *SSCG* et *SCG* ne préservent pas les propriétés de branchement, exprimées dans les logiques à temps arborescent comme CTL. Dans [18], les auteurs étendent la technique "standard" de graphes de classes pour obtenir des classes *atomiques* préservant les propriétés de branchement.

1.3 Le problème de l'explosion combinatoire

La principale limite à l'utilisation du model-checking est liée au problème de l'*explosion combinatoire* : le nombre d'états augmente de façon exponentielle en fonction de la complexité du système. La taille de l'espace d'états peut alors excéder la quantité de mémoire disponible, rendant impossible la vérification de propriétés sur l'espace d'états.

Les systèmes que nous étudions sont généralement composés de processus qui interagissent. La description du système spécifie le comportement de chacun de ces processus et leurs interactions. L'explosion combinatoire est liée au nombre de processus qui compose le système : la taille de l'espace d'état est exponentielle par rapport au nombre de processus. Par exemple, considérons un ensemble de n processus à deux états. L'état global du système peut-être représenté par un vecteur de n bits et le nombre maximum d'états dans le système est 2^n.

Différentes techniques permettent de limiter l'explosion combinatoire. Nous présentons dans cette section quelques techniques de réductions parmi les plus connues.

Il est important de noter que la modélisation est une étape priviliégiée pour la prévention de l' explosion combinatoire : Le choix des abstractions et des simplifications influe énormément sur la taille de l'espace d'états.

Une bonne abstraction est une abstraction qui ne conserve que le minimum de comportements nécessaires pour la vérification des propriétés. Il est alors important de ne vérifier qu'un nombre limité de propriétés. Les simplifications doivent n'éliminer, dans la mesure du possible, que les comportements inutiles à la vérification des propriétés. Finalement, il n'est pas possible de définir des abstractions ou simplification sans connaitre précisément la sémantique du langage de modélisation ni les comportements, souvent subtils, du système à vérifier.

Nous présentons dans le chapitre 2 un exemple de modélisation et vérification formelle par model-checking dans un contexte avionique.

1.3.1 Model-checking symbolique par les BDD

Le model-checking symbolique [20] est une technique où les états sont représentés symboliquement par des diagrammes de décision. Cette représentation symbolique permet une représentation plus compacte de l'espace d'états.

Les valeurs de vérité d'une proposition booléenne peuvent être représentées par un arbre de décision binaire. Dans cet arbre, une branche représente une assignation aux variables de la formule et la feuille de la branche porte la valeur de vérité de la formule étant donnée cette assignation. Cette représentation n'est pas optimale : l'arbre peut contenir des sous-arbres isomorphes et par définition il n'y a que deux valeurs de vérité possibles. La solution à ce problème est un graphe acyclique, appelé diagramme de décision binaire (BDD), qui identifie de manière unique les sous-arbres isomorphes et les feuilles identiques. Il a été démontré [21] qu'il existe un diagramme unique pour chaque ensemble d'états, représentés par des tuples booléens sur un ensemble de propositions atomiques ordonnées.

Un diagramme de décision binaire ordonné (OBDD) est un BDD avec un ordre total sur les noeuds. Pour tout chemin du graphe allant de la racine jusqu'à une feuille, les variables apparaîtront dans le même ordre. La taille d'un OBDD est fortement dépendante de cet ordre et décider si un ordre conduit à une taille polynomiale de l'OBDD est un problème NP-complet [22]. Néanmoins, le model-checking symbolique peut-être très efficace en pratique pour les modèles non temporisés (ou en temps discret) [23].

Des structures analogues aux BDD, appelées diagrammes de différences d'horloges (CDD), ont été proposées pour la vérification des automates temporisés [24, 25]. Un CDD est un graphe orienté acyclique qui capture une union de zones (Section 1.2.2). Un noeud terminal représente une valeur booléenne. Un noeud non terminal est associé à une paire d'horloges. Chaque arc sortant est associé à un intervalle bornant la différence entre les horloges du noeud. Un chemin dans le CDD représente une zone définie par la conjonction des contraintes associées au chemin. Étant donnée une zone, elle appartient à l'union s'il existe un chemin dans le graphe qui se termine par une constante vraie. Comme pour les BDD, la taille d'un CDD est fonction de l'ordre (total) choisi pour les noeuds mais contrairement aux BDD il n'existe pas de CDD unique pour un ensemble d'horloges. L'efficacité de cette approche dépend fondamentalement de l'ordre choisi pour ordonner les paires d'horloges. En fonction du modèle à vérifier, il n'est pas toujours possible de trouver un ordre pour lequel la méthode est efficace.

1.3.2 Ordres partiels

Une des causes de l'explosion combinatoire réside dans la représentation du parallélisme par l' entrelacement d'actions. L'idée des ordres partiels est d'éliminer autant que se peut les entrelacements inutiles lors de la construction de l'espace d'états.

La méthode est basée sur une analyse structurelle préalable des relations de dépendances entre les différentes transitions. La réduction exploite la commutativité des transitions concurrentes indépendantes, pour lesquelles l'état obtenu est le même quel que soit l'ordre d'exécution. Suivant le cas, seul un sous-ensemble de transitions sera localement exploré *stubborn sets* [26], les *persistent sets* [27] et les *ample sets* [28], soit, sous certaines conditions, on franchira en un seul pas atomique un ensemble de transitions indépendantes *pas couvrants* [29].

Il est possible de combiner la technique des *stubborn sets* et des pas couvrants, puis de construire le graphe de pas persistants [30].

Les techniques de réduction ordres partiels préservent l'absence de blocage et, sous certaines conditions, la structure linéaire ou arborescente de l'espace d'états.

1.3.3 Exploitation des symétries

La notion de symétrie d'un objet peut se définir comme la stabilité de cet objet vis à vis de certaines transformations. On dit que l'objet est *symétrique*. En géométrie par exemple, un polygone régulier est invariant par rotation et réflexion. On retrouve ce concept dans de nombreux domaines scientifiques ou artistiques comme l'illustre la Figure 1.6. Elle reproduit un tableau de Maurits Cornelis Escher, artiste peintre néerlandais du 20ème siècle, qui construisait ses oeuvres en utilisant des translations, rotations, réflections, ...

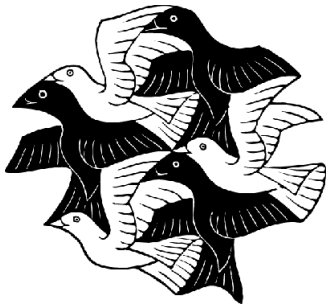


FIGURE 1.6: Vlakvullingsmotief met vogels, M.C Escher, 1949

L'intérêt pratique des symétries est qu'elles simplifient l'étude des objets symétriques : puisque l'objet est stable vis à vis de certaines transformations, certaines de ses propriétés le sont aussi. On peut réduire l'étude des propriétés aux parties de l'objet qui ne sont pas stables par ces transformations.

Supposons par exemple que l'on veuille s'assurer que tous les oiseaux de la Figure 1.6 possèdent deux ailes. On peut compter toutes les ailes dans le tableau, ou bien remarquer que le tableau est stable par rotation d'un couple d'oiseaux blanc et noir, illustré dans la Figure 1.7. Il suffit alors de compter les ailes de chacun des oiseaux d'un couple, et par symétrie, on déduit que tous les oiseaux ont deux ailes.

FIGURE 1.7: Vogels, réduit

Dans notre contexte, nous exploitons les symétries *structurelles* des systèmes. Intuitivement, un système est structurellement symétrique lorsque ses composants et leurs relations sont identiques modulo un identifiant. Nous pouvons utiliser ces symétries pour réduire l'espace d'états grâce au *principe de symétrie* [31] Ce principe énonce que :

Il y a au moins autant de symétries dans les effets que dans les causes

Par analogie, on identifie les *causes* avec la description du modèle du système et les *effets* avec les comportements du système, capturés dans l'espace d'états. Les symétries structurelles du système induisent des symétries sur le graphe de l'espace d'états. Les symétries du graphe, qui sont des automorphismes, induisent une relation d'équivalence sur ses sommets.

Les méthodes de réduction par symétries construisent un quotient de l'espace d'états par cette relation d'équivalence, réduisant sa taille par un facteur proportionnel à la quantité de symétries dans le système. Le cœur de cette thèse est une méthode de réduction par symétries pour les TPN, traitée en détail dans la Partie II.

Les méthodes de réduction par symétries préservent l'accessibilité, modulo symétrie, mais peuvent aussi préserver les formules de logique en temps linéaire [32].

1.4 FIACRE et TINA

Les sections 1.1 et 1.2.3 ont présenté les modèles formels permettant d'exprimer les propriétés ou les comportements. Dans cette section, nous présentons la boîte à outils TINA (TIme Petri Net Analyzer) permettant l'édition et l'analyse de réseaux de Petri temporels, le langage Fiacre – Format Intermédiaire pour les Architectures de Composants Répartis Embarqués – ainsi que la technique de vérification des descriptions Fiacre via TINA.

1.4.1 La boite à outils TINA

TINA (TIme Petri Net Analyzer [1]) est un environnement logiciel pour l'édition et l'analyse de réseaux de Petri et de réseaux de Petri temporels étendus par des priorités et des données. Sur ces modèles, TINA permet la vérification de propriétés par model-checking pour, notamment, la logique temporelle à temps linéaire State/Event LTL. Cette section présente un panorama général des fonctionnalités offertes par cet environnement, son architecture et les principales applications de la boîte à outils. Une description plus détaillée est disponible dans [33].

En plus des fonctionnalités usuelles d'édition et d'analyse offertes par des environnements comparables, TINA permet de construire différentes abstractions de l'espace d'états préservant des classes de propriétés telles que les propriétés générales d'accessibilité (absence de blocage, ...) ou des propriétés spécifiques telles que celles exprimables en logique temporelle à temps linéaire ou arborescent.

Pour les systèmes atemporels, les abstractions d'espaces d'états fournies permettent de limiter l'explosion combinatoire. Pour les systèmes temporisés, ces abstractions sont indispensables car leurs espaces d'états sont généralement infinis, TINA implémente ainsi différentes abstractions basées sur la notion de classes d'états.

Tina accepte en entrée des réseaux décrits sous forme textuelle, graphique ainsi que les descriptions $PNML$ (un format d'échange de réseaux de Petri, basé sur XML). En sortie, TINA produit des structures de Kripke étiquetées sous divers formats textuels ou binaires pour des vérificateurs de modèles natifs ou externes.

Constructions

Réseaux non temporisés Un premier groupe de méthodes fournies par TINA implémente les constructions classiques pour les réseaux de Petri : graphe des marquages accessibles, graphe de couverture (permettant de déterminer les places non bornées) et les techniques d'analyse structurelle (Invariants).

Un second groupe d'outils implémente les techniques d'ordres partiels des *stubborn set*, des *pas couvrants* et des *pas couvrant persistants* (Section 1.3.2).

Réseaux temporisés TINA implémente les abstractions des graphes d'états des TPN décrites dans la Section 1.2.3 : graphe des classes, inclusion, graphes des classes fortes et graphe des classes atomiques.

Architecture

L'architecture de TINA est schématisée figure 1.8. Le noyau du moteur d'exploration est paramétré par la classe de propriétés à préserver.

Les différents formats d'entrée sont convertis dans la représentation interne utilisée par l'explorateur de TINA. Le système de transitions de Kripke obtenu après l'exploration peut être fourni sous différents formats reconnus par le model-checker interne de TINA, ou des outils de vérification externes.

La figure 1.9 représente une session typique d'utilisation de TINA incluant une l'édition graphique d'un TPN et les représentations textuelle et graphique d'un graphe de classes.

Vérification de modèles (Model-Checking)

TINA peut fournir ses résultats sous différents formats, compatibles avec les formats d'entrée de model-checkers tels que MEC [34], acceptant des formules du μ-calculus, ou

1. `http://www.laas.fr/tina`

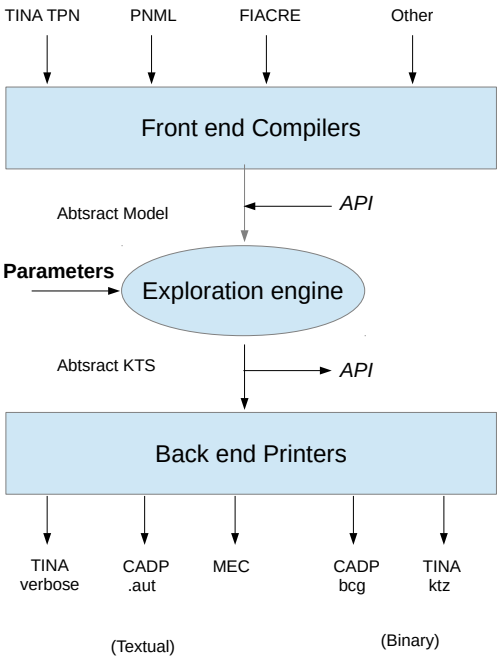


FIGURE 1.8: Architecture de TINA

encore la boîte à outils $CADP$ [35] pour les équivalences comportementales (bisimulation, ...). Pour finir, deux model-checkers natifs sont spécifiquement proposés dans TINA : `selt`, qui accepte les formules de $State/Event - LTL$ [36], notée $SE-LTL$, une logique temporelle linéaire dont les propositions atomiques peuvent être des propositions d'états et d'événements et `muse`, acceptant les formules du μ-calcul modal. Nous décrivons brièvement les caractéristiques du vérificateur `selt`.

$SE-LTL$ est une variante de LTL [36], qui permet de traiter de facon homogène des propositions d'états et des propositions de transitions. Les modèles pour la logique $SE-LTL$ sont des structures de Kripke étiquetées (ou SKE), aussi appelées systèmes de transitions de Kripke (ou KTS).

Quelques formules de `selt` concernant le réseau représenté dans la figure 1.9:

$t1 \wedge p2 \geq 2$
tout chemin commence par $t1$ et $m_0(p2) \geq 2$

$\Box\ (p2 + p4 + p5 = 2)$
un invariant de marquage linéaire

$\Box\ (p2 * p4 * p5 = 0)$
un invariant de marquage non linéaire

$infix\ q\ R\ p = \Box\ (p \Rightarrow \Diamond\ q)$
déclare l'opérateur «répond à», noté R

$t1\ R\ t5$
$t1$ «répond à» $t5$.

La vérification via `selt` comporte deux phases :

— construire un automate de Büchi acceptant les mots qui ne satisfont pas la formule $SE-LTL$ à vérifier ; cette phase est réalisée de facon transparente pour l'utilisateur en invoquant l'outil `ltl2ba` [2] [37] ;

— construire la composition de la structure de Kripke obtenue depuis le graphe des

2. `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba`

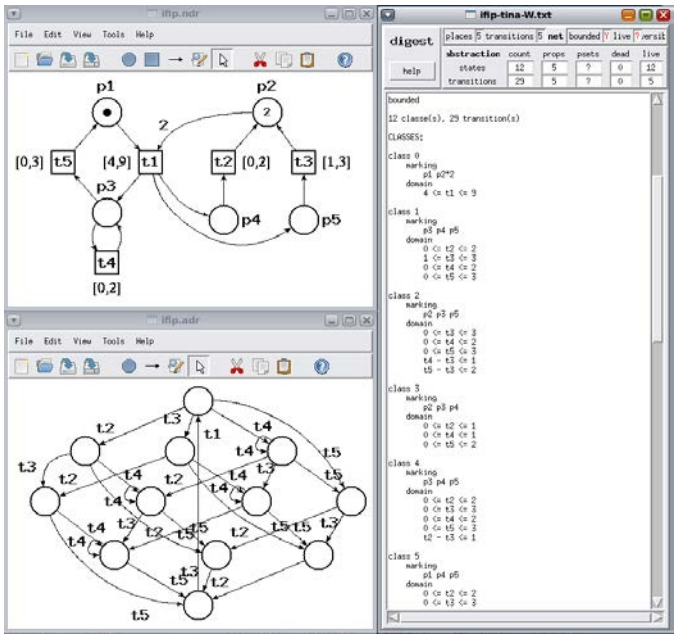


FIGURE 1.9: Une session de travail TINA

classes et de l'automate de Büchi, et rechercher à la volée une composante fortement connexe contenant un état acceptant de l'automate de Büchi. Si aucune telle composante n'est trouvée, alors la formule est satisfaite, sinon elle permet de définir un contre-exemple.

En cas de non-satisfaction d'une formule, `selt` peut fournir une séquence contre-exemple en clair ou sous un format exploitable par le simulateur de TINA, afin de pouvoir l'explorer pas à pas. Notons que, dans le cas d'un modèle temporisé, il faut au préalable associer à cette exécution un échéancier temporel. Celui-ci sera calculé de facon transparente par le module `plan` de la boîte à outils.

Il est aussi possible de vérifier des propriétés dites "temps réel", incluant des indications temporelles. Ces propriétés peuvent être vérifiées en utilisant la technique des observateurs : les propriétés temporisées sont traduites en propriétés LTL et la vérification est conduite sur le produit du modèle initial et de l'observateur associé à la propriété.

1.4.2 Le langage Fiacre

Fiacre [38] – Format Intermédiaire pour les Architectures de Composants Répartis Embarqués – est un langage formel pour la description de systèmes temps réel.

Il offre un cadre formel pour représenter, dans la même syntaxe, les aspects comportementaux ainsi que les contraintes temporelles d'un système. L'objectif est de pouvoir utiliser la spécification d'un système en Fiacre pour mener des activités de vérification formelle ou de simulation.

Fiacre a été initialement développé dans le cadre du projet AESE/Topcased, avec pour objectifs de servir de modèle "pivot" pour la traduction, en vue de leur vérification, de formalismes utilisateurs tels que AADL ou UML dans les modèles de bas niveau compris par les outils de vérification comme TINA ou CADP.

La conception de Fiacre s'inspire des résultats de recherche acquis sur les systèmes temps-réels et sur la théorie de la concurrence. Ainsi, le traitement des aspects temporels est emprunté au modèle des réseaux de Petri temporels [39] [40], les primitives de synchronisation et communication sont inspirées de celles des calculs de processus CSP et NT-Lotos [41] [42], tandis qu'on peut comparer l'intégration des contraintes temporelles et de priorités dans le langage à ce qui existe dans le cadre BIP [43]. Enfin, le langage

est fortement et statiquement typé, et le système de types est structurel (deux types sont équivalents si ils ont la même structure).

À des fins de vérification, les descriptions Fiacre, enrichies de déclarations de propriétés, peuvent être traduites par un compilateur dédié dans l'extension des réseaux de Petri temporels acceptée par la boîte à outils TINA. Les propriétés sont traduites par le compilateur en propriétés de logique temporelle à temps linéaire (donc vérifiables avec les outils disponibles), après instrumentation automatique des descriptions Fiacre par des observateurs pour les propriétés temporisées.

Programme Fiacre

La syntaxe du langage Fiacre est stratifiée par deux notions principales: les *processus*, qui décrivent les comportement de composants séquentiels ; et les *composants*, qui décrivent un système comme une composition de processus, éventuellement de manière hiérarchique. La Figure 1.10 montre un exemple de programme Fiacre modélisant l'exemple classique de l'anneau à jeton ("token ring"). Il s'agit d'un réseau d'unités de calculs, logiquement organisées dans une topologie en anneau, qui synchronisent leurs communications par le biais d'un jeton qui circule parmi eux et permet de contrôler l'accès aux canaux de communication.

```
1  process Start
2    [start0: none, start1: none, start2: none] is
3    states s0, s1
4    from s0 select
5       start0
6    [] start1
7    [] start2
8    end;
9    to s1
10
11 process Node
12   [prev : none, succ : none, start : in none] is
13   states idle, waitcs, cs, st1
14   from idle select
15      start; to st1
16   [] prev; to st1
17   end
18   from st1 succ; select to idle [] to waitcs end
19   from waitcs prev; to cs
20   from cs   succ; to idle
21
22 component root is
23   port s0 : none, s1 : none, s2 : none,
24        p0 : none, p1 : none, p2 : none
25   par * in
26           Start[s0,s1,s2]
27         || Node[p0, p1, s0]
28         || Node[p1, p2, s1]
29         || Node[p2, p0, s2]
30         end
31
32 root
```

FIGURE 1.10: Un exemple de programme Fiacre: l'anneau à jeton

Un programme Fiacre est un séquence de déclarations. Plus particulièrement : des déclarations de types, pour décrire des contraintes sur les valeurs des variables locales ou échangées entre processus ; et des déclarations de processus et de composants, pour décrire le comportement des éléments du système. Fiacre est un langage fortement typé, ce qui signifie que des annotations de type sont exploitées afin de garantir la compatibilité des données manipulées par les processus (par exemple, on ne pourra pas envoyer une entier sur un port là où une valeur booléenne est attendue).

Processus Fiacre Un processus est défini par un ensemble d'états de contrôle et de paramètres. Chaque état est associé à une *macro-transition*, c'est-à-dire une expression complexe qui décrit l'ensemble des transitions ayant cet état pour origine. La macro-transition décrit également de quelle manière les paramètres sont mis à jour après chaque transition. Par exemple, la déclaration suivante définit que le processus `T` peut interagir sur deux ports : `p`, qui transmet des valeurs booléennes, et `q`, qui ne peut être utilisé que pour la synchronisation. Le processus possède également deux paramètres: `v`, qui est un entier, et `u`, qui est une référence (pour une variable) partagée pour un tableau de booléens de taille 5.

```
process T
  [p : bool, q : none]
  (v : int, &u : array 5 of bool) is
  ...
```

Les macro-transitions sont définies par un langage d'expressions bâti au dessus "d'opérateurs déterministes" classiques, que l'on retrouve dans la plupart des langages de programmation (affectations, conditionnelles, boucles while et foreach, composition séquentielle), et d'opérateurs non déterministes comme le choix et la communication par événements sur les ports de communication. Par exemple, la déclaration:

```
from s0
   select
       p!5; to s1
   []  x := x + 1; to s2
   end
```

exprime le fait que, dans l'état `s0`, le processus peut choisir (de manière non-déterministe) entre deux alternatives: soit envoyer la valeur `5` sur le port `p` puis entrer dans l'état `s1` ; ou bien incrémenter la valeur de la variable `x` et entrer dans l'état `s2`.

Un processus, à la manière d'une classe dans un langage à objets, définit le comportement de base d'un élément du système. Les processus peuvent être instanciés et mis en relation au sein de composants, qui décrivent l'architecture du système que l'on souhaite modéliser.

Composant Fiacre Un composant est défini comme la composition parallèle de processus et/ou d'autres composants. Cet opération est dénotée par la construction `par ... || ... end`. Les composants représentent à la fois l'unité de composition d'une spécification Fiacre, ainsi que l'unité d'instanciation des processus, et de création des ports et des variables partagées. La syntaxe des composants permet de restreindre le mode d'accès et la visibilité des variables partagées et des ports. Elle permet aussi d'associer des contraintes temporelles sur les interactions et de définir des priorités entre les interactions. Par exemple, dans un composant `C`, la déclaration `port p : none` définit un port appelé `p` qui est privé pour (ne peut pas être utilisé en dehors de) `C`. De même, la déclaration `port p : none in [min ,max]` définit un port qui ne peut interagir que `min` unités de temps après qu'il ait été activé et doit être utilisé ou désactivé avant `max` unités de temps (`min` et `max` sont des flottants ou des constantes entières).

Chaîne de vérification Fiacre

La figure 1.11 représente la chaîne de vérification Fiacre initialement développée dans le cadre du projet Topcased. L'architecture commune des compilateurs permet de factoriser au plus la chaîne de traduction vers les outils CADP et TINA. Une partie avant (FRONT) du compilateur, partagée, assure l'analyse lexicale, syntaxique, le typage, et la vérification d'un certain nombre de contraintes statiques. Toujours dans un souci de factoriser au plus la phase de traduction, l'outil FRONT embarque certaines procédures génériques d'élimination de constructions dérivées (expressions conditionnelles, par exemple) utilisées

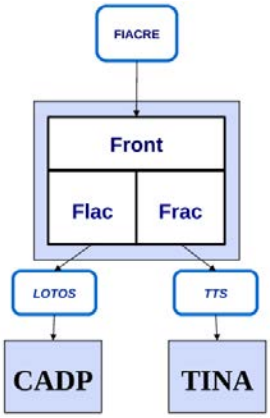


FIGURE 1.11: Architecture de compilation Fiacre

par l'un ou d'autre des générateurs. FRONT produit un arbre abstrait qui constitue le point d'entrée des générateurs FLAC (Fiacre to Lotos Adaptation Component) produisant du code LOTOS pour la connexion à CADP, et FRAC produisant des modèles au format "Time Transition Systems" (TTS) pour la connexion à TINA.

Des versions pré-compilées de FRONT, FLAC et FRAC pour diverses cibles sont disponibles sur leurs sites respectifs [3].

Depuis une description Fiacre, le compilateur `frac` produit un Système de transitions temporels, une extension avec données et priorités des TPN, au format attendu par $TINA$. Une description Fiacre ainsi compilée peut être vérifiée avec les mêmes outils que tout autre modèle accepté par $TINA$. On peut en particulier construire son graphe de classe (SCG ou $SSCG$) et vérifier sur cette abstraction des propriétés $SE-LTL$ à l'aide du vérificateur `selt`.

Le compilateur `frac` admet dans les descriptions Fiacre des déclarations de propriétés ainsi que des directives de vérification. Ces propriétés et directives sont automatiquement traduites par le compilateur `frac` en un fichier de commandes utilisé par le vérificateur `selt` pour mener à bien la vérification.

Pour simplifier l'expression des propriétés, celles-ci peuvent aussi être exprimées à l'aide de *patrons* (*patterns*), à la façon de Dwyer [44] [4].

Les patrons supportés par `frac` incluent des propriétés prédéfinies, comme l'absence de blocage ou l'équité, et la plupart des patrons de [44]. L'utilisateur peut définir lui même d'autres propriétés. L'implantation de ces patrons se base sur les mécanismes de sondes et d'observateurs qui ont été introduits dans la version 1.7 de FRAC. Une présentation plus détaillée de cette approche peut être trouvée dans [45].

3. `http://www.laas.fr/fiacre` pour FRAC
4. `http://patterns.projects.cis.ksu.edu`

Chapitre 2

Model-checking dans un contexte avionique

Sommaire

Dans ce chapitre, nous présentons comment les méthodes de vérification formelles peuvent être utilisées pour de la vérification des systèmes avioniques. Notre attention se porte plus particulièrement sur le model-checking. Des exemples de cas d'utilisation tirés de la littérature sont présentés et nous détaillons une exercice de vérification de propriétés temps réel mené dans le cadre de cette thèse sur une fonctionnalité d'un système de gestion automatique de vol. Cette expérimentation montre que le model-checking peut être utilisé pour garantir une propriété de sûreté dans une architecture Command/Monitoring.

Gravité	Effet de la défaillance	Probabilité
Catastrophique	L'avion ne peut voler de façon sûre, perte de l'avion, de l'équipage ou de passagers	$10^{-9}/hv$
Dangereux	Réduction importante des marges de sécurité ou des fonctions, augmentation importante de la charge de travail pour l'équipage, blessures sévères	$10^{-7}/hv$
Majeur	Réduction significative des marges de sécurité ou des fonctions, augmentation de la charge de travail pour l'équipage, inconfort, blessures possibles	$10^{-5}/hv$
Mineur	Réduction légère des marges de sécurité ou des fonctions, augmentation de la charge de travail pour l'équipage, inconfort	$10^{-3}/hv$

TABLE 2.1: Classification des défaillances par heure de vol

2.1 Systèmes avioniques

Les systèmes avioniques sont des systèmes électroniques, électriques et informatiques embarqués dans un avion et indispensable à son fonctionnement. Ce chapitre se concentre sur les systèmes informatiques qui sont, dans ce contexte, temps réel et critiques quant à leur sûreté de fonctionnement.

Les systèmes temps réel sont des systèmes où une opération est correcte si elle délivre le bon résultat et qu'elle le délivre au bon moment. La dimension temporelle est très importante. Par exemple, lorsqu'un pilote actionne une manette de gaz électrique, le système doit calculer la bonne puissance, toujours avec le même délai.

Les systèmes critiques sont des systèmes où une erreur peut avoir des conséquences catastrophiques : perte de mission ou perte de vies humaines. Une faute mal gérée dans un système avionique peut entraîner la perte de l'avion et des passagers. En conséquence la sûreté de fonctionnement, qui passe notamment par le respect des contraintes temporelles, est un objectif essentiel et structurant dans la réalisation de ces systèmes.

2.1.1 Contraintes règlementaires, processus de développement

Les systèmes avioniques et plus généralement les avions qui les embarquent ont un très fort impact social. La responsabilité des avionneurs est très importante et il est légitime que les citoyens exercent un contrôle sur eux. Ce contrôle se fait à travers des autorités de certification, l'European Aviation Safety Agency(EASA) en Europe et la Fededral Aviation Administration(FAA) aux États-Unis. Ces institutions s'assurent pour le compte de la société civile que les avions utilisés au quotidien par des millions de personnes sont sûrs.

Les autorités de certification définissent des objectifs de certification qui garantissent autant que possible la sûreté des avions construits. Dans l'aéronautique, l'approche utilisée est basée sur les standards. Elle repose sur la prise en compte de l'expérience accumulée par la communauté industrielle (notamment à travers les enquêtes après accident) [46].

Une des difficultés de l'activité de certification est que la qualité d'un avion est difficilement quantifiable. Les autorités l'évaluent et la mesurent à travers un ensemble de contraintes sur chacun des processus impliqués. Ces contraintes doivent être respectées par l'avionneur tout au long du cycle de vie de l'avion.

Au niveau le plus haut, on trouve le système "avion" qui implémente des *fonctions*. La première des fonctions d'un avion est de voler. Le système avionique est directement impliqué dans les fonctions de commande de vol, de pilote automatique, de gestion du carburant ... L'évaluation de sûreté à ce niveau analyse les défaillances possibles des fonctions : dys-

fonctionnement, fonctionnement intempestif ou perte. L'analyse, guidée par l'ARP-4761 [47], établit la criticité d'une fonction par rapport à la gravité de ses défaillances. C'est une analyse de sûreté du système. A chaque niveau de criticité est associée une probabilité d'occurrence tolérée. La Table 2.1 montre quelques exemples de niveau de gravité, leur conséquence et leur probabilité tolérée.

De nombreux critères influent sur la conception d'un avion : le poids, la consommation, le nombre de passagers qu'il peut embarquer,... Cependant et quelque soit ces critères, il est indispensable de garantir la sûreté de fonctionnement. En ce sens, la sûreté de fonctionnement est un critère structurant pour la conception du système avionique : le système est raffiné avec comme objectif de minimiser les risques identifiés. Il s'agit alors de réduire la fréquence d'occurrence ou la gravité des défaillances, en introduisant par exemple des mécanismes de tolérance aux pannes ou de redondances. Ce processus de raffinement est guidé par les recommandations de l'ARP-4754 [48].

Le raffinement s'arrête lorsque les éléments de base du système, appelés *composants*, sont tous identifiés. Ce processus produit un ensemble d'exigences de haut niveau (HLR), fonctionnelles ou de sûreté, qui spécifient les composants et leurs interactions. Chacun de ces composants hérite du niveau de criticité de la fonction qu'il contribue à réaliser. A ce niveau d'abstraction, les composants sont supposés *corrects*, c'est à dire conforme à leurs exigences. Nous ne nous intéresserons qu'aux niveaux d'abstraction inférieurs à la sortie du processus de raffinement. Le problème est alors de prouver que les composants sont corrects.

Dans ce chapitre, focalisé sur la dimension informatique d'un système avionique, un composant est soit matériel soit logiciel. Ici encore, le processus de réalisation des composants est guidé par des contraintes pour la certification. Un composant logiciel ne peut être certifié que s'il atteint les objectifs définis par la norme DO-178 [49] tandis qu'un composant matériel doit atteindre ceux définis par la norme DO-254 [50]. Nous ne nous intéressons qu'aux composants logiciels.

Comme pour le système, le processus de développement du composant est structuré par son niveau de criticité, appelé Design Assurance Level(DAL). La norme DO-178 définit 5 DALS, de A à E. Le niveau A, catastrophique, est le plus sévère. Une défaillance dans un logiciel de niveau A empêche le vol ou l'atterrissage. Par exemple, les composants impliqués dans la fonction de commande de vol ont un DAL A. A l'opposé, une défaillance dans un logiciel de niveau E n'a aucune conséquence.

A chaque niveau de criticité correspond un ensemble d'objectifs et de moyens de prouver que ces objectifs ont été atteints. La norme DO-178 proposait jusqu'à une évolution récente trois moyens de preuves: la revue, l'analyse et le test. Pour chaque objectif il faut prouver qu'il est atteint en utilisant une combinaison d'un ou plusieurs de ces moyens. Mais la norme n'est pas prescriptive, le choix du ou des moyens de preuve est laissé à l'industriel.

Le processus de développement est un processus de raffinement. La spécification du composant par ses HLR est successivement raffiné jusqu'au code objet qui sera exécuté sur un composant matériel embarqué. Dans un premier temps, les HLR sont raffinées jusqu'à des exigences de bas niveaux (LLR). Un des objectifs à atteindre est la traçabilité totale entre les HLR et les LLR. Cette traçabilité est généralement prouvée par des revues. Le code source est ensuite écrit à partir des LLR puis compilé vers du code objet. Un autre objectif de la DO est de montrer l'équivalence du code objet avec les LLR. Comme la norme n'est pas prescriptive, plusieurs options sont possibles. Une première option peut être de montrer par analyse l'équivalence du code source par rapport à ses LLR, puis de montrer que le code source est équivalent au code objet en utilisant un compilateur certifié. Une deuxième option est de montrer que le code objet est équivalent aux LLR par des tests. Dans ce cas, il faut prouver que les objectifs de couverture du code sont atteints.

Les outils utilisés dans ce processus (eg: le compilateur) sont eux aussi contraints par la norme. Deux types d'outils sont considérés :

— Les *outils de développement* sont les outils qui peuvent introduire des erreurs dans le programme, par exemple les compilateurs. Ces outils héritent du même niveau de criticité que les logiciels qu'ils aident à développer ;

— Les *outils de vérification* sont les outils qui peuvent échouer à détecter des erreurs, par exemple les simulateurs. La criticité est moins élevée et les contraintes de certification moins fortes.

Nous concluons cette section par un mot sur une évolution récente de la DO-178. La version B a été publiée en 1992 et ne tenait pas compte des méthodes de vérification formelle. Les progrès de ces moyens au cours de ces vingt dernières années tout autant que l'évolution des pratiques des industriels dans ce sens ont fait évoluer la situation. Très récemment, en 2011, la version C de la norme a été publiée. Cette version intègre l'utilisation des méthodes formelles comme moyen de preuve des objectifs de certification. L'utilisation des ces nouveaux outils est très encadrée mais cette évolution offre des opportunités aux méthodes formelles. Les conditions d'utilisation des méthodes formelles sont décrites dans un document compagnon, la norme DO-333. Nous présentons dans les sections suivantes différentes utilisations des méthodes formelles pour la vérification des systèmes avioniques.

2.1.2 Avionique Modulaire Intégrée

L'architecture *avionique modulaire intégrée*, aussi appelée IMA pour Integrated Modular Avionic, permet à des composants logiciels, de partager des ressources de composants matériels. C'est une architecture constituée de composants matériels, de composants logiciels et d'un réseau de communication. Nous nous conformerons à la terminologie IMA en appelant *fonctions* les composants logiciels et *modules* les composants matériels. L'IMA est apparue dans les années 90 et est utilisée par Airbus depuis l'A380 et par Boeing depuis le B777.

Un des avantage d'IMA est de permettre à des fonctions de niveau de criticité différents de partager des modules. C'est ce que l'on appelle la *ségrégation*. La ségrégation des fonctions a permis de nombreux progrès dans l'avionique : gains de poids, optimisation du processus industriel, standardisation des modules, ... La ségrégation des différentes *fonctions* repose sur les concepts de *liens virtuels* du réseau AFDX et *partition* du système d'exploitation ARINC-653.

Une partition s'exécute sur un module à travers le système d'exploitation ARINC-643. Elle est définie par une zone mémoire et une tranche de temps périodique, pendant laquelle elle est la seule à s'exécuter sur le module [51]. Le temps d'exécution d'une partition est borné par un pire cas de temps d'exécution (WCET). Nous supposons, à des fins de simplification, qu'une fonction est toujours implémentée par une seule partition. Plusieurs partitions peuvent s'exécuter sur un module. Ces partitions sont ordonnancées statiquement et de façon strictement périodique. Une trame cyclique englobante, appelée MAF (MAjor time Frame), décrit l'ordonnancement de toutes les partitions du module. La période de la MAF est l'hyper-période des périodes des partitions. L'*onset* d'une partition est la phase de la partition par rapport à celle de la MAF. L'ordonnancement est réalisé par le système d'exploitation du module.

Un lien virtuel est un mécanisme de communication offert par le réseau AFDX qui permet la ségrégation de flux provenant de fonctions de différents niveaux de criticité. Le réseau AFDX pour Avionics Full-DupleX, est un réseau dérivé de l'Ethernet commuté. Il est composé de commutateurs et à ses extrémités de producteurs/consommateurs de données, appelés *abonnés*. Un lien virtuel est vu par les fonctions comme un bus virtuel

mono-émission/multi-réception qui garantit une bande passante, une latence et une gigue bornées.

Les partitions communiquent à travers des ports de communication unidirectionnels, appelés *APEX*. Il existe deux types de ports :

— Port *queuing* : le port se comporte comme une FIFO,

— Port *sampling* : seule la dernière donnée reçue est conservée.

Ces ports permettent aux partitions de communiquer via le réseau AFDX ou via des bus de terrain. Ces bus de terrain connectent les capteurs et actionneurs des systèmes mécaniques, électriques ou hydrauliques avec le reste du système avionique.

Les modèles d'exécution et de communication des architectures IMA sont spécifiés par les standards ARINC653 [52] et ARINC664 (partie 7) [53].

Langages synchrones et architectures GALS

Dans cette section, nous présentons très rapidement le langage synchrone Lustre et le concept d'architecture *Globalement Asynchrone Localement Synchrone* (GALS). Ce patron d'architecture correspond précisément à l'avionique modulaire intégrée.

Les langages synchrones regroupent une famille de langages de programmation qui combinent les notions de synchronisme et de concurrence. C'est un choix technologique très répandu pour la spécification fonctionnelle et l'implémentation d'applications temps réel embarquées. En particulier le langage Lustre est utilisé dans la conception des systèmes avioniques par Airbus et ses principaux équipementiers.

Dans le modèle synchrone, le temps est discrétisé et les temps de calcul et de communication sont négligés, considérés comme infiniment rapides. Un programme synchrone P est une succession de *réactions atomiques* : $P \equiv R^\omega$, où R est l'ensemble des réactions atomiques et R^ω dénote une itération infinie de réactions. Ces dernières sont des fonctions déterministes de la forme $(\text{état courant}, \text{entrées}) \mapsto (\text{état suivant}, \text{sorties})$. La Figure 2.1.1 montre un schéma d'exécution synchrone, utilisé par le langage Lustre, qui décrit une itération infinie de réactions.

Algorithme 2.1.1 Modèle d'exécution synchrone

```
Initialiser la mémoire
for each top horloge do
    Lire les entrées
    Calculer les sorties
    Mettre à jour la mémoire
end for
```

Lustre Lustre est un langage synchrone à flot données. Un programme Lustre est une boucle infinie où les variables sont définies une seule fois par des équations. La définition de $x = y + z$ implique $x_k = y_k + z_k$ à chaque itération k de la boucle. Une variable est une fonction du temps et représente un flot de données : une séquence infinie de valeurs de son type [54].

L'opérateur *pre* désigne la valeur à $k-1$ d'un flot, pour $k > 0$. Si $k = 0$ alors $pre(x)$ vaut *nil*, une constante spéciale indiquant que la valeur n'est pas définie. L'opérateur $\rightarrow$ définit des valeurs par défaut : le flot $a \rightarrow b$ prend la valeur du flot a à sa première itération et la valeur du flot b ensuite. Ces deux opérateurs permettent de construire les autres opérateurs du langage. Les programmes Lustre sont structurés par des *noeuds*.

Un flot n'est actif que si son horloge l'est. Chaque programme Lustre est associé à une horloge de référence. Chaque flot est attaché à une horloge, soit celle du programme

soit une horloge (plus lente) définie par rapport à l'horloge de référence. Un flot n'a de valeur que si son horloge est vraie, si l'horloge est fausse, le flot est indéfini. Par exemple, si x est un flot et c est une horloge, alors le flot x when c est un flot qui prend une valeur de x lorsque c est vraie mais n'est pas défini lorsque c est fausse. Ce mécanisme d'horloge permet d'activer des parties du programme à des fréquences différentes. Pour des raisons pratiques, un mécanisme de condition d'activation a été ajouté. A la différence des horloges, un flot a une valeur définie même lorsque la condition est fausse. Tant que la condition est fausse, la valeur du flot est figée (ou prend la valeur par défaut du flot jusqu'à la première activation).

Déploiement sur une architecture GALS

Les programmes synchrones sont compilés puis déployés dans une architecture IMA. Une architecture IMA est globalement asynchrone : il n'y a pas d'horloge globale ni d'état global et l'AFDX est asynchrone. Un tel système se comporte comme un réseau de noeuds localement synchrones qui communiquent via des réseaux asynchrones : IMA est un système GALS.

Les programmes synchrones permettent de décrire les composants logiciels. Des générateurs de code produisent le code des partitions qui les implémentent. D'une certaine manière, les programmes synchrones peuvent être vus comme des exigences de bas niveau. La compilation garantit que la sémantique fonctionnelle est préservée : la partition est conforme, fonctionnellement, à sa spécification. Néanmoins, les durées d'exécution des partitions et les temps de communication ne sont pas négligeables : les architectures IMA ne vérifient pas l'hypothèse de synchronisme.

Le déploiement des programmes synchrones dans une architecture GALS est un problème connu [54]. En pratique, le déploiement est possible au prix d'une estimation très fine des temps d'exécution des partitions et de l'introduction d'exigences non fonctionnelles dans la spécification système. Ces exigences spécifient notamment les contraintes temporelles induites par les asynchronismes. Nous donnons dans ce chapitre un exemple de vérification dans une architecture GALS, qui montre que le model-checking temps réel est très efficace pour la spécification et vérification de ces exigences non fonctionnelles, indissociable des systèmes avioniques tels qu'ils sont réalisés aujourd'hui.

2.1.3 Vérification formelle dans le contexte avionique

Cette section présente des exemples d'utilisation opérationnelle, dans un contexte avionique. Bien que ne se limitant pas au model-checking, la plupart des expérimentations présentées utilisent cette technique.

Afin de structurer cette section, nous définissons deux niveaux d'application pour la vérification formelle :

1. *Le niveau système* : correspond à la spécification des composants et de leurs interactions. Cette spécification est exprimée par des HLR, construites à partir de l'analyse système, et les LLR obtenues par raffinement des HLR. Ces LLR peuvent être sous forme textuelle ou sous la forme de programme Lustre.
2. *Le niveau logiciel* : correspond au code source et objet des composants, par exemple en langage C. Les HLR du logiciel sont les LLR du composant qu'il implémente. Le code source d'un composant est écrit à partir de LLR déduites de ses HLR. Le code source peut aussi être généré à partir des LLR du composant si ces dernières sont écrites en Lustre. Le code objet est obtenu par compilation du code source.

Dans l'industrie, la vérification d'un logiciel consiste à vérifier que le code exécutable du logiciel est conforme à ses exigences. Plus le logiciel est critique, plus la part de la

vérification dans le coût du logiciel est importante. Cette activité représente donc une part très importante du coût des logiciels dans l'industrie aéronautique.

Autres méthodes de vérification

Le cœur de cette thèse est la méthode du model-checking. Néanmoins, nous nous attachons à donner des exemples d'utilisation des méthodes formelles dans notre contexte industriel, pas uniquement du model-checking. Deux des méthodes de vérification les plus utilisées sont la preuve de programme et l'interprétation abstraite.

Preuve automatique de programme La preuve de programme recouvre un ensemble de techniques permettant de prouver qu'un programme satisfait sa spécification par des déductions logiques. La correction d'un programme et sa sémantique sont formulées par des théorèmes. Ces théorèmes sont généralement prouvés semi-automatiquement par des outils d'aide à la preuve. Parmi les outils les plus connus on peut citer Coq [55], PVS [56] ou Isabelle [57].

Un programme et son environnement d'exécution sont modélisés dans un formalisme approprié. Le comportement attendu du programme est décrit avec le même formalisme. Le modèle ainsi obtenu peut être transformé vers un ensemble de formules logiques. La preuve de correction est alors synthétisée puis vérifiée automatiquement par un outil d'aide à la preuve à partir d'un ensemble d'axiomes et de règles d'inférences.

Interprétation abstraite L'interprétation abstraite [58, 59, 60] est une théorie des approximations des sémantiques utilisée pour l'analyse et la vérification statiques des logiciels. Elle peut être définie comme une exécution partielle d'un programme pour obtenir des informations sur sa sémantique (par exemple, sa structure de contrôle, son flot de données) sans avoir à en faire le traitement complet. Sa principale utilisation est l'analyse statique, l'extraction automatique d'informations sur les exécutions possibles d'un programme. Ces analyses ont deux usages principaux :

— Pour les compilateurs, afin d'analyser le programme pour déterminer si certaines optimisations ou transformations sont possibles ;

— pour prouver l'absence de certains types d'erreurs dans un programme, appelées *erreurs à l'exécution* et notées RTE.

Dans un contexte avionique, les contraintes de certification interdisent la plupart, sinon toutes, des optimisations réalisées par le compilateur. Ce qui va nous intéresser dans cette section, ce sont surtout les capacités de détection d'erreurs RTE.

Vérification au niveau logiciel

La méthode de vérification la plus utilisée aujourd'hui est le test. Un test est composé d'un scénario, d'un environnement et d'un résultat attendu. Il est réussi si, après l'exécution du logiciel décrite par le scénario dans l'environnement de test, le résultat obtenu est identique au résultat attendu. Deux grandes familles de tests coexistent. Le *test unitaire*, qui permet de vérifier la conformité d'un logiciel par rapport à ses LLR, et le test fonctionnel qui permet de vérifier la conformité par rapport à ses HLR. Une analyse de *couverture structurelle*, qui détecte les parties du logiciel couvertes par les tests, permet de prouver (aux autorités) que le logiciel est suffisamment testé. Plus le niveau de criticité est élevé plus la couverture doit être large : la totalité du code source d'un logiciel de criticité DAL-A doit être testée.

Dans [61], les *tests* unitaires d'un logiciel sont avantageusement remplacés par des *preuves unitaires* qui reposent sur l'outil Caveat [62], un outil de preuve automatique de

programme. Le logiciel est écrit avec un sous-ensemble de C pour lequel Caveat définit une sémantique formelle. Les LLR du logiciel sont formalisés grâce au langage formel proposé par Caveat. Ces LLR formalisées et le code de chaque fonction du logiciel sont analysés par Caveat. Ce dernier vérifie l'équivalence des spécifications et du code source en utilisant une approche déductive. L'activité est semi-automatique : l'utilisateur doit intervenir pour résoudre des obligations de preuves.

La preuve unitaire n'est pas suffisante car il faut prouver l'équivalence du code objet avec ses HLR. Le test fonctionnel reste indispensable pour couvrir le risque d'erreurs introduites à la compilation. Des travaux sont en cours pour l'utilisation d'un compilateur certifié qui permettrait de réduire le nombre de tests fonctionnels nécessaires.

La vérification d'un logiciel nécessite aussi de prouver l'absence de RTE. Le test peut ne pas être suffisant et la preuve doit alors se faire par analyse, souvent très fastidieuse. L'outil Astrée [63], basé sur les résultats théoriques de l'interprétation abstraite, est utilisé pour garantir l'absence de RTE à partir de sources C.

La conception du système dans lequel le logiciel s'exécute repose sur une hypothèse de synchronisme (Section 2.1.2). Il faut prouver que l'implémentation du système vérifie cette hypothèse : le système doit être capable de réagir à un évènement reçu avant l'arrivée d'un nouvel évènement. La connaissance précise des pires temps d'exécution d'un logiciel (WCET) est nécessaire pour prouver que l'hypothèse de synchronisme est vérifiée [64].

Cependant, les architectures modernes des microprocesseurs utilisés dans l'avionique induisent du non-determinisme temporel. Par exemple, le temps d'accès à une donnée varie selon que la donnée est en cache ou en mémoire. Les méthodes classiques d'analyse, manuelles, deviennent impossibles à appliquer.

La solution retenue dans [64] est l'utilisation de techniques d'interprétation abstraite et l'outil aiT. Une approximation très fine de la cible d'exécution est construite. A partir de cette approximation, l'outil aiT est capable de calculer un WCET suffisamment précis et optimiste.

Ce dernier exemple montre comment des méthodes formelles peuvent venir compléter l'approche de conception synchrone des systèmes par l'évaluation quantitative des contraintes temporelles. De plus, il montre que ces méthodes formelles tendent à devenir indispensables, moins pour des gains éventuels de productivité que pour les conséquences de la complexité croissante des matériels, qui empêchent l'utilisation des méthodes manuelles classiques.

Il est important de noter que toutes ces techniques sont utilisées opérationnellement, c'est d'ailleurs notre motivation à les présenter. Les premières expérimentations ont démarré au début des années 2000 et aujourd'hui ces techniques formelles sont complètement intégrées au processus industriel. Leurs utilisations contribuent directement à la certification en complément des moyens de vérification plus classique de la DO-178 que sont le test, l'analyse et la revue. La DO-333 capitalise sur ces expériences et normalise les usages possibles des méthodes formelles pour prouver l'atteinte des objectifs.

Model-checking au niveau logiciel

Cette section donne quelques exemples d'utilisation du model-checking pour la vérification de logiciel. L'avantage du model-checking, par rapport à la preuve de programme et l'interprétation abstraite, est qu'il permet de détecter les erreurs de concurrence (parfois appelées erreurs non déterministes). Elles sont généralement très difficiles à reproduire. Il est donc difficile, si ce n'est impossible, de les détecter par le test, l'analyse ou la revue [65]. Le model-checking est donc un outil adapté à la vérification de logiciels concurrents.

L'expérimentation décrite dans [66] se situe dans le contexte d'un système d'exploitation temps réel pour une architecture IMA. Les auteurs cherchent à vérifier des propriétés de partitionnement temporel de l'ordonnanceur : à chaque cycle de l'ordonnanceur, le budget CPU alloué à un thread doit lui être complètement attribué. Le logiciel à vérifier est écrit avec un sous-ensemble de C++. Le code du logiciel est converti automatiquement vers Promela, le langage de spécification du model-checker Spin [1]. La propriété d'allocation totale est vérifiée par Spin.

Dans d'autres contextes que l'avionique, [67] décrit l'utilisation de Spin pour la détection d'erreurs de concurrence dans un logiciel embarqué de la mission Curiosity. Dans [68] les auteurs utilisent Spin pour vérifier un logiciel de contrôle autonome de mission spatiale. Le logiciel traité est écrit en LISP [69]. Finalement, l'outil SLAM [70] inclut un model-checker pour la vérification des pilotes matériels pour le système Windows. Cet outil a eu un succès considérable, puisqu'il a grandement contribué à la fin du fameux *écran bleu* ©.

Ces exemples de vérification logicielle par model-checking mettent en avant la capacité du model-checking à détecter les erreurs de concurrence. Les conséquences de ce type d'erreurs peuvent être catastrophiques. La vérification formelle par model-checking apporte une plus-value par rapport aux activités traditionnelles de tests, revues et analyses qui peut justifier l'effort conséquent de sa mise en oeuvre. La complexité croissante des logiciels, de plus en plus concurrents, augmente la probabilité de ce type d'erreurs, réduit l'efficacité des méthodes traditionnelles de vérification et tend à justifier une plus grande utilisation du model-checking.

A notre connaissance, la vérification de logiciel par model-checking n'est pas utilisée opérationnellement dans l'aéronautique.

Model-checking niveau système

La conception des systèmes avioniques repose en partie sur l'utilisation de modèles formels. Parmi les outils de modélisation les plus répandus on trouve Scade [2], Simulink [3] et Stateflow [4]. Scade est la version commerciale de
LUSTRE. Simulink/Stateflow est un outil de modélisation qui combine la modélisation des systèmes dynamiques et à évènements discrets.

Ces modèles décrivent ce que nous appellerons dans cette section le niveau système. Ils permettent de donner une sémantique formelle aux composants et à leurs interactions : le modèle Lustre d'une fonction donne les LLR des composants qui l'implémentent. L'utilisation des modèles s'est généralisée grâce à la simplification des processus qu'elle permet, notamment par la génération de code certifié.

La technique de vérification des modèles la plus utilisée est le test. Les tests fonctionnels, élaborés à partir des HLR du système, sont exécutés sur une plateforme de simulation. Cette plateforme exécute le code généré automatiquement à partir du modèle, autrement dit le code implémentant les LLR du système.

Le test permet de vérifier les comportements nominaux, la robustesse du système, le fonctionnement en cas de panne, ... L'inconvénient majeur de cette approche est lié à la nature du test : il n'est pas exhaustif [71]. La conséquence est que des erreurs dans la spécification du système peuvent ne pas être détectées et se propager au niveau des composants logiciels, ce d'autant plus qu'une grande partie du code est généré automatiquement à partir du modèle. Il est bien connu que plus une erreur est introduite tôt dans le cycle de développement, plus elle sera coûteuse à corriger.

1. `http://spinroot.com/spin/whatispin.html`
2. `http://www.esterel-technologies.com/products/scade-suite/`
3. `http://fr.mathworks.com/products/simulink/`
4. `http://fr.mathworks.com/products/stateflow/`

Un autre problème du test au niveau système est lié à l'architecture GALS de ces systèmes. Les asynchronismes inhérents à cette architecture engendrent des fautes de concurrence. Or ces fautes sont très difficiles à détecter par le test. Il y a donc des opportunités pour les méthodes formelles lorsqu'elles sont appliquées au niveau système.

Dans [72], les auteurs décrivent l'effort de vérification formelle mis en oeuvre pour un système d'affichage d'informations de vol critiques, l'"Adaptive Display & Guidance System", développé par Rockwell Collins Inc.

Les modèles sont réalisés avec Simulink/Stateflow. Ils sont ensuite transformés en Lustre puis vers le langage d'entrée du model-checker NuSmv [5], un model-checker symbolique. On obtient ainsi un modèle de vérification des LLR du système.

Les HLR sont formalisés par des formules de logique temporelle CTL. Ces formules logiques sont vérifiées par NuSmv sur le modèle de vérification. Cette démarche permet donc de vérifier les HLR du système sur un modèle formel de ses LLR. C'est précisément ce que fait le test, l'exhaustivité en plus.

Le modèle comporte 16000 primitives Simulink, 4000 sous-systèmes. Un total de 563 propriétés formelles ont été vérifiées. Une centaine d'erreurs ont été détectées dans le modèle. La taille des espaces d'états vérifiés est comprise entre 10^9 et 10^{37} états, ce qui est acceptable pour des model-checker symboliques. Le même processus de vérification a été appliqué, avec succès, sur la vérification d'un sous-système de gestion de la redondance d'un système de gestion de vol d'un drone dans [73].

Cet exemple montre qu'il est possible d'appliquer le model-checking sur des systèmes réels. Néanmoins, l'exercice n'est pas trivial et plusieurs difficultés apparaissent.

La première difficulté est la formalisation des exigences. En particulier, le niveau de détail des exigences est un facteur déterminant [74]. Plus précisément, c'est l'écart plus ou moins important des niveaux d'abstraction entre les exigences de haut niveau et les modèles. De nombreux travaux essaient d'apporter des solutions à ce problème ([75, 76]).

Une deuxième difficulté tient à la nature même des systèmes. Les systèmes fortement numériques, où les comportements manipulent des réels, sont difficiles à vérifier par model-checking. Les nombres flottants aggravent le problème de l'explosion combinatoire. Une solution ici est d'utiliser une simplification en modélisant les nombres à virgules flottante en nombre à virgules fixe [73]. Une simplification a toujours un coût : elle permet de détecter des erreurs mais pas d'en prouver l'absence.

Une troisième difficulté est l'analyse des contre-exemples. Les erreurs détectées sont subtiles par nature et les contre-exemples sont exprimés comme des séquences d'états du modèle de vérification. Généralement les langages de vérification sont plus bas niveau que les modèles systèmes, les contre-exemples, souvent longs, en sont d'autant moins lisibles. Or, les ingénieurs exigent, légitimement, un certain niveau de confort dans l'utilisation des outils. Des travaux existent sur l'analyse simplifiée des contre-exemples ([77],[78]).

Fondamentalement, la quatrième difficulté est l'explosion combinatoire. Ce problème rend difficile l'utilisation du model-checking pour prouver l'absence d'erreur dans un système : les systèmes vérifiés sont trop complexes pour être vérifiés automatiquement et uniquement par model-checking.

Ce que l'on peut tirer comme enseignement ici est que le model-checking est moins efficace pour prouver la sûreté d'un système que pour détecter des erreurs au plus tôt dans un processus de développement. Autrement dit, il peut être plus utile pour le développement d'un système que pour sa vérification au sens de la certification.

En conclusion, lorsque le système vérifié ne réalise pas de calcul numérique complexe et que les types de données manipulés sont simples (booléen, énumération,...) alors des

5. `http://nusmv.fbk.eu/`

applications efficaces du model-checking sont possibles. Dans ce cas, le model-checking permet de détecter des erreurs dans la spécification, ce qui peut justifier l'effort nécessaire à sa mise en oeuvre.

Les exemples donnés dans cette section ne traitent pas de contraintes temporelles exprimées en temps physique. Dans une architecture GALS, en l'absence de mécanismes de synchronisations, la seule référence commune de temps est le temps physique. Dans ce cas, il faut utiliser des modèles temporisés pour la vérification, les automates temporisés ou les réseaux de Petri temporels. La section suivante montre un exemple de vérification au niveau système réalisé avec le langage Fiacre.

2.2 Un exemple de vérification par model-checking

Les modèles formels évoqués dans la section précédente reposent sur une sémantique synchrone. Cette sémantique abstrait le temps et par là même simplifie la spécification, la conception, l'implémentation et la validation des systèmes temps réel. En particulier, Scade est devenu un standard de fait pour la réalisation de systèmes avioniques, au moins au sein d'Airbus et de ses principaux équipementiers.

Lustre est un outil reconnu pour la spécification fonctionnelle d'un système. Un générateur de code certifié au niveau DAL-A permet de générer environ 90% du code des composants logiciels [71]. Il est prouvé que le code généré est fonctionnellement équivalent au modèle. Comme le générateur est certifié en tant qu'outil de développement, c'est à dire avec les mêmes contraintes que le logiciel qu'il génère, la certification de la conformité fonctionnelle du code et au système est simplifiée. De fait l'outil Scade et donc la sémantique synchrone sont devenus incontournables dans l'industrie aéronautique.

Néanmoins, les composants logiciels développés via Scade sont déployés dans une architecture globalement asynchrone. De cette divergence émergent des exigences non-fonctionnelles, des contraintes temporelles que le model-checking temporisé, asynchrone par nature, peut aider à définir et vérifier.

Pour illustrer les difficultés dues à ce type d'architecture, nous considérons l'exemple d'une architecture Command/Monitoring (COM/MON). Certaines fonctions de l'avion doivent, pour des raisons de sûreté, être décomposées en deux fonctions exécutées sur deux composants distincts [79](Section 2.1.1). La fonction de commande (COM) assigne des commandes aux actuateurs tandis que la fonction de monitoring (MON) vérifie l'absence d'erreurs du COM. Pour simplifier on considère que les composants logiciels de chaque fonction sont implémentés par une partition ARINC-653 (voir Section 2.1.2). Les deux partitions s'exécutent sur deux modules IMA distincts et communiquent entre elles de manière asynchrone via le réseau AFDX.

Nous ne traitons pas les aspects de disponibilité. Pour garantir un niveau de disponibilité satisfaisant, en regard de la criticité de la fonction, les composants qui implémentent les fonctions COM et MON sont redondés.

Une conséquence de l'architecture GALS est que l'ingénieur doit prendre en compte les asynchronismes de l'architecture au niveau du modèle Scade. Une solution pratique consiste à prolonger un évènement suffisamment longtemps pour que les deux composants aient une vision consistante de cet évènement. L'ingénieur doit pour ce faire exprimer des durées et justifier, dans le contexte de la certification, que ces durées sont correctes. Ces durées sont exprimées en temps logique, c'est à dire en terme de cycles d'exécution (une partition peut être abstraite comme une tâche périodique). Si une erreur est introduite à ce niveau, par exemple une durée trop courte, son coût de correction sera d'autant plus élevé qu'elle sera détectée tard dans le cycle de développement.

Dans cette section, nous décrivons une expérimentation qui montre comment le model-checking temps réel peut être utile pour vérifier ce type de propriété temporelle. Cette étude de cas est cohérente vis à vis des exemples d'utilisation présentés dans la Section 2.1.3 : la fonction de monitoring n'effectue pas de calcul numérique, le périmètre de la vérification est limité et la propriété est principalement influencée par le modèle d'exécution temporel du système. Ce modèle d'exécution inclut les phases, périodes et durées d'exécution des partitions ainsi que les délais de communication. La durée d'exécution d'une tâche est précisément estimée par les techniques d'analyse du WCET (Section 2.1.3). Les délais de communication sont estimés par analyse du pire cas de traversée (WCTT) dans un réseau AFDX. Il est donc possible d'avoir une description assez précise du comportement temporel au niveau de la spécification et donc appliquer le model-checking temps-réel [80]. La pratique courante aujourd'hui est de prouver la correction de ces durées, au sens de la DO, par analyse humaine. Il y a donc un intérêt à automatiser cette tâche par la méthode de vérification exhaustive que propose le model-checking temps-réel.

Cette section a fait l'objet d'une publication lors de la 25-ième édition du symposium IEEE "Software Reliability Engineering Workshops (ISSREW)" sous le titre :

> *Model-Checking Real-Time Properties of an Auto Flight Control System Function* [1]

2.2.1 Système de gestion automatique de vol

L'objet de la vérification présentée dans cette section est une fonctionnalité d'un système de gestion automatique de vol.

Un système de gestion automatique de vol, ou AutoFlight Control System (AFCS), fournit un support opérationnel aux pilotes. Il vise à réduire la charge de travail des pilotes, stabiliser au mieux l'avion, améliorer le confort de vol et finalement augmenter les capacités opérationnelles (eg : atterrissage avec visibilité réduite). Ces services sont proposés à travers quatre sous-systèmes :

- Le *pilote automatique(AP)* commande les actuateurs, directement ou indirectement via le système de contrôle de vol ;
- La *commande de gaz automatique* commande la poussée moteur ;
- Le *directeur de vol* indique au pilote, sur l'horizon artificiel, la position que doit prendre l'avion par rapport à une trajectoire cible ;
- Le *guidage de vol* permet au pilote d'opérer l'avion à travers des scénarios de haut niveau, appelés *modes*. Ces modes contraignent les interactions possibles entre le pilote et l'avion. Des exemples de modes sont "maintenir une altitude cible", "atteindre une vitesse cible", ...

Nous nous intéressons au sous-système de guidage de vol et plus particulièrement à une fonction de capture d'altitude cible dans le mode "atteindre une altitude cible". Cette fonction permet au pilote, à partir d'un bouton rotatif placé dans le cockpit, de sélectionner la consigne d'altitude transmise au pilote automatique. Un exemple d'équipement installé dans le cockpit est montré dans la Figure 2.1. Le bouton est encadré en rouge.

L'architecture fonctionnelle répond à des exigences de sûreté via une architecture fonctionnelle COM/MON. Comme indiqué en Section 2.2, ces composants sont spécifiés fonctionnellement par une sémantique synchrone et déployés dans un environnement asynchrone. En conséquence des exigences non fonctionnelles émergent [81]. Les exigences auxquelles nous nous intéressons sont des contraintes temporelles que les composants doivent respecter. Ces composants héritent du niveau de criticité DAL-B. Le pilote peut toujours désactiver le système de guidage de vol lorsque ce dernier tombe en panne, à condition

FIGURE 2.1: Bouton rotatif de capture d'altitude

que la panne soit détectée. D'où la nécessité d'utiliser l'architecture COM/MON et de s'assurer que le MON détecte bien les erreurs du COM.

Architecture

Nous décrivons maintenant l'architecture simplifiée pour la capture d'altitude, illustrée dans la Figure 2.2. Une vue plus fonctionnelle de l'architecture est donnée dans la Figure 2.3. Cette dernière figure représente les variables et autres paramètres qui sont décrits ci-après.

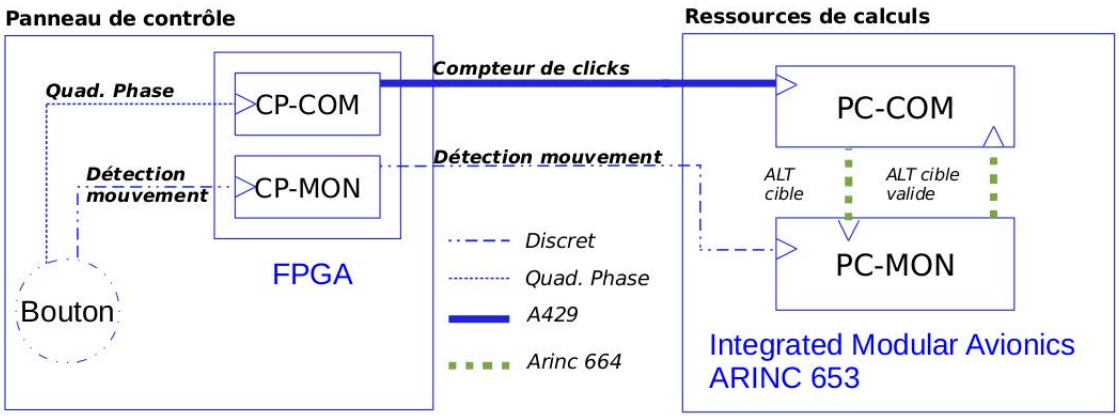


FIGURE 2.2: Architecture de la fonction de capture d'altitude

Le panneau de contrôle

Le panneau de contrôle est une carte matérielle composée

— d'un bouton rotatif avec retour tactile($knob$) ;

— d'un composant matériel de commande(CP_COM) ;

— d'un composant matériel de surveillance (CP_MON).

Les composants du panneau de contrôle sont synchronisés sur la même horloge. Le bouton possède 256 positions, appelées "clicks". Le pilote sélectionne une altitude cible en tournant le bouton, à droite ou à gauche. Lorsqu'il est en mouvement le bouton émet deux signaux :

1. Une quadrature de phase qui encode un incrément $+1$ (resp. -1) pour chaque click à droite(resp. à gauche). La durée minimale entre deux click est notée δ.
2. Un signal de détection de mouvement, appelé mvt, haut tant que le bouton bouge, bas sinon

CP_COM incrémente (resp. décrémente) un compteur local, modulo 256, à chaque mouvement vers la droite (resp. vers la gauche) du bouton et transmet à PC_COM la valeur

du compteur avec une période $p3$. La valeur est transmise via un bus A429 avec une latence notée $a429\_ltcy$. Les propriétés physiques du bouton et la fréquence d'échantillonage de CP_COM garantissent que le bouton ne peut pas faire plus de la moitié d'un tour entre deux échantillonages. La norme A429 [82] spécifie des bus de terrains mono-émetteur/multi-receveur très utilisés dans l'avionique. Le temps de traitement nécessaire à CP_COM pour convertir la quadrature de phase et émettre la valeur du compteur est de l'ordre de la micro-seconde et peut-être négligé.

CP_MON traite le signal de détection de mouvement. Dès que celui-ci est haut, CP_MON émet un signal haut, noté $cMvt$ sur un "discret" à destination de PC_MON. Un discret est un "fil" capable de transmettre un signal haut/bas d'un point à un autre avec une probabilité d'erreur très faible ($< 10^{-9}/hv$). CP_MON maintient ce signal pendant $cProl$ ms après que le signal de détection de mouvement soit repassé à bas.

Les partitions PC_COM et PC_MON

Le composant logiciel de commande PC_COM et le composant logiciel de surveillance PC_MON sont implémentés par des partitions A653. Nous choisissons d'abstraire ces partitions, générées à partir de leur spécification Lustre, par des tâches périodiques. Chacune de ces deux tâches a un période, une durée d'exécution et une phase. La période de PC_COM est noté $p1$, sa durée $d1$. La période de PC_MON est notée $p2$, sa durée $d2$. Les partitions lisent leurs données d'entrée au début de l'exécution et écrivent leurs sorties à la fin. Sans perte de généralité, nous considérons que les lectures et écritures sont atomiques et en temps nul. Les deux partitions communiquent de manière asynchrone via un réseau AFDX et des ports "sampling" (Section 2.1.2).

A chaque exécution PC_COM lit le nombre de click émis par CP_COM et calcule une altitude cible, notée alt_target. Cette altitude est envoyée à PC_MON via l'AFDX avec une latence $afdx\_ltcy$.

PC_MON détecte les changements d'altitude en comparant les valeurs reçues successives. Une variable booléenne, alt_change, est positionnée si deux valeurs successives diffèrent. De plus, PC_MON prolonge localement le signal de détection de mouvement émis par CP_MON. Le signal local est noté $pProl$ et il est maintenu pendant $pProl$ ms après la dernière exécution où le signal émis par CP_MON était haut.

Le calcul et la validation des durées de prolongation $cProl$ et $pProl$ représentent une difficulté pour l'ingénieur. En Lustre, la notion de durée n'existe pas, il faut que l'ingénieur compte les cycles en incrémentant/décrémentant un compteur de cycles. La durée calculée doit capturer tous les asynchronismes possibles entre les deux partitions et intégrer les différents délais de communication. Nous pensons que la vérification par model-checking temporisé facilite le calcul et la validation de ces durées de prolongation.

Objectifs de vérification

La première étape dans la lutte contre l'explosion combinatoire est la construction du modèle. Comme le choix des abstractions est guidé par la propriété à vérifier, il est essentiel de définir au plus tôt l'objectif de la vérification. Nous considérons l'erreur suivante :

> A partir d'un état sans erreur du système, PC_COM émet deux altitudes cibles successives différentes non précédées d'un mouvement de bouton.

Pour ce qui nous concerne, le rôle de PC_MON est de détecter cette erreur. Pour bien délimiter le problème, nous faisons l'hypothèse que PC_MON, le panneau de contrôle et les réseaux sont fiables. A cause des asynchronismes, PC_MON peut ne pas détecter un

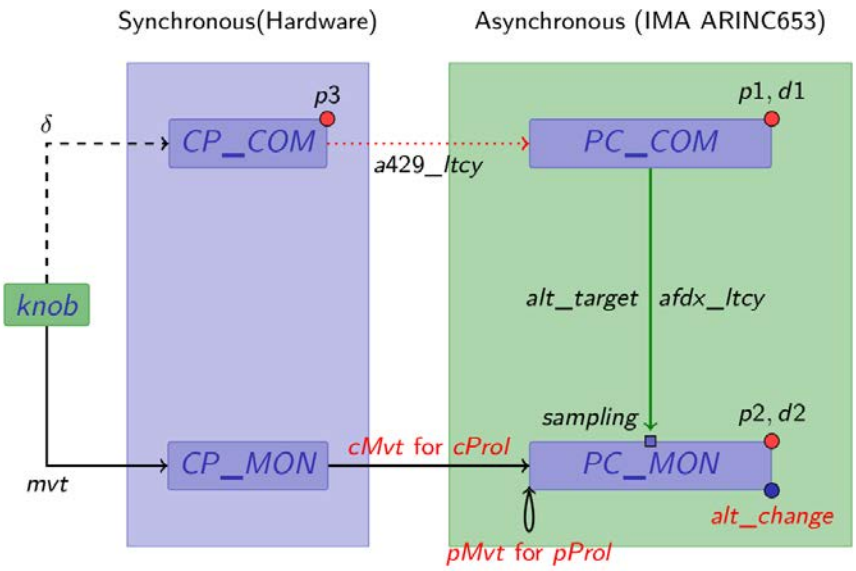


FIGURE 2.3: Vue fonctionnelle

mouvement du bouton. Si les signaux $pMvt$ et $cMvt$ ne sont pas prolongés suffisamment longtemps, un mouvement entre deux exécutions de PC_MON est ignoré par ce dernier. Les durées de prolongation doivent permettre à PC_MON de toujours détecter les mouvements du bouton. Notre objectif est de vérifier que les durées de prolongations fournies par l'ingénieur permettent à PC_MON de détecter tous les mouvements du bouton.

Plus formellement, nous vérifions qu'un modèle $\mathcal{M}$ du système, instancié par les durées $\Gamma = (cProl, pProl)$, vérifie la propriété de sûreté $DETECT$. Les autres paramètres du modèle ($afdx\_ltcy$, $p1$, $d1$, ...) sont fixés. Une configuration Γ telle que $DETECT$ est vérifiée garantit que PC_MON détecte tous les mouvements du bouton.

$$\Box(alt\_change \Rightarrow pMvt) \qquad (DETECT)$$

La propriété $DETECT$ est une propriété de sûreté. Elle spécifie l'absence d'états où un changement d'altitude a eu lieu mais aucun mouvement de bouton n'a été détecté. Dans ce cas d'étude, ce type de propriété permet une vérification plus efficace, en autorisant l'utilisation de l'abstraction de l'inclusion des classe d'états (Section 1.2.3).

2.2.2 Modélisation de la capture d'altitude

Nous détaillons maintenant le modèle de vérification FIACRE construit pour vérifier la propriété $DETECT$. Des extraits de ce modèle sont présentés. Cette étude de cas est tirée d'un système développé par Thales Avionics SA. Afin de préserver la confidentialité, les bornes temporelles sont représentées par des chaines de caractères (eg: [`afdx_ltcy` − `jitter`, `afdx_ltcy` + `jitter`]). En réalité, le langage FIACRE n'autorise que les bornes entières. Néanmoins, les modèles FIACRE étant décrits sous forme textuelle, on peut toujours les paramétrer statiquement via l'utilisation de pré-traitement.

Abstraction des données

La propriété $DETECT$ ne dépend pas d'une valeur particulière d'altitude, on peut abstraire les altitudes en modélisant uniquement les changements d'altitudes. Cette abstraction est indispensable car la valeur du compteur est émise tous les $p3$ ms, avec $p3$ petit, et que cette valeur est comprise entre 0 et 255. Comme aucune hypothèse n'est faite

sur les entrées (ie: les mouvements du pilote), les valeurs possibles du compteur sont un facteur important d'explosion.

Les incréments d'altitude et leurs directions sont abstraits par une énumération à deux valeurs $\{Change, NoChange\}$. Lorsque le bouton est en mouvement, le compteur contient la valeur $Change$ et lorsqu'il est à l'arrêt le compteur vaut $NoChange$.

Certains comportements peuvent être manqués à cause de cette abstraction. La valeur du compteur est échantillonnée par CP_COM avec une période $p3$. Dans le scénario où le compteur est incrémenté entre deux échantillonnages et comme δ est plus petit que $p3$, la valeur abstraite du compteur restera la même. La Figure 2.4 illustre ce scénario. Une variable intermédiaire, qui mémorise les mouvements entre deux échantillonnages, est ajoutée au modèle pour corriger le problème.

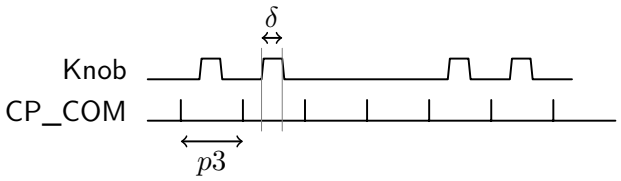


FIGURE 2.4: Mouvements ignorés

Abstraction temporelle

Dès lors que les clicks sont abstraits on peut abstraire le comportement temporel du signal de quadrature de phase. L'abstraction sur les données se décline en une abstraction temporelle.

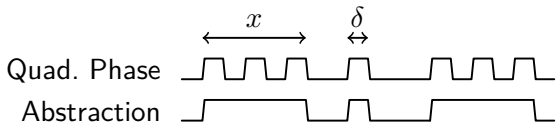


FIGURE 2.5: Knob movement abstraction

Le premier chronogramme de la Figure 2.5 montre un comportement possible du signal de quadrature émis par le bouton, en négligeant le sens de rotation. Dans cette figure, x représente une séquence de 3 clicks et δ est la durée (fixe) d'un click. Le bouton est toujours actionnable par le pilote et aucune hypothèse n'est faite sur ses actions. De plus δ est deux ordres de grandeur plus petit que la période de PC_COM. Comme PC_COM est exécutée infiniment souvent, ces différences d'échelles entre les contraintes temporelles aggravent le problème d'explosion combinatoire. Intuitivement, les mouvements du bouton introduisent beaucoup d'états intermédiaires entre deux exécutions de PC_COM.

L'abstraction temporelle est illustrée dans la Figure 2.5. Au lieu de modéliser les clicks du bouton (chronogramme du haut) nous modélisons les mouvements du bouton (chronogramme du bas), sans aucune contrainte temporelle entre deux mouvements. Cette abstraction simplifie les contraintes temporelles impliquées lors du calcul des classes d'états tout en réduisant le nombre de classes. Néanmoins, elle introduit des comportements Zeno : une infinité de mouvements peut se produire en un temps fini. Comme $DETECT$ est une propriété de sûreté, si elle est vraie dans le modèle abstrait alors elle est vraie dans le modèle concret.

Abstraction des réseaux

Le réseau A429 est un bus simple mono émetteur qui garantit que les récepteurs verront le dernier message émis après un temps borné. En pratique, le coupleur réseau stocke ce

```
1  process AFDX_ALT_TARGET[input:AltTarget](&output:AltTarget) is
2  states idle,send
3  var alt_target:AltTarget:=ChangeAltNo
4  from idle
5    input?alt_target; to send
6  from send
7    wait [afdx_ltcy-jitter,afdx_ltcy+jitter];
8    output:=alt_target;
9    to idle
```

FIGURE 2.6: Modèle du réseau AFDX

message dans une mémoire partagée accessible par l'application. A intervalle périodique, l'application lit la valeur dans la mémoire partagée. Le modèle FIACRE de l'A429 implémente cette description.

Le réseau AFDX est plus complexe, composé de switch ethernet dans lesquels les messages peuvent être mis en file d'attente (Section 2.1.2). L'utilisation de files d'attente dans un modèle aggrave le problème de l'explosion combinatoire. Par ailleurs, des travaux existent sur l'estimation de bornes sur la latence du réseau qui prennent en compte l'impact des files [83]. Le pire cas de traversée peut être estimé avec une précision suffisante pour chaque lien virtuel proposé par le réseau. De plus, la configuration du réseau (nombre et types des messages, tables de routages) est statique. A partir de ces données, le modèle FIACRE abstrait l'AFDX comme un ensemble de ces liens virtuels, avec un processus par lien, synchronisé avec l'émetteur unique du lien virtuel. Une particularité du cas d'étude est que les fonctions communiquent avec le réseau via des ports échantillonnés. Cette propriété permet de modéliser la sortie du lien virtuel par une variable partagée. La Figure 2.6 montre le processus FIACRE qui modélise le lien virtuel acheminant l'altitude cible depuis *PC_COM* vers *PC_MON*.

Les réseaux sont par hypothèses toujours disponibles : cette hypothèse est réaliste en avionique où les réseaux utilisés sont conçus pour être extrêmement fiables. Néanmoins, lorsque `AFDX_ALT_TARGET` est dans l'état `send` (ligne 6) il ne peut pas se synchroniser sur le port `input`. Cela ne pose pas de problème car le processus `AFDX_ALT_TARGET` se synchronise uniquement avec *PC_COM*. Or la période de ce dernier est approximativement quinze fois supérieure au pire cas de traversée. En conséquence, si le réseau est disponible au premier cycle de *PC_COM*, il le sera pour tous les autres.

Finalement, la gigue possible du réseau AFDX est prise en compte dans l'intervalle temporel de la ligne 7. Cette gigue, ainsi que la latence, sont des paramètres du modèle.

Asynchronismes

En l'absence d'erreur, il y a deux causes possibles d'asynchronisme :

1. Les dérives d'horloges
2. Les déphasages à l'initialisation

Les modèles temporisés couramment utilisés (Chapitre 1), notamment les TPN, font l'hypothèse d'horloges parfaites. En réalité les horloges dévient par rapport au temps newtonien. Nous ne prenons pas en compte ces dérives d'horloges dans le modèle. A la place, l'hypothèse est faite que le système ne s'exécute pas suffisament longtemps pour que les dérives d'horloges puissent changer son comportement. Cette simplification se justifie car le système modélisé est un système avionique et sa durée d'exécution est bornée par la durée du vol de l'avion dans lequel il est embarqué. C'est une hypothèse faite en pratique par les ingénieurs qui conçoivent ces systèmes. Le risque induit par les dérives est généralement mitigé par du monitoring d'horloges en vol.

Dans une architecture GALS, les différents composants ne se synchronisent pas entre eux. Nous ne faisons donc aucune hypothèse sur d'éventuelles synchronisations. Le modèle doit donc capturer les ordres possibles d'initialisation et parmi ces ordres les contraintes temporelles possibles. Cela aggrave le problème de l'explosion combinatoire.

Nous adoptons une stratégie diviser pour mieux régner. Au lieu d'un seul modèle on construit plusieurs modèles, un pour chaque ordre d'initialisation possible (eg: composants fiacres A puis B et composant B puis A). Chacun de ces modèles capture les contraintes temporelles possibles étant donné un ordre d'initialisation. Il serait alors nécessaire de prouver que ces différents modèles couvrent effectivement tous les cas. Nous ne proposons pas cette preuve.

Modélisation du panneau de contrôle

La Figure 2.7 montre les processus *KNOB* et *CP_MON* du panneau de contrôle.

```
1  type Clicks is union Change | NoChange end
2
3  process KNOB [up,down:sync] (&ck:Clicks) is
4  states idle,move
5  from idle
6    up; ck:=Change; to move
7  from move
8    down; ck:=NoChange; to idle
9
10 process CP_MON [up,down:sync] (&cMvt:bool) is
11 states idle,sustain,waitdown
12 from idle
13   up; cMvt:=true; to waitdown
14 from waitdown
15   down; to sustain
16 from sustain
17   select
18      wait [cProl,cProl]; cMvt:=false; to idle
19   [] up; cMvt:=true; to waitdown
20   end
21
22 component CONTROL_PANEL (&cMvt: bool) is
23 var ck:Click := NoChange
24 port up,down:sync
25 par * in
26    KNOB [up,down] (&ck)
27 || CP_MON [up,down] (&cMvt)
28 || CP_COM (&ck)
29 end
```

FIGURE 2.7: Modèle Fiacre du panneau de contrôle

Le bouton (`KNOB`) a deux états qui indiquent le mouvement ou l'absence de mouvement. Il se synchronise avec `CP_MON` par l'intermédiaire des ports `up` et `down`. La partie intéressante ici est la prolongation du signal `cMvt` modélisé par `CP_MON`.

A l'état initial et dès que le bouton commence à bouger, le processus `CP_MON` positionne la variable partagée `cMvt`, qui modélise le discret $cMvt$. Lorsque le bouton repasse dans l'état `idle`, `CP_MON` passe dans l'état `sustain`. Il enclenche alors un timer, modélisé par la transition en ligne 18. On remarque que cette transition est toujours sensibilisée (la garde est vraie par défaut). Elle le restera jusqu'à ce qu'elle soit tirée, sauf si le bouton reprend ses mouvements avant `cProl` unité de temps. En conséquence, la variable `cMvt` reste vraie pendant $cProl$ après que le bouton ait cessé de bouger, modélisant ainsi la prolongation du signal. La Figure 2.8 donne le modèle TTS du *CP_MON*, généré par le compilateur FRAC (Section 1.4).

Le processus *CP_COM* est modélisé comme une tâche périodique. La modélisation

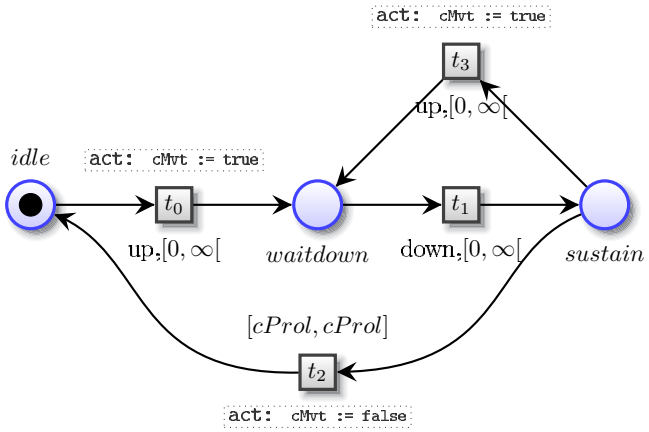


FIGURE 2.8: Modèle TTS du *CP_MON*

des tâches périodiques est présentée dans la section suivante. A intervalles réguliers, il lit la valeur du compteur puis se synchronise sur un port urgent avec l'A429. Ce dernier peut toujours se synchroniser avec *CP_COM*. Le processus *CP_COM* gère une variable intermédiaire pour ne pas ignorer les rafales de mouvements brefs qui peuvent se produire entre deux échantillonnages.

Modélisation des tâches périodiques

Les partitions *PC_COM* et *PC_MON* sont modélisées comme des tâches périodiques. Comme ces tâches ont des durées d'exécution, nous découplons la période et la durée. Une partition est modélisée par un composant FIACRE composé d'un processus d'horloge et d'un processus de traitement.

Le processus d'horloge active le processus de traitement à chaque début de période (eg: tous les $p1$ pour `PC_COM`). Le processus de traitement modélise la durée d'exécution par une attente FIACRE.

La Figure 2.9 montre le modèle FIACRE de la partition *PC_COM*.

L'état `rst` du processus `PC_COM_CLOCK` (ligne 17) sert à modéliser les ordres d'initialisations. Les paramètres `initmin` et `initmax` varient en fonction de l'ordre d'initialisation courant. A partir de l'état `rst`, l'horloge passe dans l'état `top` qui modélise le top d'horloge. Au premier top, le processus `PC_COM_CTRL` peut se synchroniser sur le port `clock`(ligne 5) et démarrer son exécution. Il lit la valeur du compteur émise par *CP_COM* (ligne 6), puis après un délai $d3$, émet l'altitude calculée à *PC_MON* via l'AFDX. La valeur émise par le processus *PC_COM* est la valeur abstraite de compteur. Dans le système concret, la partition maintient une altitude cible à partir des incréments du compteur et transmet l'altitude. Notre abstraction est suffisante par rapport à notre objectif de vérification.

Résultats expérimentaux et discussion:

Dans cette expérimentation, nous avons construit des modèles qui sont paramétrés par des durées de prolongation (*cProl*,*pProl*) mais aussi un ordre d'initialisation, les durées d'exécution et périodes des tâches, les latences de communication, ...

Les paramètres concrets du système sont extraits de sa spécification. *pProl* et *cProl* prennent leurs valeurs entre 1 et 6 cycles de *PC_MON* et 4 ordres d'initialisation sont considérés. Au final, c'est une collection de 144 modèles qui est générée. Pour chacun de ces modèles nous avons calculé l'ensemble de ses états discrets. Ces espaces d'états contiennent en moyenne 2 millions d'états et occupent approximativement 2Mb d'espace disque. La vérification de l'ensemble des modèles prend environ 1h sur un poste de travail

```
process PC_COM_CTRL[clock:none,afdx:AltTarget](&cpt:read AltInc) is
states idle,work,output
var alt_target:AltTarget := ChangeAltNo
from idle
  clock;
  alt_target := cpt;
  to work
from work
  wait [d3,d3];
  to output
from output
  afdx!alt_target;
  to idle

process PC_COM_CLOCK[clock:none]() is
states rst,idle, top
from rst
  wait [init0,init1];
  to top
from idle
  wait [p3,p3];
  to top
from top
  #CLOCK;
  clock;
  to idle

component PC_COM[afdx:AltTarget](&cpt:AltInc) is
port clock:none in [0,0]
par * in
      PC_COM_CLOCK[clock]()
  || PC_COM_CTRL[clock,afdx](&cpt)
end
```

FIGURE 2.9: Modèle de *PC_COM*

standard. La Figure 2.10 montre le résultat des expérimentations. Chaque point représente une configuration Γ pour laquelle la propriété $DETECT$ est satisfaite, quelque soit l'ordre d'initialisation. Ces valeurs sont exprimées en nombre de cycles de PC_MON et incluent celles trouvées analytiquement par les ingénieurs.

Les configurations limites pour lesquelles $DETECT$ n'est pas satisfaite ont été analysées. Dans ce cas, comme les traces sont nécessaires pour pouvoir construire les contre-exemples, on ne peut plus utiliser l'abstraction d'inclusion des classes. Tina permet de générer des contre-exemples temporisés les plus courts possibles. Ces contre-exemples contiennent environ 200 états et ont été analysés manuellement. Certain d'entre eux correspondent à des erreurs détectées par analyse par les ingénieurs.

Les abstractions réalisées sont efficaces et permettent d'obtenir des espaces d'états exploitables. Une conséquence directe est qu'en plus de pouvoir vérifier des configurations Γ calculées analytiquement par les ingénieurs, cette modélisation permet de synthétiser ces Γ.

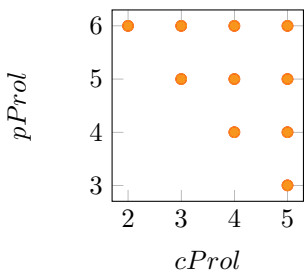


FIGURE 2.10: Configuration pour lesquelles $DETECT$ est satisfaite

Beaucoup de travail reste à faire pour automatiser cette approche et l'intégrer dans un processus industriel. En particulier, il est nécessaire d'automatiser la construction du modèle. La sémantique formelle de Lustre doit permettre d'aider à la construction automatique des abstractions, par exemple en utilisant les techniques présentées dans la Section 2.1.3 sur les programmes Lustre.

Une autre aspect important est le retour de vérification. Les contre-exemples doivent s'exprimer sur les modèles Lustre afin d'en simplifier l'analyse.

Des progrès doivent être faits sur la prise en compte des dérives d'horloges. De plus, il reste nécessaire de prouver que l'ensemble des initialisations est bien couvert par notre stratégie de diviser pour mieux régner.

Cette expérimentation illustre l'intérêt du model-checking temps réel pour la vérification de propriétés temps réel dans une architecture COM/MON d'un système GALS en montrant qu'il est possible d'automatiser certaines vérifications réalisées par analyse.

2.3 Conclusion

Dans ce chapitre, nous avons présenté le contexte des systèmes avioniques, les méthodes de vérification dans ce contexte et un exemple de vérification par model-checking de contraintes temporelles.

Les systèmes avioniques sont des systèmes temps réel, distribués et critiques. Le développement d'un tel système est structuré par des contraintes très fortes de certification. Ces contraintes structurent aussi l'activité de vérification et en particulier l'application des méthodes de vérification formelles.

Certaines méthodes formelles sont utilisées pour la vérification de systèmes aujourd'hui en production(ie: embarqués dans des avions). La preuve de programme et l'interprétation abstraite ont trouvé des applications réussies, après plusieurs années d'investissement par les industriels et de recherche par les académiques.

Néanmoins, à notre connaissance, ces usages opérationnels restent limités à la vérification de programme séquentiels. La preuve de programme est utilisée pour remplacer le test unitaire, tandis que l'interprétation est utilisée pour éliminer les erreurs d'exécution et calculer des temps d'exécution. Ces applications permettent déjà des simplifications considérables des processus de vérification. De plus les normes pour la certification, qui évoluent en capitalisant sur les pratiques de l'industrie, commencent à accepter les méthodes formelles comme moyen de preuve.

La technique de vérification par model-checking s'est moins diffusée dans l'industrie que la preuve de programme ou l'interprétation abstraite. Un des problèmes principaux est l'explosion combinatoire. Nous pouvons nous risquer à dire que le model-checking a une mauvaise image auprès des utilisateurs industriels moins par ses limitations théoriques que par le fait qu'il ai été survendu à ses débuts, il y a plus de 30 ans.

Nous pensons que le model-checking et le model-checking temps réel ont un rôle important à jouer dans la vérification des systèmes avioniques. Aujourd'hui, la preuve de programme et l'interprétation abstraite ne peuvent pas s'appliquer simplement aux systèmes distribués. Le model-checking est par construction très bien adapté à la vérification des systèmes où la dimension concurrente est importante.

En particulier, les asynchronismes des architecture GALS peuvent activer des fautes de concurrence. De plus, la dimension temporelle est très importante dans le contexte avionique. C'est pour cette raison que nous avons proposé une expérimentation de vérification de contraintes temporelles dans ce contexte. Cette expérimentation contribue très modestement à montrer que le model-checking peut s'appliquer et aider à détecter au plus tôt des erreurs spécifiques aux systèmes distribués temps réel.

Deuxième partie

Symétries pour les réseaux de Petri temporels

Chapitre 1

Rappels sur les groupes de permutations

Sommaire

Ce chapitre donne quelques éléments de la théorie des groupes de permutations utiles à la compréhension de cette thèse. Toutes les définitions et théorèmes qui sont présentés sont des résultats standard des groupes de permutations. On trouvera dans [84] une présentation plus complète des groupes de permutations.

Dans cette thèse tous les groupes sont finis.

1.1 Groupes

Un groupe G est un ensemble muni d'une opération binaire $\bullet$ vérifiant les lois ci-dessous :

Fermeture : Pour tout $g, h \in G$, $g \bullet h \in G$

Associativité : Pour tout $g, h, k \in G$ nous avons $g \bullet (h \bullet k) = (g \bullet h) \bullet k$

Identité : Il existe un élément neutre $e \in G$ tel que pour tout $g \in G$ nous avons $g \bullet e = e \bullet g = g$.

Inverse : Pour tout $g \in G$, il existe $g^{-1} \in G$ tel que $g \bullet g^{-1} = g^{-1} \bullet g = e$.

Lorsque le contexte le permet, nous omettrons le symbole $\bullet$, et écrirons gh pour $g \bullet h$. L'inverse d'un produiit $ab \in G$ est défini comme $(ab)^{-1} = b^{-1}a^{-1}$.

Soit $X \subseteq G$ un sous ensemble de G. On note $G = \langle X \rangle$ lorsque tous les éléments de G peuvent s'écrire comme des produits d'éléments de X. Les éléments de X sont appelés des générateurs.

Le nombre d'éléments de G est appelé *ordre* de G, noté $|G|$. Le groupe trivial est le groupe d'ordre 1.

1.1.1 Sous-groupes et cosets

Un sous-ensemble non vide H de G est un sous groupe de G, noté $H \leq G$, si il contient l'identité, si $a \in H \Leftrightarrow a^{-1} \in H$ et si pour tout $a, b \in H$ on a $ab \in H$. Un sous groupe est *strict* si il est strictement inclus dans G. La taille d'un sous-groupe divise toujours celle du groupe (Théorème de Lagrange). Le diviseur est appelé *index* de H, et noté $[G : H]$:

$$|G| = |H| \times [G : H]$$

Soit $\sim \subseteq G \times G$ la relation définie, pour $a, b \in G$, par :

$$a \sim b \text{ si et seulement si } a^{-1}b \in H$$

La relation $\sim$ définie est une relation d'équivalence dont les classes sont appelées *cosets* à gauche. Tous les cosets de H sont de la forme

$$aH = \{ah \mid h \in H\}$$

Une définition analogue pour les coset à droite existe. Dans cette thèse, nous n'utiliserons que les cosets à gauche.

Il y a $[G : H]$ cosets de H et ils ont tous la même taille. Un élément a appartient à un coset C de H si et seulement si $C = aH$. Soit deux cosets aH et bH, alors les cosets aH et bH sont égaux si et seulement si on a $a^{-1}b \in H$. Sinon, ils sont disjoints.

Une *transversale à gauche* de H, notée F, est un sous-ensemble de G qui contient exactement un représentant pour chaque coset de H. Pour tout $a \in G$, on a donc $|aH \cap F| = 1$.

Soit $a, b \in G$, le *conjugué* de a par b est l'élément $aba^{-1} \in G$. Le conjugué de H par a est le sous-groupe $aHa^{-1} = \{aha^{-1} \mid h \in H\}$. Un sous-groupe normal N de G, noté $N \trianglelefteq G$, est un sous-groupe de G tel que pour tout $g \in G$ et $n \in N$ on a $gng^{-1} \in N$.

Le quotient G/N d'un groupe G par un sous-groupe normal N est le groupe formé par l'ensemble des cosets de N équipé de la loi de composition $(aN)(bN) = abN$.

L'exemple 1.1.1 illustre les définitions de groupes et de cosets.

Exemple 1.1.1. *Soit G le groupe des entiers modulo 8 équipé de la loi de composition $+$. L'élément neutre est 0, l'inverse de g est noté $-g$ (eg: -2) et le groupe peut être généré par 1 : $G = \langle 1 \rangle$.*

Soit le sous-ensemble $H = \{0, 4\}$. H est un sous-groupe puisque $0+4 = 4 \in H$ et $4+4 \mod 8 = 0 \Leftrightarrow 4 = -4$ et donc l'inverse de 4 est dans H. Il est normal car $1 + 0 - 1 = 0$ et $1 + 4 - 1 = 4$.

Il y a 4 cosets de H :

$$\begin{aligned} 0 + H &= \{0, 4\} \quad H \text{ est un coset de lui même} \\ 1 + H &= \{1, 5\} \\ 2 + H &= \{2, 6\} \\ 3 + H &= \{3, 7\} \end{aligned}$$

L'index de H dans G est $8/2 = 4$. Il y a bien 4 cosets et ceux ci forment une partition de G. Le quotient de G par H est le groupe des entiers modulo 4.

1.1.2 Homomorphisme de groupes

Soit deux groupes G et G'. Un *homomorphisme* de G vers G' est une fonction $\theta : G \to G'$ telle que pour tout $a, b \in G$

$$\theta(a \bullet_G b) = \theta(a) \bullet_{G'} \theta(b)$$

Un homomorphisme bijectif est appelé un *isomorphisme*. Un isomorphisme de G vers G est appelé un *automorphisme*. Le groupe des automorphismes de G est noté $Aut(G)$. Nous noterons $G \simeq G'$ l'isomorphisme entre G et G' et $G = G'$ l'égalité entre G et G'. La composition de deux homomorphismes est un homomorphisme. Si G est généré par X alors l'image par θ de G est générée par l'image de X par θ : $\theta(\langle X \rangle) = \langle \theta(X) \rangle$.

L'*image* de θ, notée $Img(\theta)$, est l'ensemble $\{g' \in G' \mid \exists g \in G \text{ et } g' = \theta(g)\}$, et c'est un sous-groupe de G'. Le *noyau* de θ, noté $Ker(\theta)$, est l'ensemble $\{g \in G \mid \theta(g) = e\}$. Le noyau est un sous-groupe normal de G.

Le *premier théorème d'isomorphisme* montre que le quotient de G par le noyau de θ est isomorphe à l'image de θ. On en déduit que si $Ker(\theta)$ est trivial, alors G est isomorphe à l'image de θ.

Théorème 1.1. *Soit $\theta : G \to G'$ un homomorphisme de groupe. Alors :*

$$G/Ker(\theta) \simeq Img(\theta)$$

1.2 Groupes de permutations

Une *permutation* d'un ensemble Ω est une bijection $\pi : \Omega \to \Omega$. L'ensemble de toutes les permutations de Ω, équipé de la loi de composition, forme un groupe appelé *groupe symétrique* sur Ω et noté $Sym(\Omega)$ ou S_n si la nature précise de l'ensemble Ω n'a pas d'importance. Le *degré* d'un groupe de permutations est la taille de Ω.

Un *groupe de permutations* est un sous-groupe de S_n.

Les permutations sont données en notation cyclique (eg:$(1, 2, 3)(6, 9)$) et l'élément neutre est noté $()$. Un cycle de longueur 2 est appelé une *transposition*. Nous noterons C_n le groupe formé par les rotations de n éléments (eg: $(1, 2, \ldots, n), (2, 3, \ldots, n, 1), \ldots$).

Exemple 1.2.1. *Le groupe $G = \langle (1, 2, 3) \rangle \leq S_3$ est un groupe de permutations Soit un réseau N avec $P = \{p1, p2, p3\}$ et $T = \{t1, t2, t3\}$. Le groupe $H = \langle (p1, p2, p3)(t1, t2, t3) \rangle$ est un sous-groupe de $Sym(P \cup T)$ et donc un groupe de permutations de $P \cup T$. On peut définir un homomorphisme $\theta : G \to Sym(P \cup T)$ par*

$$\theta((1, 2, 3)) = (p1, p2, p3)(t1, t2, t3)$$

Le noyau de θ est trivial et son image est le sous-groupe H.

1.2.1 Action de groupe

Une *action* γ d'un groupe G sur un ensemble Ω est un homomorphisme de G vers $Sym(\Omega)$ (Exemple 1.2.1). L'image de γ est notée G^γ. Pour tout $g \in G$ et $\omega \in \Omega$, nous noterons l'image de ω par $\gamma(g)$ comme $g \cdot \omega$ ou $g\omega$ lorsque le contexte le permet.

L'action γ est *fidèle*(ou injective) si pour tout $g_1 \neq g_2 \in G$, il existe un $\omega \in \Omega$ tel que $g_1\omega \neq g_2\omega$. De manière équivalente, γ est fidèle si pour tout g différent de e il existe un ω tel que $g\omega \neq \omega$. Si γ est fidèle alors son noyau est trivial et l'image de γ est isomorphe à G par le Théorème 1.1.

Exemple 1.2.2. *Tous les sous-groupes de S_n agissent fidèlement sur $1..n$. Soit $G = \langle (1,2),(4,5)\rangle \leq S_5$.*

L'action γ de G sur $1..5$ définit par $\gamma(g) = g$ est fidèle.

Par contre, l'action γ de G sur $1..2$ n'est pas fidèle puisque $(4,5)$ fixe tous les points de $1..2$.

Deux actions $\gamma_1 : G \to Sym(\Omega_1)$ et $\gamma_2 : G \to Sym(\Omega_2)$ sont *équivalentes* si il existe une bijection $f : \Omega_1 \to \Omega_2$ telle que pour tout $\omega_1 \in \Omega_1$ on a $f(\gamma_1(g)(\omega_1)) = \gamma_2(g)(f(\omega_1))$.

Dans le cas particulier où $G \leq Sym(\Omega)$ et γ est l'identité (ie: $\gamma(g) = g$) alors γ est fidèle, son image est isomorphe à G et nous noterons G l'image de γ. Le reste de cette section repose sur cette configuration particulière.

Orbites

Soit $G \leq Sym(\Omega)$. L'orbite d'un point $\omega \in \Omega$ par l'action de γ est l'ensemble des images de ω par G :

$$\omega^G = \{g\omega \mid g \in G\}$$

La notion d'orbite induit une relation d'équivalence sur Ω dont les classes sont les orbites. Les orbites forment une partition de Ω. Cette propriété est fondamentale pour la réduction d'espace d'états dans le model-checking.

Théorème 1.2. *Soit une action de G sur Ω. Alors pour tout $\omega_1, \omega_2 \in \Omega$, la relation $\approx$ définie par*

$$\omega_1 \approx \omega_2 \Leftrightarrow \omega_1 \in \omega_2^G$$

est une relation d'équivalence.

On dira que γ est *transitive* ou que G est transitif lorsqu'il n'y a qu'une seule orbite. Si G a plusieurs orbites alors G agit transitivement sur chacune ses orbites.

Soit $\Delta \subseteq \Omega$. Alors Δ est un *bloc d'imprimitivité* si pour tout $g \in G$ on a $g\Delta = \Delta$ ou $g\Delta \cap \Delta = \emptyset$. Le groupe G est *primitif* si tous les blocs ont 0, 1 ou $|\Omega|$ éléments. Dans le cas contraire, il est *imprimitif*. Si Δ est un bloc alors l'ensemble des images de Δ est une partition de Ω, appelée *système de bloc*. Il peut y avoir plusieurs systèmes de blocs pour G.

Exemple 1.2.3. *Soit $G = \langle (1,2),(3,4),(1,3)(2,4)\rangle$. L'action de G sur $\{1,\ldots,4\}$ est transitive et imprimitive, car les sous-ensembles $\{1,2\}$ et $\{3,4\}$ sont des blocs.*

Soit $G = S_4$, alors G est transitif et primitif.

Stabilisateurs

Le *stabilisateur* de $\omega \in \Omega$, noté G_ω est l'ensemble des éléments de G qui fixent ω :

$$G_\omega = \{g \in G \mid g\omega = \omega\}$$

Le stabilisateur d'une séquence $\sigma = [\omega_1, \ldots, \omega_k]$ est l'ensemble des $g \in G$ qui fixent σ : $G_\sigma = \{g \in G \mid [g\omega_1, \ldots, g\omega_k] = [\omega_1, \ldots, \omega_k]\}$.

Pour $g \in G$, l'image de $\Psi \subseteq \Omega$ par g est l'ensemble des images des $\psi \in \Psi$: $g\Psi = \{g\psi \mid \psi \in \Psi\}$. Le stabilisateur de Ψ est l'ensemble des éléments qui fixent Ψ:

$$G_\Psi = \{g \in G \mid (\forall \psi \in \Psi)(g\psi \in \Psi)\}$$

Un stabilisateur de point, d'ensemble ou de séquence, est un sous-groupe de G. Les stabilisateurs des points d'une orbite sont conjugués.

L'orbite est une notion *géométrique* : l'orbite d'un point est l'ensemble des lieux que peut visiter le point. La notion de stabilisateur est *algébrique* : c'est l'ensemble des éléments qui fixent le point. Ces deux notions sont reliées par le théorème suivant, fondamental pour le reste de cette thèse :

Théorème 1.3 (Orbite-stabilisateur)**.** *Soit $\gamma : G \to Sym(\Omega)$ une action de G sur Ω. Alors pour tout élément $\omega \in \Omega$, il existe une bijection entre l'orbite de ω et les cosets du stabilisateur de ω:*

1. *Pour tout $a, b \in G$, l'image de ω par a et b est la même si et seulement si a et b appartiennent au même coset de G_ω.*

$$(\forall a, b \in G)(a \cdot \omega = b \cdot \omega \Leftrightarrow aG_\omega = bG_\omega)$$

2. *La taille de l'orbite de ω est égale à l'index du stabilisateur de ω:*

$$|\omega^G| = [G : G_\omega]$$

3. *Le stabilisateur de $g\omega$ est le conjugué du stabilisateur de ω :*

$$G_{g\omega} = gG_\omega g^{-1}$$

Une conséquence du Théorème 1.3 est que l'action de G sur une orbite ω^G est équivalente à l'action de G sur les cosets de G_ω. Cette propriété fondamentale est utilisée dans cette thèse pour construire le groupe des symétries de réseaux.

L'action de G sur les cosets d'un sous-groupe $H \leq G$ est définie, pour tout $a, b \in G$ par :

$$a \cdot bH = abH$$

En choisissant $H = G_\omega$, on obtient le résultat suivant :

Théorème 1.4. *Soit $G \leq Sym(\Omega)$, $\omega \in \Omega$, ω^G l'orbite de ω et G_ω son stabilisateur. Alors l'action de G sur ω^G est équivalente à l'action de G sur les cosets de G_ω.*

1.2.2 Produits de groupes

Cette section présente différents types de *produits de groupe*. Un produit de groupe est une opération qui permet de construire des groupes à partir d'autres groupes.

Produit disjoint

Soit A et B deux groupes. Le *produit direct* C de A et B, noté $A \times B$, est le produit cartésien des éléments de A et B : $\{(a, b) \mid a \in A \wedge b \in B\}$. A et B sont appelés les *facteurs* de C. Les opérations de groupes sont définies point à point :

— $(a_1, b_1)(a_2, b_2) = (a_1a_2, b_1b_2)$, où $a_1, a_2 \in A$ et $b_1, b_2 \in B$;

— $() = (()_A, ()_B)$, où $()_A$(resp. $()_B$) est l'élément neutre de A(resp. B) ;

— $(a, b)^{-1} = (a^{-1}, b^{-1})$

Les sous-groupes $C_1 = A \times \{()_B\}$ et $C_2 = \{()_A\} \times B$ sont normaux dans C et tous les éléments de A commutent avec ceux de B et réciproquement. Pour tout $c \in C$, il existe une paire (a, b) unique telle que $c = ab = ba$. Nous noterons donc l'élément (a, b) par ab.

Exemple 1.2.4. *Soit $A = \langle(1, 2, 3), (1, 2)\rangle$, $B = \langle(4, 5)\rangle$ et le produit direct $C = A \times B$. Alors $((1, 2, 3), ()_B) \in C$ et $(4, 5)(1, 2, 3) = (1, 2, 3)(4, 5) \in C$. De plus $((1, 2), ()_B) \circ ((1, 2, 3), (4, 5)) = (1, 2) \circ (1, 2, 3)(4, 5) = (2, 3)(4, 5) \in C$.*

Un produit disjoint ([85]) est un cas particulier de produit direct où les facteurs agissent sur des ensembles disjoints. L'exemple 1.2.4 est un produit disjoint.

Soit la fonction $moved : G \to \Omega$ qui retourne l'ensemble des points de Ω permutés par un groupe G.

Définition 1.5 (Produit disjoint)**.** *Soit le produit direct $G = L_1 \times \ldots \times L_k \leq Sym(\Omega)$, avec $\Omega \neq \emptyset$.*

Alors G est un produit disjoint *si pour tout $i, j \in 1..k$ avec $i \neq j$, on a $moved(H_i) \cap moved(H_j) = \emptyset$.*

Tous les produits directs ne sont pas des produits disjoints. A titre d'illustration, considérons le groupe G ci-dessous :

$$G = \langle (1,4,7)(2,5,8)(3,6,9), (1,2,3)(4,5,6)(7,8,9) \rangle$$

Le groupe G est isomorphe au produit direct $\langle (1,2,3) \rangle \times \langle (4,5,6) \rangle$. L'isomorphisme associe $(1,2,3)$ à $(1,4,7)(2,5,8)(3,6,9)$ et $(4,5,6)$ à $(1,2,3)(4,5,6)(7,8,9)$. D'où G est un produit direct. Néanmoins l'intersubsection

$$moved(\langle (1,4,7)(2,5,8)(3,6,9) \rangle) \cap moved(\langle (1,2,3)(4,5,6)(7,8,9) \rangle)$$

n'est pas vide et donc G n'est pas un produit disjoint.

Produit couronne

Un produit couronne peut être vu comme un produit direct dont tous les facteurs sont isomorphes sur lequel agit transitivement un groupe en permutant ces facteurs.

C'est un cas particulier de produit semi-direct. Soit A et H deux groupes et $\gamma : H \to Aut(A)$ une action de H sur A. Alors le *produit semi-direct* de A par H, noté $A \rtimes H$, est le produit cartésien de $H \times A$ équipé de la loi de composition :

$$(h_1, a_1) \circ (h_2, a_2) = (h_1 h_2, a_1(\gamma(h_1)(a_2)))$$

Soit A un groupe et $B = A_1 \times \ldots \times A_n$ un produit direct de n copies de A, où chaque facteur est renommé de façon unique, de telle sorte que B est un produit disjoint. Soit $L \leq S_n$ et $\gamma : L \to Aut(B)$ une action transitive de L sur les automorphismes de B qui agit en permutant les coordonnées des facteurs :

$$\gamma(l)(a_1, \ldots, a_n) = (a_{l1}, \ldots, a_{ln})$$

Le produit semi-direct de B par L via γ est appelé *produit couronne* de A par L et noté $A \wr L$. Le produit direct B est un produit disjoint par construction.

Exemple 1.2.5. *Soit $A = \langle (1,2,3), (1,2) \rangle$ et $L = \langle (1,2) \rangle$.*

Alors le produit couronne $A \wr L$ est le groupe :

$$A \wr L = \langle (1,2,3), (1,2), (4,5,6), (4,5), (1,4)(2,5)(3,6) \rangle$$

Le produit direct $B = \langle (1,2,3), (1,2), (4,5,6), (4,5) \rangle$ est un produit disjoint.

Le produit couronne a une propriété importante, dite propriété universelle des actions imprimitives. Soit $G \leq Sym(\Omega)$ un groupe de permutations dont l'action sur Ω est imprimitive (Section 1.2.1). Alors il existe un système de blocs stable par G (ie: pour tout bloc Δ, $g\Delta$ est un bloc). Soit α une action de G sur l'ensemble des blocs par permutations des blocs et L l'image de α. Soit β l'action du stabilisateur G_Δ sur Δ et A l'image de β. Alors G est isomorphe à un sous-groupe du produit couronne $A \wr L$.

Exemple 1.2.6. *Soit* $G = \langle (1,2,3)(4,5,6), (1,4)(2,5)(3,6)\rangle$. *$G$ peut être vu comme le groupe de symétrie d'une grille de* 3 *lignes et* 2 *colonnes.*

L'action de G sur $[1..6]$ *est imprimitive et on a un système de blocs où chaque bloc est une colonne :*

$$\Delta_1 = \{1,2,3\} \quad et \quad \Delta_2 = \{4,5,6\}$$

Le stabilisateur de Δ_1 est $G_{\Delta_1} = \langle (1,2,3)(4,5,6)\rangle$ et $G_{\Delta_2} = G_{\Delta_1}$.

L'image de α est $L = \langle (1,2)\rangle$ puisque $\alpha((1,4)(2,5)(3,6))$ transpose Δ_1 et Δ_2. L'image de β, l'action de G_{Δ_1} sur Δ_1 est le groupe $A = \langle (1,2,3)\rangle$.

Soit le produit couronne $H = A \wr L = (1,2,3) \wr (1,2) = \langle (1,2,3), (4,5,6), (1,4)(2,5)(3,6)\rangle$. Alors on a bien un sous-groupe H' de H avec $H' = \langle (1,2,3)(4,5,6), (1,4)(2,5)(3,6) \leq H$ tel que $G \simeq H'$.

1.2.3 Base et ensemble générateur fort

Soit $G \leq Sym(\Omega)$. Une base est une séquence de longueur m de points de Ω dont le stabilisateur est trivial. À cette séquence est associée une chaîne de stabilisateurs où chaque stabilisateur $G_{(i)}$ stabilise la séquence de longueur $i-1$. L'ensemble de générateurs fort est un ensemble d'éléments de G qui permettent de générer la chaîne de stabilisateur. Cette sous-section présente rapidement ces concepts. On trouvera une présentation plus complète dans [86].

Une *base* B est une séquence $[\beta_1, \beta_2, \ldots, \beta_m]$ telle que le stabilisateur de B dans G est trivial : $G_B = ()$. La séquence B définit la *chaîne de stabilisateurs* :

$$G = G_{(1)} \geq G_{(2)} \geq \ldots \geq G_{(m)} \geq G_{(m+1)} = ()$$

où $G_{(i)}$ est le stabilisateur des $i-1$ premiers points de B, eg $G_{(i)} = G_{[1,\ldots,i-1]}$. Nous ne considérons que les bases pour lesquels $G_{(i+1)}$ est un sous-groupe strict de $G_{(i)}$.

L'orbite d'un point β_i par $G_{(i)}$ est notée $\Delta^{(i)}$. Par le Théorème 1.3, pour tout $i \in 1..n$, il y a une bijection entre $\Delta^{(i)}$ et les cosets de $G_{(i+1)}$ dans $G_{(i)}$. Soit $F^{(i)}$ une transversale $G_{(i+1)}$ dans $G_{(i)}$. Alors pour tout $\gamma \in \Delta^{(i)}$, il existe un g unique dans $F^{(i)}$ tel que $g\beta_i = \gamma$.

Un ensemble de générateurs X de G pour une base B est appelé *ensemble générateur fort* si X permet de génèrer chacun des stabilisateurs :

$$(\forall i \in 0..m+1) \left(\langle X \cap G_{(i)} \rangle = G_{(i)} \right)$$

L'exemple 1.2.7 illustre ces définitions.

Exemple 1.2.7. *Soit $G = S_4$. La séquence $B = [1,2,3]$ est une base pour G car $G_B = \{()\}$. Nous pouvons donner la chaîne de stabilisateurs suivante :*

$$\begin{aligned} G_{(1)} &= \langle (1,2), (2,3), (3,4)\rangle \\ G_{(2)} &= \langle (2,3), (3,4)\rangle \\ G_{(3)} &= \langle (3,4)\rangle \\ G_{(4)} &= () \end{aligned}$$

L'orbite de 1 *dans $G_{(1)}$ est $= \{1,4,2,3\}$, le stabilisateur de la séquence* $[1]$ *est $G_{(2)}$ et une transversale est $F^{(1)} = \{(), (1,4,3,2), (1,2), (1,3,2)\}$.*

L'ensemble $\{(1,2), (2,3), (3,4)\}$ est un ensemble générateur fort. Par contre, l'ensemble $\{(1,2), (1,2,3,4)\}$, qui pourtant génère G, n'est pas un ensemble générateur fort, puisque $X \cap G_{(2)} = \{()\}$.

La structure formée par la base, la chaîne de stabilisateurs et l'ensemble générateur fort est notée BSGS. Les BSGS peuvent se calculer efficacement par l'algorithme de Schreier-Sims [87] à partir d'un ensemble de générateurs quelconques. Différentes variantes de cet algorithme existent qui dépendent notamment de la taille de G.

Les BSGS permettent d'implémenter efficacement de nombreux algorithmes de combinatoire des groupes, comme par exemple le calcul de l'ordre de G, le test d'appartenance d'un élément, le calcul des transversales, ...

La sous-section suivante présente un algorithme de recherche d'éléments de G appelé recherche avec retour arrière.

1.2.4 Recherche avec retour arrière

Cette sous-section donne un aperçu de la méthode la plus connue pour la recherche d'ensembles d'éléments d'un groupe vérifiant une propriété donnée. Par exemple, elle est utilisée pour le calcul du stabilisateur d'un ensemble. On en trouvera une présentation détaillée dans [86].

Soit $G \leq Sym(\Omega)$, $B = [B_1, \ldots, B_m]$ une base pour G et $G = G_{(1)} \geq G_{(2)} \geq \ldots \geq G_{(m)} \geq G_{(m+1)} = ()$ une chaîne de stabilisateurs.

Par le Théorème 1.3, chaque élément de G peut être identifié par une image de B. L'image d'un préfixe de B de longueur l définit un coset de $G_{(l+1)}$.

Exemple 1.2.8. *Reprenons l'exemple précédent (Exemple 1.2.7). La séquence* $[2,4,3]$ *correspond à l'élément* $(1,2,4)$ *puisque* $(1,2,4)[1,2,3] = [2,4,3]$. *Le préfixe* $[1,3]$ *définit le coset de* $(2,3)G_{(3)}$, *c'est à dire tous les éléments* g *de* G *tels que* $g1 = 1$ *et* $g2 = 3$.

Les images de tous les préfixes de longueur l définissent un ordre partiel par inclusion appelé *l'arbre de recherche* et noté $\mathcal{T}$. Le niveau l de $\mathcal{T}$, noté $\mathcal{T}_l$, correspond à tous les cosets de $G_{(l+1)}$. En particulier, le niveau 1 qui ne contient que la racine correspond à G et les feuilles correspondent aux éléments de G.

On suppose l'ordre $(\Omega, \prec)$ tel que $B_1 \prec B_2 \prec \ldots \prec B_m$ et $B_m \prec B$ pour tout $B \in \Omega \backslash B$. Cet ordre induit un ordre lexicographique sur les éléments de G défini, pour tout $g, h \in G$, par :

$$h \prec g \stackrel{\text{déf}}{=} (\exists l \in [1..m])(\forall i < l)(h\beta_i = g\beta_i \wedge h\beta_l \prec g\beta_l)$$

L'ordre $(\Omega, \prec)$ induit aussi un ordre lexicographique sur chacun des niveau $\mathcal{T}_l$ (par comparaison lexicographique des images des préfixes de longueur l de B).

L'algorithme de *recherche avec retour arrière* est un parcours en profondeur d'abord de $\mathcal{T}$, en explorant les sous arbres d'un niveau l dans l'ordre lexicographique.

Généralement, cet algorithme est utilisé pour chercher un sous-groupe ou un coset d'un sous-groupe de G. Dans ce cas, plusieurs techniques permettent de ne pas explorer certains sous arbres de $\mathcal{T}$. Cet algorithme est la solution la plus efficace en pratique pour tous les problèmes de groupe de permutations n'ayant pas de solution polynomiale connue. En particulier, le calcul du stabilisateur d'un sous-ensemble de Ω utilise une recherche arrière.

Exemple 1.2.9. *Soit* $G = \langle (1,2,3), (2,3,4) \rangle$. *Soit la séquence* $B = [1,2]$. B *est une base pour* G. *La chaîne de stabilisateurs est :*

$$G_{(1)} = \langle (1,2,3), (2,3,4) \rangle \geq G_{(2)} = \langle (2,3,4) \rangle$$

La Figure 1.1, tirée de [88], montre l'arbre de recherche construit. Il y a 12 *feuilles qui correspondent à tous les éléments de* G. *Un chemin depuis la racine jusqu'à la feuille donne une image de* $[1,2]$. *Le noeud* 2 *du niveau* 1 *(la racine est au niveau* 0*) définit le coset* $(1,2)(3,4)G_{(2)}$ *qui contient les éléments* $\{(1,2,3), (1,2)(3,4), (1,2,4)\}$.

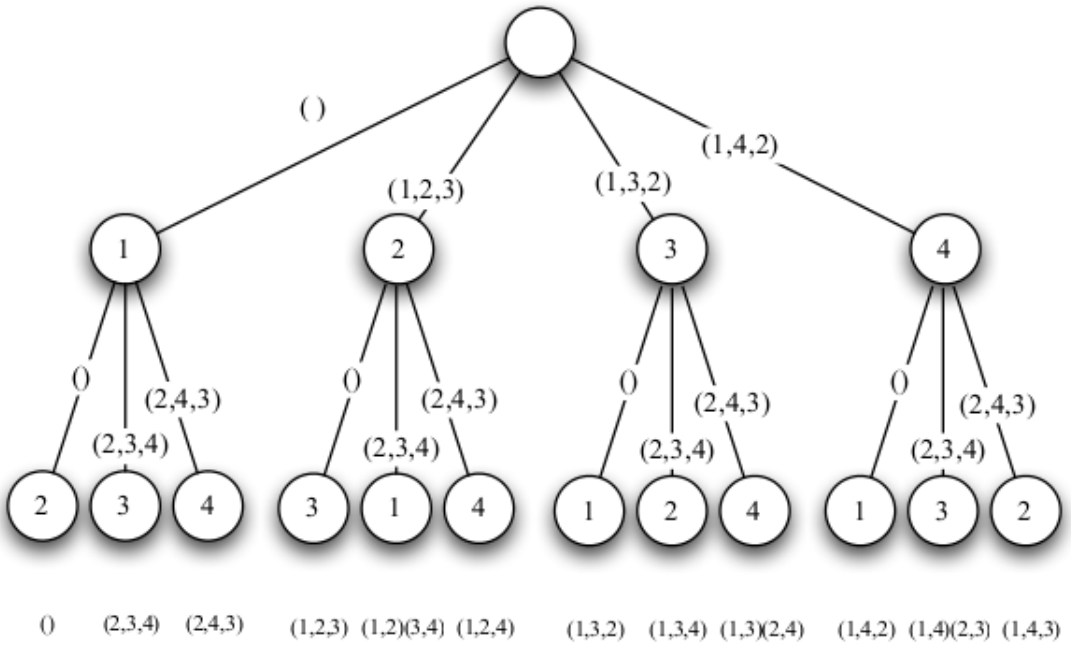


FIGURE 1.1: Exemple d'arbre de recherche pour A_4

1.2.5 Symétries des graphes

Soit $\Gamma = (V, E)$ un graphe orienté où V est l'ensemble de ses sommets, et $E \subset V \times V$ l'ensemble de ses arêtes. Un graphe orienté tel que pour tout $(u, v) \in E$ il existe un unique $(v, u) \in E$ est un graphe non-orienté.

Un automorphisme d'un graphe est une permutation de ses sommets qui préserve les arêtes :

Définition 1.6. *Soit $\Gamma = \langle V, E \rangle$ un graphe orienté, V ses sommets et $E \subseteq V \times V$ l'ensemble des arêtes. Soit G un groupe de permutations sur V. Alors g est un* automorphisme *de Γ si et seulement si :*

$$(\forall i, j \in V)((i, j) \in E \Leftrightarrow (gi, gj) \in E)$$

Chapitre 2

Réduction par symétries

Sommaire

La méthode de réduction par symétries exploite les symétries d'un système pour minimiser le nombre d'états à explorer. Cette méthode construit un quotient de l'espace d'états par la relation d'équivalence induite par les symétries du système. Lorsque les applications ont des structures symétriques, ce qui est généralement le cas pour des applications de taille importante, la réduction par symétrie peut contribuer à la réduction de l'explosion combinatoire. Deux problèmes se posent alors pour l'utilisation des symétries.

Le premier est d'identifier les symétries d'un système, soit automatiquement, soit à partir d'annotations du modèle. Le deuxième problème est l'exploitation de ces symétries pour la réduction de l'espace d'états.

Depuis les premiers travaux exploitant les symétries, pour la vérification de programmes ou de réseaux de Petri de haut niveau, elles ont été utilisées dans de nombreux contextes. Les méthodes d'exploitation des symétries ont été implémentées dans de nombreux outils : [89, 90, 91, 92, 93].

Ce chapitre présente les principales techniques d'identification des symétries et de réduction par symétries en particulier pour les réseaux de Petri.

La section 2.1 présente les symétries des réseaux de Petri et le problème de l'équivalence des marquages. Elle s'inspire de la présentation de [94].

La section 2.2 traite de l'identification des symétries en général puis plus particulièrement pour les réseaux de Petri.

La section 2.3 traite des méthodes de réduction par symétries, pour différents modèles.

Finalement, la section 2.4 présente la méthode dans le contexte des automates temporisés, implémentée dans UPPAAL [95, 93].

A notre connaissance, il n'existe pas de méthode de réduction par symétries pour les TPN. L'objet de cette thèse est la définition d'une méthode de réduction des symétries pour les TPN et propose une solution pour l'identification et l'exploitation des symétries. Une implémentation est présentée. La méthode et son implémentation sont détaillées dans les chapitres 3, 4 et 5.

2.1 Les symétries des réseaux de Petri

Une symétrie d'un réseau de Petri est un automorphisme du réseau vu comme un graphe orienté. C'est une permutation des places et transitions du réseau qui préserve le type des noeuds, la relation de flot et les multiplicités.

Définition 2.1 (Symétrie d'un réseau [96]). *Une* symétrie *(ou automorphisme) d'un réseau Place/Transition N est une permutation $g \in Sym(P \bigcup T)$ qui préserve le type des nœuds, les préconditions et postconditions :*

1. $(\forall x)(x \in P \Leftrightarrow gx \in P)$
2. $(\forall t, p)(\mathbf{Pre}(t)(p) = \mathbf{Pre}(gt)(gp))$
3. $(\forall t, p)(\mathbf{Post}(t)(p) = \mathbf{Post}(gt)(gp))$

Le réseau illustré par la Figure 2.1 contient deux symétries :

$$\{(), (p_0, p_1)(p_2, p_3)(t_0, t_1)(t_2, t_3)\}$$

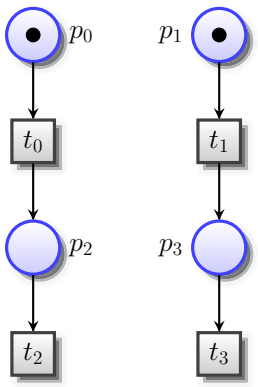


FIGURE 2.1: Réseau Place/Transition N

Un réseau ne contient pas de noeud parallèle si et seulement si pour tout $x, y \in P \cup T$ tel que $\mathbf{Pre}(x) = \mathbf{Pre}(y)$ et $\mathbf{Post}(x) = \mathbf{Post}(y)$ alors $x = y$. Dans ce cas, pour toute permutation g de P, il existe au plus une permutation h de T telle que gh est une symétrie du réseau. La symétrie $(p_0, p_1)(p_2, p_3)(t_0, t_1)(t_2, t_3)$ peut donc s'écrire $(p_0, p_1)(p_2, p_3)$. Cette propriété nous permet de simplifier les notations, mais il est important de noter que les permutations de transitions deviennent essentielles dans le cas des réseaux temporels.

L'ensemble des symétries d'un réseau N équipé une loi de composition forme un groupe, le groupe des automorphismes de N, noté $Aut(N)$.

On note M l'ensemble des marquages accessibles de N. L'action $\gamma : Aut(N) \rightarrow Sym(M)$ de $Aut(N)$ sur M est définie, pour tout $m \in M$ et $g \in Aut(N)$ par :

$$(\forall p \in P)(g \cdot m(p) = m(g^{-1}p))$$

	p_0	p_1	p_2	p_3
m	1	0	0	1
gm	0	1	1	0

FIGURE 2.2: Action d'une symétrie sur un marquage

La Figure 2.2 illustre l'action de $g \in Aut(N)$ sur un marquage.

Le graphe d'accessibilité de N, noté R_N, et les orbites induites par $Aut(N)$ sont illustrées, dans la Figure 2.3. On peut observer l'existence d'automorphismes dans ce graphe, indiqués par des arcs pointillés bleu.

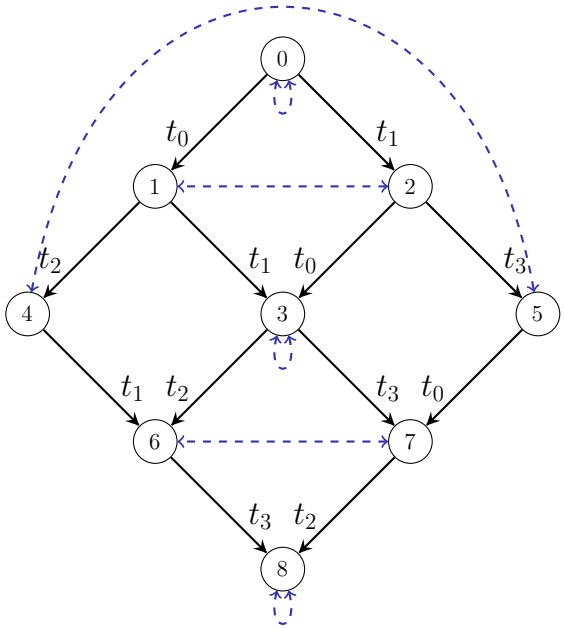


FIGURE 2.3: Orbites dans R_N

Le lemme 2.2, proposé par [96], montre que cela est vrai dans tous les cas : les symétries d'un réseau induisent des automorphismes sur le graphe d'accessibilité de ce réseau.

Lemme 2.2. *Soit $g \in Aut(N)$, $t \in T$ et $m \in M$. On a :*

$$m \xrightarrow{t} m' \iff gm \xrightarrow{gt} gm'$$

Démonstration. Omise. Se référer à [96] □

Deux marquages m_1 et m_2 sont dans la même orbite si il existe un automorphisme g tels que $m_2 = gm_1$. On dira alors que m_1 et m_2 sont équivalents par symétrie.

Les orbites des marquages 0, 3 et 8 sont des singletons. Ces marquages sont *symétriques* et jouent un rôle important dans l'analyse d'accessibilité.

Définition 2.3. *Soit m un marquage de N. m est* symétrique *si et seulement si :*

$$(\forall g \in Aut(N))(gm = m)$$

Par le lemme 2.2, si un marquage m' est atteignable depuis m, alors son image gm' est atteignable depuis gm. De même, si aucun marquage n'est atteignable depuis m alors aucun marquage n'est atteignable depuis gm. En particulier pour $m = m_0$ et si m_0 est symétrique, l'image d'un marquage atteignable est atteignable.

Corollaire 2.4. *Soit m un marquage de M et m_0 symétrique :*

$$m \in M \iff (\exists g \in Aut(N))(gm \in M)$$

Étant donné un réseau N et ses symétries $Aut(N)$, on peut toujours construire un quotient du graphe d'accessibilité, noté $R_{\approx}$. L'algorithme de construction de $R_{\approx}$ se déduit de l'algorithme classique de construction d'un graphe d'accessibilité. L'idée est de n'insérer un nouveau marquage dans $R_{\approx}$ que si il n'existe pas déjà un marquage équivalent. Le graphe $R_{\approx}$ ne contient alors qu'un seul marquage par orbite.

Théorème 2.5. *Soit N un réseau, $Aut(N)$ ses symétries et m_0 le marquage initial. Soit $R_N = \langle M, R, m_0 \rangle$ le graphe d'accessibilité et $R_{\approx} = \langle M', R', m_0 \rangle$ le graphe d'accessibilité quotient. Si le marquage m_0 est symétrique alors un marquage appartient au graphe d'accessibilité si et seulement si il existe un marquage équivalent dans le graphe d'accessbilité quotient :*

$$(\forall g \in Aut(N))(gm_0 = m_0) \Rightarrow (\forall m \in M)(m \in M \iff (\exists m' \in M')(m \approx m'))$$

Démonstration. Omise. Se référer à [96] □

Pour construire le graphe quotient, il faut pouvoir décider si deux marquages sont équivalents par $\approx$. Ce problème, appelé *problème de l'équivalence des marquages* est fondamental dans toutes méthodes de réduction par symétries. C'est un problème au moins aussi difficile que celui de l'isomorphisme de graphe [97].

Définition 2.6 (Problème de l'équivalence des marquages)**.** *Soit N un réseau, $Aut(N)$ ses symétries, M l'ensemble de ses marquages et $\approx$ la relation d'équivalence induite par $Aut(N)$.*

Soit $m_1 \in M$ et $m_2 \in M$ deux marquages. Le problème de décider si m_1 et m_2 sont dans la même classe d'équivalence dans $\approx$, eg

$$m_1 \approx m_2$$

est appelé problème de l'équivalence des marquages.

2.2 Identification des symétries

La mise en oeuvre des méthodes de réduction par symétrie nécessite deux activités : l'identification des symétries puis l'exploitation de la relation d'équivalence induite par ces dernières. Cette section traite du problème de l'identification des symétries d'un modèle : étant donné un formalisme de modélisation et une description formelle d'un système dans ce formalisme, comment calculer le groupe des symétries ?

On peut distinguer trois approches pour l'identification :

— Donner les symétries explicitement

— Annoter le modèle

— Détecter automatiquement les symétries

La première approche peut s'avérer délicate à mettre en oeuvre pour un utilisateur. Dès lors que le modèle est complexe, la description explicite de ses symétries est pratiquement impossible. De plus, un modèle symétrique est souvent construit par duplication de composants qui interagissent de manière symétrique. Cette information doit pouvoir être utilisée plus simplement. Cette solution a néanmoins été proposée par le passé [98, 99],

mais tous les travaux plus récents se concentrent sur les deux autres approches.

Nous limiterons donc notre présentation aux techniques plus envisageables en pratique, où les systèmes à vérifier sont souvent complexes. Nous présentons d'abord les *scalarset*, technique fondatrice pour l'identification des symétries par annotation. Puis nous traitons deux techniques de détection automatique. La première repose sur l'analyse des interactions des processus et la deuxième calcule automatiquement les symétries d'un réseau de Petri.

2.2.1 Les scalarsets

Ip et Dill introduisent le type de données *scalarset* [100] dans le langage de description de Murφ, un langage pour la description de systèmes concurrents asynchrones finis. Un modèle Murφ est constitué de déclarations de constantes, types et variables ; de déclaration de procédures et définitions de séquences de commandes gardées ; de définition des états initiaux et invariants du modèle.

Les états du système sont décrits par des variables globales. Lorsque le système s'exécute les gardes sont évaluées. Parmi les commandes dont la garde est vraie, un choix non déterministe en exécute une de manière atomique. L'exécution met à jour l'état du système. Le modèle de calcul est asynchrone par entrelacement.

Un scalarset est un type de donnée défini comme un ensemble fini non ordonné. Par exemple, il est possible de définir un type *pid* comme un scalarset de taille N:

pid: Scalarset(N)

Les opérations autorisées sur les scalarset sont restreintes à l'affectation, au test d'égalité et à l'indexation des tableaux. Quatre types d'assignation de valeur aux variables scalarset sont possibles :

Ruleset *ID:ScalarsetType* **Do** *ruleset* **Endruleset** : Un **ruleset** choisit de manière non déterministe une valeur dans le scalarset et l'affecte à ID.

For *ID:ScalarsetType* **Do** *stmtseq* **EndFor** : Une itération sur les éléments du scalarset. Le résultat de l'exécution du corps de la boucle doit être invariant par rapport à l'ordre d'exécution.

ForAll *ID:ScalarsetType* **Do** *booleanexpr* **EndForAll** : L'expression *booleanexpr* doit être vraie pour tous les éléments du scalarset.

Exists *ID:ScalarsetType* **Do** *booleanexpr* **EndExists** : L'expression *booleanexpr* doit être vraie pour un élément du scalarset.

Implicitement, un scalarset définit un groupe symétrique sur ses n éléments. L'action de ce groupe sur les variables d'états est définie, pour tout $g \in G$ et toute variable v du type scalarset, par :

- Si v est une variable de type scalarset, l'image de v est gv ;
- Si v est un tableau indexé par le scalarset alors g est d'abord appliquée sur les éléments du tableau, puis les éléments du tableau sont permutés.

L'image de cette action induit un groupe d'automorphismes sur l'espace d'états. La technique utilisée pour construire l'espace d'états quotient est le calcul du représentant minimal d'une orbite. En fonction du nombre et de l'imbrication des variables de type scalarset, le représentant peut être canonique (voir Section 2.3.3).

La technique des scalarsets a été réutilisée dans de nombreux outils, notamment SMC [91], Promela [101], Roméo [102] et Uppaal [95]. Nous trouvons une approche analogue

dans le cas des réseaux de Petri colorés [103, 94, 104], où certaines couleurs(eg: types de données) sont associées à un groupe de permutations. Les opérations attachées aux transitions sont restreintes afin de garantir que les symétries dans le modèle induisent effectivement des automorphismes du graphe des marquages.

L'avantage évident des scalarsets est leur simplicité : les symétries sont déduites des informations de type et les restrictions garantissent que les symétries du modèle induisent effectivement des symétries de l'espace d'états. Un scalarset dénote généralement une symétrie totale ou circulaire. L'indexation des tableaux par des scalarsets génère des produits de groupes. Si les éléments du tableau sont aussi des scalarsets alors on obtient des produits couronnes.

Les scalarsets ne permettent pas de caractériser précisément la structure du groupe de symétrie du modèle. Or nous verrons que cette information est essentielle pour l'optimisation des techniques de réduction. A titre d'illustration, comment déterminer la structure du groupe d'un modèle comprenant un tableau à deux dimensions indexés par deux scalarsets et contenant des variables scalarset. Dans [100], le critère de choix de la méthode la plus adaptée repose sur la complexité de la structure d'un état. Si ce critère est pertinent, la structure du groupe de symétrie l'est tout autant.

2.2.2 Détection automatique

Dans cette section, nous présentons deux techniques représentatives de détection automatique des symétries. La première méthode, implémentée dans l'outil SPIN[105], construit un graphe de communication par analyse d'une spécification Promela. La seconde méthode, implémentée dans l'outil LoLa [92] détecte les symétries de réseaux de Petri via un algorithme par raffinement.

Graphe de communication statique

Cette section présente une méthode de détection automatique des symétries d'un modèle Promela basée sur l'analyse de la structure de communication des processus concurrents [106],[107].

Promela est un langage de spécification pour la vérification des systèmes concurrents. Un modèle Promela est constitué de processus, canaux de communications et variables. Les processus modélisent les entités concurrentes du système. Ils communiquent par passage de messages via les canaux de communications qui sont des tampons. Les interactions entre processus peuvent être asynchrones ou synchrones. Promela propose des types de données simples ou structurés, des instructions de contrôle déterministes ou non déterministes et des actions atomiques.

Soit $\mathcal{P}$ une spécification Promela. Les comportements locaux des processus sont abstraits, pour n'observer que les communications depuis/vers un processus vers/depuis des canaux de communications et les variables partagées. Le *graphe de communication*, noté GC, est un graphe triparti dont l'ensemble des noeuds est composé de processus, de variables partagées et de canaux de communication globaux. Pour chaque émission/réception d'un processus vers/depuis un canal ou une variable partagée, un arc est ajouté entre les noeuds correspondants.

Le groupe des automorphismes du graphe, noté $Aut(GC)$, est alors calculé automatiquement, par des algorithmes de calcul d'automorphismes de graphes. Une action de $Aut(GC)$ sur la spécification $\mathcal{P}$ est définie par permutation des éléments syntaxiques correspondants aux noeuds de GC. Pour un élément $g \in Aut(GC)$, si l'image de $\mathcal{P}$ par g est une spécification équivalente, alors l'automorphisme est valide. Deux spécifications sont

équivalentes si elles ne diffèrent que par certaines permutations d'instructions. Ce test peut se faire efficacement (en ordonnant lexicographiquement ces instructions). Le groupe de symétrie retenu, noté G, est le sous-groupe de $Aut(GC)$ généré par les éléments valides de $Aut(GC)$.

Puisque G est détecté automatiquement, sa structure n'est pas connue. Or, la structure du groupe permet d'adapter la technique de réduction et ainsi d'optimiser l'efficacité de la réduction. Dans [108], les mêmes auteurs implémentent des algorithmes de calcul des groupes pour tester si G appartient à une classe pour laquelle il existe des solutions de réduction efficace ([109]). Nous présenterons plus en détail cette classe dans la Section 2.3.

L'avantage de cette approche est qu'elle est automatique et donc permet de calculer toutes les symétries d'un modèle. De plus elle est capable de décider si une solution de réduction efficace existe. Cependant, le problème d'une approche automatique est celui du passage à l'échelle. En effet, le calcul de $Aut(GC)$ se ramène au problème de l'isomorphisme de graphe. A titre d'illustration, la méthode échoue à détecter les symétries d'un hypercube de dimension 4.

Réseaux de Petri

Cette section présente la méthode de détection automatique des symétries pour les réseaux de Petri proposée dans [110, 111].

Cette méthode repose sur un algorithme de raffinement de partition de l'ensemble des places et des transitions du réseau.

Soit $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ un réseau de Petri. Supposons que l'image des fonctions **Pre** ou **Post** soit l'ensemble $\{0, 1\}$ (la méthode se généralise aisément à des réseaux arbitraires). Une spécification de symétrie est une contrainte de la forme $A \mapsto B$, où A et B sont des sous-ensembles de $P \cup T$. Soit $\mathcal{C}$ l'ensemble de ces contraintes. Une permutation $g \in Sym(P \cup T)$ est consistante avec $\mathcal{C}$ si B est l'image de A pour toutes les contraintes :

$$(\forall A \mapsto B \in \mathcal{C})(g(A) = B)$$

L'ensemble des permutations consistantes avec $\mathcal{C}$ est noté $\Sigma_{\mathcal{C}}$. Il est toujours possible de construire une forme normale de $\mathcal{C}$ où l'union des parties gauche (resp. droite) des contraintes est une partition de $P \cup T$.

Initialement, $\mathcal{C}$ contient une contrainte $T \mapsto T$, une contrainte $P_1 \mapsto P_1$ des places telles que $m_0(p) > 0$ et une contrainte $P_2 \mapsto P_2$ des places telles que $m_0(p) = 0$. Ensuite les contraintes de $\mathcal{C}$ sont raffinées. Intuitivement, pour une contrainte $A \mapsto B$ et pour tout $g \in \Sigma_{\mathcal{C}}$, un noeud x a autant d'arcs pointant vers(resp. depuis) un noeud de A que gx a d'arc pointants vers(resp. depuis) un noeud dans $gA = B$. L'algorithme construit une structure arborescente dont les feuilles décrivent des permutations. Ces permutations ne sont pas nécessairement des symétries. Cependant, il n'est pas possible d'éviter à priori d'explorer les branches qui ne mènent pas à des symétries. Il y a donc des retours en arrière.

Parce qu'il calcule toutes les symétries d'un réseau, l'algorithme est au pire cas en temps exponentiel par rapport à la taille du réseau. Les auteurs proposent une amélioration qui consiste à construire à la volée une BSGS pour le groupe de symétrie du réseau. Par les propriétés des BSGS, il est possible d'éviter des explorations superflues de branches. A la différence de l'algorithme classique de calcul des BSGS de stabilisateurs, aucun ensemble de générateurs n'est connu à priori. Le temps d'exécution pour le calcul d'une ensemble générateurs de taille k est exponentiel en k.

2.2.3 Discussion

Cette section a présenté quelques exemples de méthodes d'inférence des symétries.

L'approche des scalarset présente l'avantage de l'efficacité au détriment de la généralité et de la précision.

A l'opposé, l'approche par détection automatique permet de prendre en compte un groupe de symétries quelconque mais au détriment de l'efficacité (en terme de passage à l'échelle) et de la précision. Les travaux de [108] visent à améliorer la précision en détectant des cas favorables de structure de groupes.

Ces deux méthodes ont en commun la simplicité d'utilisation, par rapport à la description explicite des symétries.

Nous proposons dans le Chapitre 3 une approche intermédiaire : générale, efficace, précise et semi-automatique.

Dans notre solution, l'utilisateur doit avoir une idée des symétries des composants, abstraits par des identifiants, indépendamment de la manière dont ils se synchronisent. Nous faisons l'hypothèse qu'un système symétrique peut s'obtenir par composition et synchronisation des composants. Partant de là, étant donné un sous-ensemble des composants, quelques synchronisations entre les composants et un groupe de permutations G sur les identifiants des composants, notre solution construit le réseau par composition de telle manière que ses symétries soient isomorphes à G. Le groupe de symétries du réseau obtenu est calculé.

La généralité est obtenue par l'absence de contrainte sur le groupe G. N'importe quel groupe de permutations peut être utilisé. Il y a cependant une contrainte sur la présentation du groupe, mais uniquement pour éliminer quelques cas pathologiques. Il n'y a pas non plus de contrainte sur les synchronisations : la construction garantit que les synchronisations calculées ne brisent pas les symétries.

L'efficacité est obtenue par l'approche constructive. La plupart des calculs nécessaires sont polynomiaux. Pour les autres, l'approche constructive permet de donner des conditions suffisantes simples pour les éviter. Nous verrons dans le Chapitre 4 que ces conditions suffisantes correspondent aux cas où des méthodes de réduction efficaces existent.

La précision vient du fait que les symétries du réseaux sont, par construction, isomorphes à celles demandées par l'utilisateur lorsqu'il fournit G. Le groupe G est un groupe de permutations sur des entiers, ce qui est plus simple à présenter qu'un groupe sur des places et transitions. De plus, il est assez facile de proposer des constructions de haut niveau (eg: anneau, pool, hypercube, torus, ...) paramétrées par un nombre de composants, qui évitent à l'utilisateur de décrire G par ses générateurs.

Notre approche est moins simple que celle par détection automatique. L'utilisateur doit fournir un certain nombre de paramètres. Mais la partie la plus difficile dans la construction par composition du réseau et de son groupe de symétrie est le calcul des synchronisations. Notre approche calcule automatiquement ces synchronisations à partir d'un sous-ensemble de transitions. Dans cette thèse, le sous-ensemble de transitions est fourni explicitement, mais il serait assez simple de définir un langage d'étiquetage des transitions des composants qui génère ce sous-ensemble. Par exemple, pour un anneau, des étiquettes spécifiant le successeur, le successeur du successeur, ...

2.3 Méthodes de réduction par symétries

La méthode de réduction par symétrie repose sur la construction du quotient de l'espace d'états par la relation d'équivalence induite par les symétries du modèle. Pendant la construction du graphe réduit, avant d'insérer un état, il faut déterminer si il n'existe pas déjà dans le graphe un état équivalent par la relation de symétrie. Nous reprendrons la

dénomination de Schmidt en appelant ce problème le problème de l'intégration [111] :

> Étant donné un groupe de symétries G, un ensemble d'états M et un nouvel état m_1, existe-t-il un $g \in G$ et un $m_2 \in M$ tel que $g(m_1) = m_2$

Trois solutions sont possibles :

1. Itérer sur les symétries de G
2. Itérer sur les états de M
3. Calculer, à partir de m_1, un représentant canonique de sa classe d'équivalence.

Cette section présente ces solutions appliquées dans le contexte des réseaux de Petri. Ensuite elle présente des classes de groupes pour lesquels des solutions efficaces existent. Finalement, elle présente le cas des automates temporisés et leur traitement dans UPPAAL . Il est important de noter qu'aucun travail n'est disponible pour la réduction par symétries des TPN. La principale contribution de cette thèse est une méthode de réduction par symétries pour les TPN.

Dans cette section, nous supposons défini un réseau $N = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$, un groupe de symétries G et son action sur les marquages de N. Nous noterons gm l'image de m par g dans l'action de G sur M. Nous supposons aussi qu'il existe un ordre total sur $P \cup T$.

2.3.1 Itérer sur les symétries

La méthode consiste à appliquer sur un marquage m toutes les symétries de G et pour chacun des états obtenus vérifier son appartenance à M (Algorithme 2.3.1).

Algorithme 2.3.1 Itérer sur les symétries

```
for all g ∈ G do
    if gm ∈ M then
        return yes
    end if
end for
return no
```

Calculer gm consiste à permuter les places dans le marquage ce qui peut se faire efficacement. Le test de l'appartenance de gm à M dépend de la structure de données utilisées pour représenter M. Par exemple, dans le cas d'un arbre AVL [112], l'accès à un état se fait en moyenne en $O(log(|M|))$.

La principale limitation de cette solution est l'ordre de G. Supposons que G soit isomorphe à S_{10}, alors $|G| = 10! = 3628800$, ce qui rend inapplicable l'itération des symétries. Naturellement, lorsque $|G|$ est petit, par exemple dans le cas des réseaux en anneau, alors l'itération des symétries est une solution convenable.

L'itération des symétries peut être optimisée par l'utilisation d'une BSGS pour G [110, 85].

2.3.2 Itérer sur les états

Cette méthode consiste, pour chaque nouvel état m_1, à considérer tous les états déjà explorés et pour chacun de ces états, dénoté m_2, à chercher un élément de $g \in G$ tel que $gm_2 = m_1$ (Algorithme 2.3.2).

Algorithme 2.3.2 Itérer les états

```
for all m2 ∈ M do
    if ∃g ∈ G : gm2 = m1 then
        return yes
    end if
end for
return no
```

Le test $\exists g \in G : g(m_2) = m_1$ repose sur un algorithme de recherche avec retour arrière (Section 1.2.4). Au pire cas, il faut énumérer tous les éléments de G, et le temps d'exécution dépend de l'ordre de G. Dans [110, 91] les auteurs optimisent cette solution par l'utilisation de fonctions de hachage qui respectent les symétries. Une fonction f de hachage qui respecte les symétries calcule la même valeur pour tous les marquages équivalents :

$$\forall m_1, m_2 \in M, m_2 \approx m_1 \Rightarrow f(m_2) = f(m_1)$$

Si une telle fonction existe, il n'est plus nécessaire d'itérer sur tous les marquages mais seulement sur un sous-ensemble. Cet ensemble est nécessairement une sur-approximation de l'orbite de m_1, sans quoi la fonction est une implémentation du test $(\exists g \in G)(gm_2 = m_1)$.

2.3.3 Calculer un représentant canonique

Cette section présente une solution au problème de l'intégration qui consiste à calculer un représentant canonique d'une orbite. C'est la solution la plus répandue dans la littérature [92, 93, 109, 113].

Définition 2.7 (Représentant canonique)**.** *La fonction* $repr : M \to M$ *est une fonction de* minimisation *si pour tout* $m \in M$ *alors* $repr(m) \approx m$. $repr(m)$ *est appelé un* représentant *de* m.

Si de plus, pour tout $m_1, m_2 \in M$ *tels que* $m_1 \approx m_2$, *nous avons* $repr(m_1) = repr(m_2)$ *alors repr est une fonction de* canonisation *et* $repr(m)$ *est appelé le* représentant canonique.

Généralement, le représentant canonique est choisi comme le plus petit d'une orbite pour un ordre total donné. Nous pouvons par exemple définir un ordre $\leq_M$ sur les marquages à partir de l'ordre sur les places

$$\begin{aligned}(\forall m_1, m_2 \in M)(m_1 \leq_M m_2) &\stackrel{def}{=} \\ &(\forall p \in P)(m_1(p) = m_2(p)) \vee \\ &\quad (\exists p \in P)(\forall q \in P, q < p)(m_1(q) = m_2(q) \wedge m_1(p) < m_2(p))\end{aligned}$$

Dans cette section, nous noterons $min[m]_G$ le représentant d'un marquage m. L'orbite d'un marquage m par l'action de G est finie et donc $min[m]_G$ existe toujours. Lors de la construction du quotient, il faut calculer $min[m]_G$ pour chaque nouvel état m puis insérer $min[m]_G$ dans le graphe.

Nous appellerons *problème constructif de l'orbite*, noté COP, le problème du calcul du représentant canonique d'un état. C'est un problème **NP**-complet [97]. Dans la section suivante, nous donnons des classes de groupes pour lesquels des solutions efficaces au COP existent.

Représentants minimaux

Il est possible de calculer plus efficacement une approximation du COP. Cette approximation consiste à calculer des représentants minimaux et non pas canoniques : le graphe quotient contiendra plusieurs représentants par orbite [114, 111, 100]. En conséquence, le facteur de réduction sera moins important, mais le temps de calcul sera lui aussi moins important.

Une première solution au calcul des représentants minimaux consiste à utiliser un sous-ensemble de G plutôt que G. Dans [109], les auteurs proposent d'utiliser une transversale d'un stabilisateur de séquence. Soit H le stabilisateur de la séquence $1..k$, avec $k < n$. Nous notons X le sous ensemble de G contenant les éléments de la transversale F_H et clos par l'inverse (ie: $g \in X \Leftrightarrow g^- \in X$). Si $G \simeq S_n$, alors l'ordre de H est $|S_{n-k}|$. L'index $|G : H|$ est donc égal à $\frac{n!}{(n-k)!}$ et donc $[G : H] \leq n^k$. Comme la taille de X est polynomiale dans l'index de H, la taille de X est polynomiale par rapport à n.

La deuxième solution repose sur l'utilisation d'une BSGS et d'un algorithme avec retour arrière. Cet algorithme produit une structure arborescente (Section 1.2.4). Une première option est de n'explorer, pour chaque niveau l de l'arbre, que le noeud qui produit un préfixe minimum de longueur l. De cette façon la recherche est guidée par les minimums locaux et on évite le retour arrière mais le résultat n'est pas nécessairement canonique [111]. Une deuxième option est d'utiliser des heuristiques pour décider le noeud à explorer [115]. Enfin d'autres variations sont possibles, toutes basées sur cet algorithme de recherche avec retour arrière [116, 85].

2.3.4 Classes de groupes simples

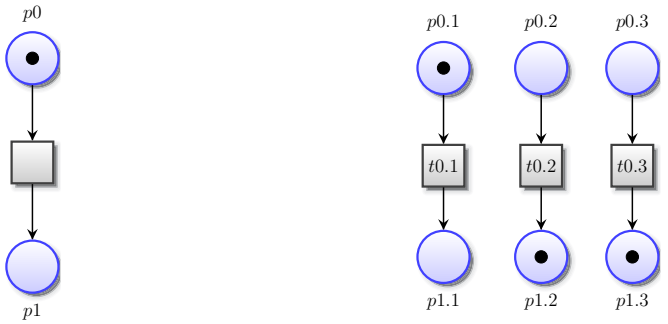
Cette section présente des classes de groupes, identifiées dans [109, 117], pour lesquelles des solutions efficaces au COP existent.

Groupes polynomiaux Si G est un groupe de permutations de degré n et que l'ordre de G est polynomial par rapport à n, alors le COP peut être traité en temps polynomial en itérant les éléments de G. En particulier les groupes cycliques sont d'ordre n et les groupes diédraux (symétries des polygones réguliers à n cotés) sont d'ordre $2n$. Dans ce cas l'énumération des éléments de G est une solution possible.

Groupes isomorphes à S_n Si G est isomorphe à S_n et que l'action de G sur les états est transitive et primitive, alors il suffit de trier lexicographiquement l'état pour obtenir son représentant canonique. Cette solution, initialement présentée dans [114], s'applique bien dans leur contexte, où l'état du système ne dépend que des états locaux des processus et que l'état de chacun des processus peut être abstrait comme un entier.

Dans notre contexte des réseaux de Petri, ce sont des composants qui sont permutés et chaque composant est généralement modélisé par un sous-réseau ayant plusieurs places et transitions. Le tri lexicographique ne s'applique pas.

La Figure 2.4 montre un composant et un système construit avec 3 instances de ce composant. Nous appelons cette construction un *pool* de composants.



(a) Un composant

(b) Pool de 3 composants

FIGURE 2.4: Pool de 3 composants

Chacun des composants est indicé dans le pool et les places et transitions sont indicées par l'index du composant auquel elles appartiennent. Par exemple, les places et transitions du composant 2 sont $\{p0.2, p1.2, t0.2\}$. Le groupe de symétriess du pool est $G \leq S_3$, défini par ses générateurs :

$$G = \langle (p0.1, p0.2)(p1.1, p1.2), (p0.2, p0.3)(p1.2, p1.3) \rangle$$

Soit $m = [1, 0, 0, 0, 1, 1]$ un marquage. Dans ce cas, ordonner m lexicographiquement revient à appliquer la permutation $g = (p_{0.1}, p_{1.1})$, mais g n'est pas une symétrie du réseau. Intuitivement, l'état local d'un composant est un sous vecteur de m et l'action de G sur m permute ces états locaux : le tri lexicographique de m ne correspond pas à des éléments de G [118, 85].

Dans cet exemple, il est possible de présenter m comme une matrice où chaque ligne correspond à une orbite puis de minimiser le marquage en permutant les colonnes de la matrice [85](Table 2.1). Dans la Section 4.3.3 nous donnons les conditions, dans le cas général, pour lesquelles s'applique cette méthode de canonisation par les états locaux.

$p0.1$	1	$p0.2$	0	$p0.2$	0
$p1.1$	0	$p1.2$	1	$p1.2$	1

TABLE 2.1: Marquage du pool Figure 2.4

Produits disjoints Supposons que G soit un produit disjoint $H_1 \times H_2 \times \ldots \times H_k$ (voir Section 1.1). Dans ce cas, il est possible de partitionner le marquage m, chaque partie correspondant à un facteur du produit. Si il existe une solution polynomiale au COP pour chacun des H_i, $i \in 1..k$, alors il existe une solution polynomiale au COP pour G ([117, 85]). Le représentant canonique d'un marquage m peut s'obtenir en calculant successivement le représentant canonique pour chacune des parties.

$$min[m]_G = min[\ldots min[min[m]_{H_1}]_{H_2}] \ldots]_{H_k}$$

Les produits disjoints de groupes constituent une classe que l'on rencontre fréquemment dans les applications du model-checking. Par exemple, un système d'allocation de ressource avec k priorités distinctes est un produit disjoint de k groupes, chacun permutant des processus de même priorité.

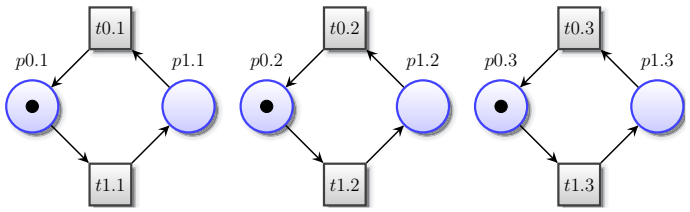


FIGURE 2.5: Produit couronnes $S_2 \wr S_3$

Produits couronnes disjoints Supposons que G soit un produit couronne $H \wr L$. On peut le voir intuitivement comme un produit disjoint où tous les facteurs sont des copies de L et auquel on rajoute des automorphismes qui permutent les facteurs. Un marquage m peut être partitionné de la même façon que dans le cas du produit disjoint.

Le représentant canonique d'un marquage m peut s'obtenir en calculant successivement le représentant canonique pour chacune des parties, puis en permutant les parties :

$$min[m]_G = min[min[\dots min[min[m]_{H_1}]_{H_2} \dots]_{H_k}]_L$$

Si il existe une solution polynomiale au COP pour H et L, alors il existe une solution polynomiale au COP pour G.

La Figure 2.5 montre un pool de 3 composants. Chacun des composants possède une symétrie (ie: $(p0, p1)$). Le groupe de symétries de la Figure 2.5 est isomorphe à $S_2 \wr S_3$. La Table 2.2 montre un exemple de calcul du représentant canonique.

	$p0.1$	$p0.2$	$p0.3$	$p1.1$	$p1.2$	$p1.3$
$m =$	1	1	0	0	1	0
$m' =$	0	1	0	1	1	0
$min[m] =$	0	0	1	1	0	1

TABLE 2.2: Marquage de la Figure 2.5

Les produits couronnes disjoints permettent de capturer les symétries de systèmes dont la structure est arborescente et plus généralement des systèmes où l'action de G est imprimitive. Par exemple, l'architecture client/serveur présentée dans la Figure 2.6, tirée de [119], possède un groupe de symétries isomorphe au produit couronne $S_3 \wr S_2$. Dans cette architecture, les feuilles sont les clients, les noeuds de niveau 1 sont des serveurs web et la racine de l'arbre est un serveur de base de données.

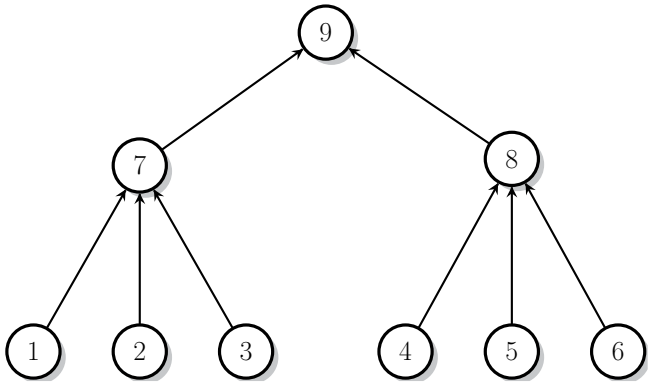


FIGURE 2.6: Architecture client serveur

2.3.5 Discussion

Cette section a présenté un panorama des méthodes de réduction, principalement dans le contexte des réseaux de Petri. Ces méthodes ont des analogues dans la plupart des modèles de vérification. L'efficacité d'une méthode de réduction dépend de la structure du groupe mais aussi de la représentation des états.

Les méthodes qui consistent à itérer sur les états ou les éléments d'un groupe sont sévèrement limitées par la taille des espaces d'états ou des groupes. Néanmoins, ces méthodes s'appliquent quelles que soient la structure des états puisqu'elles utilisent uniquement le test d'égalité entre états.

Les méthodes de calcul du représentant canonique sont les plus utilisées en pratique. Les méthodes basées sur l'algorithme de recherche avec retour arrière ont une complexité au pire cas exponentielle, mais des heuristiques peuvent optimiser la recherche en minimisant le nombre de retours arrière. Les méthodes qui reposent sur la notion de composants disjoints, groupes isomorphes à S_n et produits, sont plus efficaces mais ne peuvent s'appliquer que si il est possible de partitionner un état en état locaux de composants.

Dans ces deux cas, il est indispensable de disposer d'un ordre total sur les états équivalents, de manière à pouvoir calculer le plus petit élément d'une orbite.

La spécificité des modèles temporisés est leur dimension temporelle. Dans les deux modèles que nous considérons, TPN et TA, la dimension temporelle est représentée dans les états par des matrices à coefficients entiers. Nous verrons dans le Chapitre 4 que l'ordre lexicographique sur les matrices ne permet pas l'application des méthodes de calcul du représentant canonique.

2.4 Symétries pour les automates temporisés

Cette section présente une méthode d'exploitation pour les automates temporisés. Plus précisément, elle présente la méthode implémentée dans UPAAL [95, 93], un outil pour la vérification des automates temporisés. L'identification des symétries du modèle repose sur les scalarsets (Section 2.2). La stratégie de réduction utilise le calcul du représentant canonique (Section 2.3).

Un modèle UPPAAL est constitué de processus (ie: des automates finis), d'horloges, de variables et de canaux de communication. A l'exception des canaux de communication, les éléments du modèle peuvent être typés par des scalarsets. Des opérations sont possibles sur ces éléments. On peut par exemple définir les états et transitions des processus, réinitialiser et comparer des horloges, déclarer, initialiser, comparer et modifier des variables, ... Des restrictions sont définies pour les éléments typés par des scalarsets. Par exemple, deux variables typées par un scalarset α ne peuvent être comparées que par l'égalité.

Un état UPPAAL q est un triplet $(\overrightarrow{l}, v, Z)$ où $\overrightarrow{l}$ est le vecteur d'états locaux des processus, v est une valuation des variables et Z une zone (Section 1.2.2). Concrètement, le vecteur des états locaux et la valuation sont implémentés par des tableaux d'entiers. La zone est implémentée par une matrice de différences à coefficients entiers et indexée par les horloges actives.

A chaque point i d'un scalarset α est associé un composant. Ce composant est la projection de q sur les processus, variables et horloges associés à i. Le groupe de permutations implicitement spécifié par α induit une action sur les composants. Sous certaines conditions, les composants sont disjoints. Lorsqu'ils le sont, alors la transposition de deux composants induit un automorphisme de l'espace d'états. Les composants sont transposés successivement, à la manière d'un tri à bulles, jusqu'à obtenir un représentant canonique de q. Pour cela il faut pouvoir comparer les composants et donc disposer d'un ordre sur

ces composants.

La définition de cet ordre est rendue difficile par la dimension temporelle. Une zone est un ensemble de valuations d'horloges : deux horloges x et y indexées dans la zone ne sont pas nécessairement comparables. Il peut exister deux valuations ν et ν' telles que $\nu(x) \leq \nu(y)$ et $\nu'(y) \leq \nu'(x)$.

Lorsque les horloges sont toujours réinitialisées à zéro et que la méthode d'exploration n'utilise pas l'abstraction des enveloppes convexes, alors les zones vérifient la *propriété diagonale*. Cette propriété énonce que si il existe une valuation ν telle que $\nu(x) \sim_Z \nu(y)$ alors pour toutes les valuations ν' capturées par Z on a $\nu'(x) \sim_Z \nu'(y)$, avec $\sim_Z \in \{<, =, >\}$. Dans ce cas, on peut définir un ordre sur les horloges qui est total sur les horloges de processus d'un même scalarset et activées dans q. L'ordre sur les états locaux et variables est un ordre lexicographique sur les vecteurs d'entiers. Cette propriété permet de définir un ordre total sur les composants d'un même scalarset. Étant donné cet ordre il est possible de calculer le représentant canonique de q en temps polynomial par rapport au nombre de processus.

2.4.1 Discussion

La méthode proposée dans cette section et celle que nous proposons pour les TPN diffèrent quant à l'identification des symétries.

L'utilisation des scalarsets implique une perte de précision par rapport à la structure du groupe de symétries extrait du modèle. Il n'est pas possible dans ce cas de choisir une stratégie de réduction adaptée à la structure du groupe. Notre méthode construit un groupe de symétries du réseau isomorphe à celui spécifié dans le modèle. Il est alors possible de choisir une stratégie adaptée.

La méthode proposée dans cette section ignore les synchronisations entre les automates tandis que notre méthode de construction par composition de réseaux les prend en compte.

Les deux méthodes sont similaires sur la stratégie de réduction pour les symétries totales.

Elles reposent toutes les deux sur le calcul des représentants canoniques à partir de composants disjoints, qui permet des réductions très efficaces mais impose des conditions restrictives. Par exemple, les états d'un modèle du protocole d'exclusion mutuelle de Fischer ne vérifient pas ces conditions, à cause d'une variable globale de type scalarset. Cette limitation est une conséquence de l'utilisation des composants disjoints. Notre approche n'y échappe pas.

Notre approche définit un ordre total sur les transitions équivalentes par symétrie dans un domaine de tir. Dans les TPN les horloges sont implicites et attachées aux transitions. Sur cet aspect, l'ordre total sur les horloges proposé pour UPPAAL est analogue au nôtre. En particulier, les deux rendent des horloges comparables dans des structures où elles ne le sont pas en général. Cependant, à cause des différences sémantiques entre le graphe des zones et l'abstraction de classes d'états, le traitement technique est significativement différent.

2.5 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art des méthodes de réduction par symétrie. Ces méthodes font face à deux problèmes : l'identification des symétries et la réduction des espaces d'états.

Ce chapitre a présenté deux approches pour l'identification des symétries : les scalarsets et la détection automatique. Les scalarsets offrent l'avantage de la simplicité pour

l'utilisateur et de l'efficacité au détriment de la généralité et de la précision sur les groupes construits. La détection automatique est la méthode la plus générale et la plus simple mais au détriment de l'efficacité, puisque que ce problème peut se réduire à celui de l'isomorphisme de graphe. Nous proposons dans le chapitre suivant une approche intermédiaire, précise, efficace, générale et semi-automatique. L'utilisateur doit connaitre les symétries des composants, ce qui est aussi le cas pour les scalarsets.

Ce chapitre a présenté les différentes méthodes de réduction par symétries. Les méthodes les plus brutales, qui consistent à itérer sur les symétries d'un groupe ou les états sont limitées par la taille exponentielle des groupes ou des espaces d'états. Cependant, elles sont toujours applicables. Les méthodes exploitant la recherche avec retour arrière comme les méthodes de calcul du représentant canonique nécessitent de disposer d'un ordre total sur les états. Dans le cas des modèles non temporisés cet ordre total est l'ordre lexicographique sur les entiers. Le choses se compliquent pour les modèles temporisés où un tel ordre total n'existe pas en général à cause de la dimension temporelle. Une méthode pour les automates temporisés a été présentée.

Il n'existe pas à notre connaissance de méthode d'exploitation des symétries pour les TPN.

Chapitre 3

Construction des réseaux par composition symétrique

Sommaire

Ce chapitre décrit une méthode de construction de réseaux symétriques par composition symétrique. Par hypothèse, on connait les symétries entre les composants, vus comme des entiers. Les synchronisations entre les composants sont autorisées.

Soit $N_1, N_2, \ldots, N_k$ des composants et pour tout $i \in 1..k$, un entier $n_i \geq 1$. La somme des n_i est notée C et représente les composants de manière abstraite. On peut voir C comme l'ensemble des identifiants de composants. Par hypothèse, les symétries sur C sont connues. Elles sont capturées par un groupe $G \leq S_m$, avec $m \geq |C|$.

Dans ce chapitre, nous proposons un opérateur de composition de réseaux qui permet de construire un réseau symétrique N à partir des $N_1, \ldots, N_k$ et tel que le groupe G' de ses symétries soit isomorphe à G. L'opérateur, noté Π, construit d'abord l'union disjointe M des n_i copies de N_i, pour $i \in 1..k$. Des sous-ensembles des transitions de M sont alors synchronisés pour obtenir N. Certains de ces sous-ensembles sont des paramètres de Π tandis que les autres sont calculés par l'action de G sur M. L'opérateur calcule en même temps une action κ de G sur N. L'image de cette action est un groupe de symétrie de N isomorphe à G.

Nous avons voulu cet opérateur de composition le plus général possible tout en permettant de décider si une stratégie de réduction efficace existe pour N. La solution que nous proposons n'impose qu'une contrainte faible sur le groupe G, uniquement pour garantir l'isomorphisme de G' et G. Autrement, l'opérateur accepte des symétries arbitraires sur les composants et aucune contrainte sur les synchronisations. L'opérateur de composition est récursif, il est donc possible de composer des réseaux symétriques obtenus par composition, tout en connaissant par construction la structure exacte du groupe de symétrie du réseau obtenu. Cette propriété est essentielle pour décider si une méthode de réduction

efficace existe.

Nous faisons face à quatre problèmes.

Le premier est que, au pire cas, la taille d'un groupe est exponentielle par rapport à son degré. Il est donc impossible d'énumérer les éléments d'un groupe. Cependant, il a été montré que le nombre de générateurs d'un groupe est borné par $\log |G|$ [120] et en réalité de nombreux groupes peuvent être générés par un nombre petit de générateurs. L'opérateur de composition n'a besoin que des générateurs de G pour construire G'.

Le second problème est lié aux symétries locales aux N_i : ces composants peuvent être équipés d'un groupe de symétrie. Dans ce cas, il faut que les symétries induites par G et celles de N_i soient compatibles. Plus précisément, si G_i est le stabilisateur de i dans G et A_i le groupe de symétrie de N_i, il faut pouvoir définir une action de G_i sur N_i telle que l'image de cette action soit un sous-groupe de A_i. L'opérateur de composition est paramétré, pour tout i, par une action de G_i sur N_i.

Le troisième problème est lié aux synchronisations. Lorsque un ensemble de transitions de M se synchronisent, il faut calculer les autres synchronisations de telle manière que le réseau obtenu reste symétrique. L'opérateur de composition calcule ces ensembles de transitions en construisant une action de G sur M. Le problème se réduit alors à celui du calcul de l'orbite d'un ensemble.

Le dernier problème est lié à la structure des systèmes que l'on cherche à modéliser. Généralement un réseau de Petri modélisant un système n'est pas totalement symétrique, seuls certains sous réseaux le sont. L'opérateur de composition offre une granularité suffisante qui permet de synchroniser un réseau asymétrique et un réseau symétrique de telle manière que le réseau obtenu reste un réseau symétrique.

3.1 Définitions et notations

Nous commençons par donner la définition d'une symétrie d'un TPN. Cette définition d'une symétrie TPN est très proche de celle des réseaux Place/Transition. Il suffit de considérer que les intervalles statiques sont des étiquettes et de définir que ces étiquettes doivent être préservées par les symétries.

Définition 3.1. *Soit $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, I^s \rangle$ un TPN. Une* symétrie*(ou automorphisme) de N est une permutation $g \in Sym(P \bigcup T)$ qui préserve le type des nœuds, les préconditions, les postconditions et les intervalles statiques.*

1. $(\forall x)(x \in P \Leftrightarrow gx \in P)$
2. $(\forall t, p)(\mathbf{Pre}(t)(p) = \mathbf{Pre}(gt)(gp))$
3. $(\forall t, p)(\mathbf{Post}(t)(p) = \mathbf{Post}(gt)(gp))$
4. $(\forall t)(I_s(t) = I_s(gt))$

Un G-réseau est un triplet formé par un réseau, un groupe de permutations et une action de ce groupe sur le réseau. L'image de l'action est un sous-groupe des automorphismes du réseau. Nous montrerons dans le reste de ce chapitre comment cette action est construite.

Définition 3.2 (G-réseau)**.** *Un G-réseau est un triplet (G, N, κ), où :*

- *$G \leq S_m$ est un groupe de permutations*
- *$N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, I^s \rangle$ est un réseau de Petri temporisé*
- *$\kappa : G \to Sym(P \cup T)$, avec $Img(\kappa) \leq Aut(N)$, est une action de G sur N*

L'image de κ est notée G^κ.

On note $\mathcal{O}$ l'ensemble ordonné des orbites de G. Les composants sont des G-réseaux associés aux points des k premières orbites de $\mathcal{O}$. Il est alors utile de distinguer ces orbites, appelées des sortes.

Une *sorte* est une des k premières orbites de G. L'ensemble des sortes, noté O, est le sous-ensemble de $\mathcal{O}$ contenant ces k premières orbites de $\mathcal{O}$. Pour tout $i \in 1..k$, on note i_1 le représentant de la sorte O_i. Le représentant d'une orbite est un point choisi arbitrairement dans son orbite : $i_1 \in O_i$. La taille de la sorte O_i est notée n_i. L'ensemble C contient les identifiants de composants, ie. autant d'éléments que la somme des n_i :

$$(\forall i \in 1..k)(n_i = |O_i|) \quad \text{et} \quad C = \sum_{i=1}^{k} n_i$$

Par définition de C, il contient moins d'éléments que G n'en permute : $|C| \leq m$. De plus, tout élément de C est plus petit que m : $(\forall c \in C)(c \leq m)$. A chaque représentant de sorte i_1 est associé un réseau noté N_{i_1} et i_1 est l'*identifiant* de N_{i_1}. Nous verrons dans les prochaines sections que tous les réseaux dont l'identifiant appartient à O_i sont des copies isomorphes de N_{i_1}.

Les composants peuvent avoir des symétries locales. Par exemple, supposons que N_i soit synchronisé avec N_j sur une transition t_j et avec N_k sur une transition t_k. Si N_j et N_k peuvent être transposés, alors il faut pouvoir transposer les transitions t_j et t_k tout en fixant N_i. Cette transposition est une symétrie locale à N_i. Cette information est capturée par l'action du stabilisateur de i_1 dans G.

Le stabilisateur de i_1 dans G est noté H_{i_1} et pour chaque H_{i_1} on choisit une transversale à gauche notée F_{i_1}. Les éléments de F_{i_1} sont notés : $F_{i_1} = \{\tau_{i_1}, \ldots, \tau_{i_p}\}$. Par définition d'une transversale, pour tout $\tau_{i_j} \in F_{i_1}$ on a $\tau_{i_j} i_1 = j$.

Pour tout $i \in 1..k$, on définit γ_{i_1}, l'action de H_{i_1} sur N_{i_1}. Nous donnerons la définition d'une action, lorsque c'est nécessaire, comme une fonction sur des séquences de permutations :

$$[a_1, a_2, \ldots, a_n] \to [b_1, b_2, \ldots, b_n]$$

qui signifie que l'image de a_i par l'action est b_i, pour tout $i \in [1..n]$. Nous verrons dans la Section 3.2.1 qu'il est suffisant de définir cette action : l'action de H_{i_j} sur la copie N_{i_j} de N_{i_1} s'obtient par conjugaison de γ_{i_1}.

L'exemple 3.1.1 illustre ces définitions.

Exemple 3.1.1. *Soit* $G = \langle (1,2)(5,6,7), (3,4), (8,9) \rangle$. *Les orbites de* G *sont :*

$$\mathcal{O} = [\{1,2\}, \{3,4\}, \{5,6,7\}, \{8,9\}]$$

On choisit $k = 2$ *sortes. Elles sont de tailles respectives* $n_1 = 2$ *et* $n_2 = 2$ *:*

$$O_1 = \{1,2\} \quad O_2 = \{3,4\}$$

Nous choisissons l'identifiant de composant 1 *comme représentant de* O_1 *et l'identifiant* 3 *comme représentant de* O_2*. Les composants utilisés sont les deux* G*-réseaux :*

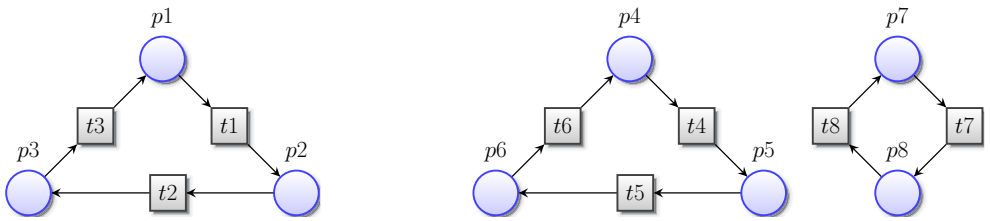
— $(N_{1_1}, C_3, [(1,2,3)] \to [(p1,p2,p3)(t1,t2,t3)])$ *;*

— $(N_{2_1}, C_3 \times S_2, [(1,2,3),(4,5)] \to [(p4,p5,p6)(t4,t5,t6),(p7,p8)(t7,t8)])$.

Le réseau N_{1_1} *(Figure 3.1a) est injecté dans* 1 *et le réseau* N_{2_1} *(Figure 3.1b) dans* 3.

Nous définissons l'action γ_{1_1} *du stabilisateur de* 1_1*,* $H_{1_1} = \langle (5,7,6), (3,4), (8,9) \rangle$*, sur* N_{1_1} *par :*

$$\gamma_{1_1} = [(5,6,7),(3,4),(8,9)] \to [(p1,p2,p3)(t1,t2,t3),(),()]$$



(a) Réseau N_{1_1} (b) Réseau N_{2_1}

FIGURE 3.1: Exemple de 2 composants injectés

De même nous pouvons définir l'action γ_{2_1} *de* $H_{2_1} = \langle (8,9), (5,6,7), (1,2)(5,6,7) \rangle$ *sur* N_{2_1} *par :*

$$\begin{aligned}\gamma_{2_1} =& [(8,9), (5,6,7), (1,2)(5,6,7)] \rightarrow \\ & [(), (p4, p5, p6)(t4, t5, t6), (p7, p8)(t7, t8)(p4, p5, p6)(t4, t5, t6)]\end{aligned}$$

L'opérateur de composition permet de synchroniser des composants. Le produit de transitions, que nous préférerons appeler synchronisation, est une opération connue. Il crée une nouvelle transition à partir d'un ensemble de transitions. La particularité de l'opérateur de composition est de calculer l'orbite de cet ensemble et de synchroniser chacun des points de cette orbite. Néanmoins, l'utilisateur doit spécifier un ensemble de transitions à synchroniser. Nous définissons pour cela les *spécifications de synchronisations*. Comme nous manipulons des TPN, une spécification contient aussi un intervalle de temps. Ainsi, toutes les transitions synchronisées à partir de cette spécification auront le même intervalle statique. Cette dernière propriété est une condition de la définition des symétries des TPN (Définition 3.1).

Définition 3.3 (Spécification de synchronisation)**.** *Soit* $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, I^s \rangle$ *un réseau. On appelle* spécification de synchronisation *un tuple* $(E, \mathcal{I})$ *où* $E \subseteq T$ *et* $\mathcal{I} \in \mathbb{I}$.

Nous donnons une définition légèrement différente de la définition classique des produits de transitions [121]. Pour distinguer notre produit de la définition classique, nous l'appellerons *synchronisation de transitions*.

Une synchronisation de transitions crée une transition à partir d'une spécification de synchronisation $\psi = (E, \mathcal{I})$. Les préconditions(resp. postconditions) de la transition créée sont la somme des préconditions(resp. postconditions) des transitions de E. L'intervalle statique associé à la nouvelle transition est celui donné dans la spécification.

Définition 3.4 (Synchronisation de transitions)**.** *Soit un réseau* $\langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ *et une spécification de synchronisation* $\psi = (E, \mathcal{I})$*. La* synchronisation *de* ψ *est une nouvelle transition* t_ψ*, appelée* synchronisation de ψ*, telle que :*

1. *Pour tout* $p \in P$ *:*

$$\mathbf{Pre}(t_\psi)(p) = \sum_{k \in E} \mathbf{Pre}(k)(p) \quad et \quad \mathbf{Post}(t_\psi)(p) = \sum_{k \in E} Post(k)(p)$$

2. *L'intervalle statique de* t_ψ *est* $\mathcal{I}$.

3.2 Opérateur de composition symétrique

L'opérateur de composition est défini dans cette section. Il est paramétré par un groupe de permutations G, un ensemble de k sortes et autant de G-réseaux. Chaque G-réseau est associé à un représentant de sorte. Le G-réseau associé au représentant i_1 est noté

$(G_{(i_1)}, N_{i_1}, \kappa_{i_1})$. La notation $G_{(i_1)}$, où i_1 est parenthésé, permet de distinguer le groupe du i-ème G-réseau du stabilisateur de i_1 dans G.

L'opérateur est aussi paramétré, pour chacun des G-réseaux $(G_{(i_1)}, N_{i_1}, \kappa_{i_1})$, par l'action γ_{i_1}. Cette action peut être triviale, notée $\emptyset$, c'est à dire que les éléments de H_{i_1} n'ont pas d'effet sur N_{i_1}. Dans ce cas, si une transition de N_{i_1} est synchronisée, la sémantique de la synchronisation permet tout de même d'obtenir des symétries (Section 3.2.2).

Finalement, le dernier paramètre est un ensemble de spécification de synchronisation, éventuellement vide.

Définition 3.5 (Opérateur de composition)**.** *L'*opérateur de composition*, noté* Π*, est défini comme*

$$\Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$$

où ,

- *k est le nombre de sortes ;*
- *$G \leq S_m$ est un groupe de permutations sur $1..m$;*
- *Pour tout $i \in 1..k$:*
  - *N_{i_1} est un G-réseau $(G_{(i_1)}, N_{i_1}, \kappa_{i_1})$;*
  - *$\gamma_{i_1} : H_{i_1} \to Sym(P_{i_1} \cup T_{i_1})$ une action de H_i sur N_{i_1} telle que $Img(\gamma_{i_1}) \leq Img(\kappa_{i_1})$*
- *Ψ est un ensemble de spécifications de synchronisation, avec pour tout $(E_1, I_1), (E_2, I_2) \in \Psi$, soit $E_1 = E_2$ et $I_1 = I_2$ ou $E_1 \neq E_2$;*

Pour tout $g \in G$ tel que g ne permute aucun point de O, alors il existe une sorte O_i telle que $\gamma_{i_1}(g) \neq ()$.

Dans un premier temps l'opérateur crée pour chaque sorte O_i, n_i copies de N_{i_1}. Le réseau obtenu par union de ces copies est appelé l'*union disjointe*, notée M. L'opérateur calcule ensuite une action de G sur M .

Dans un deuxième temps, l'opérateur va synchroniser des ensembles de transitions de M. Pour chaque spécification de synchronisation $\psi \in \Psi$, l'opérateur calcule son orbite dans l'action de G sur M et synchronise chacun des points de cette orbite séparément. Cela implique que les transitions dans Ψ soient des transitions de M.

Le réseau N obtenu par composition symétrique est l'union disjointe après synchronisation. L'action de G sur N se déduit de celle de G sur M. On obtient ainsi le G-réseau (G, N, κ).

Nous détaillons dans les sections suivantes la construction de M et le calcul de l'action de G sur M (Section 3.2.1), puis la synchronisation des transitions de M et le calcul de l'action κ de G sur N (Section 3.2.2).

La contrainte sur le groupe G est nécessaire pour obtenir l'isomorphisme avec le groupe de symétries du G-réseau (G, N, κ). Avec cette condition, l'action κ sera fidèle et donc l'image de κ est isomorphe à G.

Nous illustrons la définition de l'opérateur de composition par l'exemple des trains de Genrich.

Exemple 3.2.1 (Trains de Genrich [122])**.** *Cet exemple montre comment construire le réseau des trains de Genrich par composition symétrique. Le système modélisé consiste en deux trains circulant sur une voie circulaire découpée en sections. Afin de garantir la sûreté du système, deux sections adjacentes ne doivent jamais être occupées simultanément par plus d'un train. Le système est modélisé par un réseau de Petri temporisé.*

Dans ce réseau, une place marquée $U_{i,x}$ modélise la présence du train x dans la section i. Une place V_i modélise un sémaphore permettant à un train d'accéder à la section i.

Nous dimensionnons le système avec 7 sections ($a = 7$) et 2 trains ($b = 2$). Il y a alors $2 \times 7 = 14$ places $U_{i,x}$ et 7 places V_i. Les trains sont indicés dans 0..1 et les sections dans 0..6.

On associe à chaque paire (i, x) un composant $N_{i,x}$. De même, pour chaque sémaphore i, on a un composant N_i. On déduit du dimensionnement qu'il y a 14 composants $N_{i,x}$ et 7 composants N_i. Un composant $N_{i,x}$ et un composant N_i sont illustrés dans la Figure 3.2.

FIGURE 3.2: Composants $N_{i,x}$ et N_i

Un train qui se déplace passe d'une section i à une section $i + 1 \mod 7$. Il faut donc que la transition $s_{i,x}$ de $U_{i,x}$ soit synchronisée avec la transition $r_{p,x}$ de $U_{p,x}$ ($p = i + 1 \mod 7$).

Un train dans une section i peut accéder à la section $i+1$ si et seulement si le sémaphore associée à $i + 1$ est marqué. Il faut aussi synchroniser la transition $s_{i,x}$ du composant $N_{i,x}$ avec la transition u_p du composant N_p.

Lorsqu'un train quitte une section i, il doit rendre le jeton au sémaphore associé à la section $i - 1$, ceci pour maintenir au moins une section vide entre deux trains. Il faut aussi synchroniser la transition $s_{i,x}$ de $N_{i,x}$ avec la transition t_q de N_q ($q = i - 1 \mod 7$).

La Figure 3.3 montre le résultat de ces synchronisations pour le composant $N_{i,x}$. On peut déduire une spécification de synchronisation ψ en fixant $i = 0$, $x = 0$ et un intervalle temporel $[1, 1]$:

$$\psi = (\{s_{0,0}, r_{1,0}, u_1, t_6\}, [1, 1])$$

Lorsque nous aurons défini le groupe G, l'opérateur de composition calculera l'orbite de ψ par l'action de G sur l'union disjointe et synchronisera chacun des ensembles dans l'orbite de ψ. Nous considérons les deux groupes de symétries de ce modèle :

- *Un groupe I isomorphe à C_a, où a est le nombre de sections, qui capture les symétries des sections ;*
- *Un groupe X isomorphe à C_b, où b est le nombre de trains, qui capture les symétries des trains.*

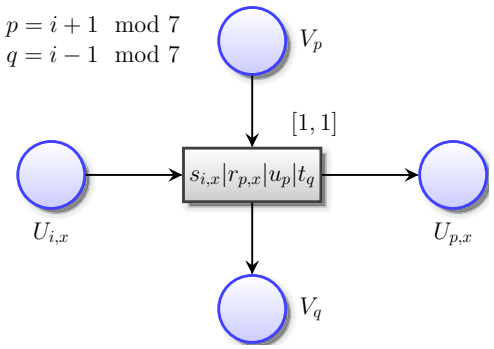


FIGURE 3.3: Produit de la spécification de synchronisation, avec t_ψ couplée à $[1, 1]$

Pour construire le réseau, il nous faut un groupe de permutations qui agit sur les identifiants des composants. On choisit d'indicer les composants $N_{i,x}$ dans 0..13 et les composants N_i dans 14..20. L'index d'un composant $N_{i,x}$ se déduit de i et x. Par exemple, le composant $N_{0,0}$ est à l'index $0 + 7 \times 0 = 0$ tandis que le composant $N_{3,1}$ est à l'index

$3 + 7 \times 1 = 10$. *De même, l'index des composants* N_i *se déduit de* i. *Par exemple, l'index de* N_0 *est* $14 + 0 = 14$ *et celui de* N_6 *est à* $14 + 6 = 20$. *On peut alors déduire le groupe* G *à partir des groupes* I *et* X. *Le groupe* $G = \langle g_1, g_2 \rangle$ *est défini par ses générateurs, avec :*

$$g_1 = (0,1,2,3,4,5,6)(7,8,9,10,11,12,13)(14,15,16,17,18,19,20)$$
$$g_2 = (0,7)(1,8)(2,9)(3,10)(4,11)(5,12)(6,13)$$

La première permutation permute cycliquement et simultanément les composants $N_{i,0}$, $N_{i,1}$ *et* N_i. *La deuxième permutation permute les trains : elle permute les composants* $N_{i,0}$ *et* $N_{i,1}$ *pour tout* $i \in 0..6$. *Les composants* N_i *sont stables par les symétries des trains. Le groupe* G *est isomorphe* $C_7 \times S_2$. *Il a deux orbites,* $0..13$ *et* $14..20$, *qui correspondent aux* 14 *composants de la sorte des composants* $N_{i,x}$ *et aux* 6 *composants de la sorte des composants* N_i.

Nous avons maintenant suffisamment d'informations pour construire le réseau de Genrich. Ce réseau est le résultat de l'opération de composition symétrique ci-dessous :

$$(G, N, \kappa) = \Pi(2, G, N_{0,0}, N_{14}, \{\psi\}, (), ())$$

Afin de donner une intuition de l'action de G *sur l'union disjointe, on peut commencer par renommer les transitions de* ψ *en tenant compte des index des composants. La transition* $s_{0,0}$ *devient la transition* s_0, $r_{1,0}$ *devient* r_1, u_1 *devient* u_{14} *et* t_6 *devient* t_{20}. *Alors l'image de* ψ *par* g_1 *est* $g_1\psi = \{s_1, r_2, u_{15}, t_{14}\} = \{s_{1,0}, r_{2,0}, u_2, t_0\}$, *l'image par* g_2 *est* $g_2\psi = \{s_7, r_8, u_{14}, t_{20}\} = \{s_{0,1}, r_{1,1}, u_1, t_6\}$.

La Figure 3.4 montre le réseau N *obtenu par la composition. Le groupe de ses symétries, notés* G^κ *est isomorphe à* G. *Les générateurs de* G^κ *peuvent être obtenus en appliquant* κ *sur les générateurs de* G. *Ce réseau est marqué manuellement. Nous présentons dans le Chapitre 5 une technique permettant de construire les marquages initiaux.*

3.2.1 Union disjointe des réseaux

Considérons une composition $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$. L'union disjointe des réseaux est une étape intermédiaire dans la construction du G-réseau (G, N, κ). Soit M un réseau vide. Alors l'union disjointe est construite en ajoutant, pour tout $i \in 1..k$, n_i copies de N_{i_1}. Chacune de ces copies a un identifiant unique qui est un élément de C, les places et transitions sont renommées de manière unique. Par définition, tous les composants identifiés par un élément d'un sorte O_i sont isomorphes.

L'intérêt de l'union disjointe est double. Elle est utile pour :

1. Passer du groupe de permutations G sur les indices de composant à un groupe de permutations sur les places et transitions des réseaux.
2. Synchroniser les transitions de ces composants pour obtenir N ;

Le point 1. est traité dans cette section, tandis que le point 2. est traité dans la Section 3.2.2.

Par le Théorème 1.3, qui met en relation l'orbite d'un point avec les cosets de son stabilisateur, on sait que l'action de G sur une sorte O_i est équivalente à l'action de G par multiplication sur les cosets de H_{i_1}. Ces derniers forment une partition de G. La transversale F_i contient exactement un représentant par coset de H_{i_1} qui est choisi arbitrairement. Intuitivement, tous les éléments $g \in G$ tels que $gi_1 = i_r$ appartiennent au coset $\tau_{i_r} H_{i_1}$ avec $\tau_{i_r} \in F_i$ tel que $\tau_{i_r} i_1 = i_r$.

Afin de simplifier la définition de l'action de G sur M, on identifie chaque copie N_{i_r} de N_{i_1} par l'élément $\tau_{i_r} \in F_i$. De cette manière, chaque copie de N_{i_1} est associée à un coset unique de H_{i_1}. Si on abstrait les symétries locales des N_{i_r} alors on obtient une action de G

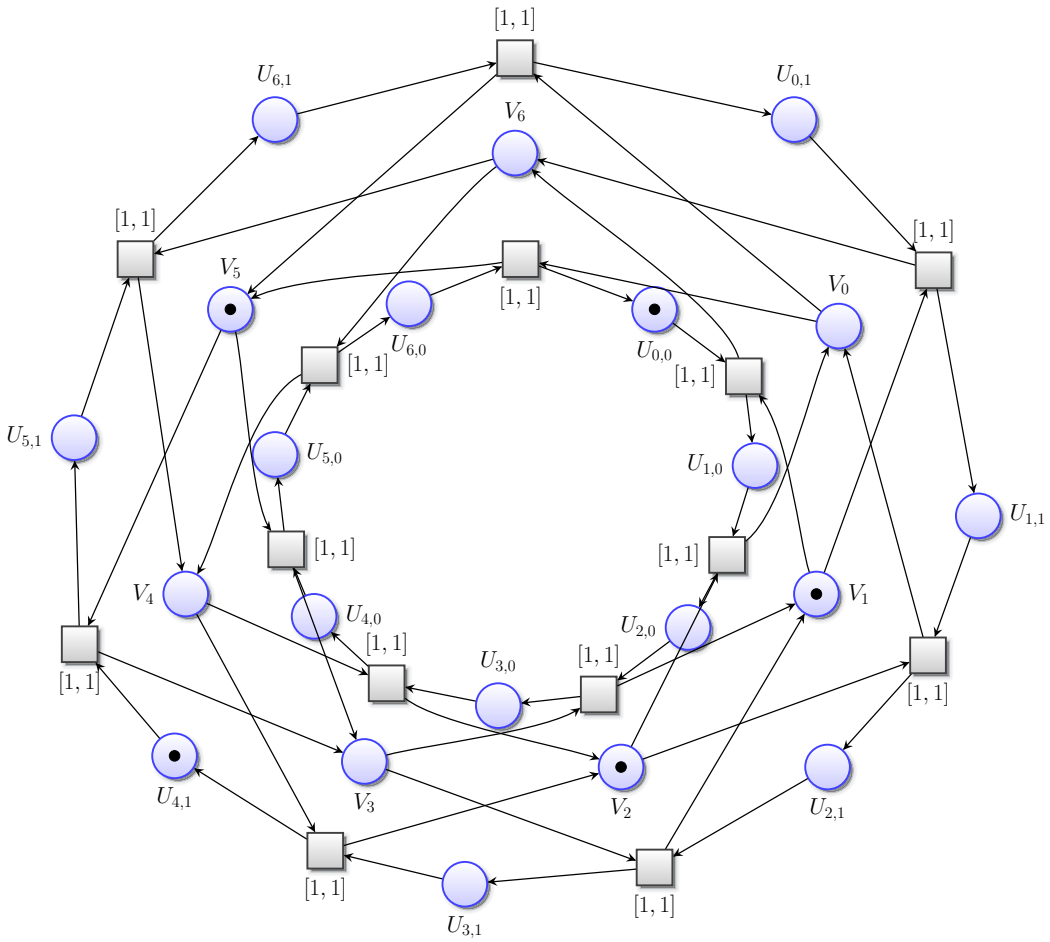


FIGURE 3.4: Trains de Genrich

sur les copies de N_{i_1} équivalente à l'action de G sur les cosets de H_{i_1} et donc équivalente à l'action de G sur O_i. Ce qui est précisément ce que l'on cherche à construire. Néanmoins, principalement à cause des synchronisations, il faut obligatoirement prendre en compte les symétries locales. Nous détaillons cela dans le reste de cette section mais avant nous définissons formellement l'union disjointe.

Définition 3.6 (Union disjointe)**.** *Soit* $\Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ *une composition symétrique. Pour tout* $i \in 1..k$ *et tout* $\tau_{i_r} \in F_{i_1}$*, la* r*-ème copie du* G*-réseau* N_{i_1} *est le* G*-réseau :*

$$(G_{(i_r)}, \langle P_{i_r}, T_{i_r}, \mathbf{Pre}_{i_r}, \mathbf{Post}_{i_r}, I^s_{i_r} \rangle, \kappa_{i_r})$$

Les places et transitions de N_{i_r} *sont renommées de manière unique par un isomorphisme* θ_{i_r}*, le groupe* $G_{(i_r)}$ *est isomorphe à* $G_{(i_1)}$ *par* θ_{i_r} *et les actions* κ_{i_r} *et* κ_{i_1} *sont équivalentes par* θ_{i_r}*.*

L'union disjointe, notée M*, est le réseau*

$$M = \langle P_M, T_M, \mathbf{Pre}, \mathbf{Post}, I^s \rangle$$

où:

— $P_M = \bigcup_{i=1}^{k} \bigcup_{j=1}^{n_i} P_{i_j}$, $T_M = \bigcup_{i=1}^{k} \bigcup_{j=1}^{n_i} T_{i_j}$ *; et*

— *Pour tout* $p, t \in P_M \cup T_M$ *:*

$$\mathbf{Pre}(t)(p) = \begin{cases} \mathbf{Pre}_{i_j}(t)(p) & \text{si } t \in T_{i_j} \\ 0 & \text{sinon} \end{cases} \qquad \mathbf{Post}(t)(p) = \begin{cases} \mathbf{Post}_{i_j}(t)(p) & \text{si } t \in T_{i_j} \\ 0 & \text{sinon} \end{cases}$$

— *Pour tout* $t \in T_M$ *et tout* $i_r \in 1..n_i$ *:*

$$I^s(t) = I^s_{i_r}(t) \quad ssi \quad t \in T_{i_r}$$

Pour tout $i \in 1..k$ et $r \in 1..n_i$, comme il y a une bijection de F_{i_1} vers les copies de N_{i_1}, le noeud $x \in (P_M \cup T_M) \cap (P_{i_r} \cup T_{i_r})$ est noté (τ_{i_r}, x).

Cette notation simplifie la définition des isomorphismes induits par G. Soit $g \in G$ et $\tau_{i_a} \in F_i$ tels que $g\tau_{i_a} = \tau_{i_b}$. Alors on peut définir l'isomorphisme ι entre deux copies N_{i_a} et N_{i_b} comme suit :

$$(\forall x \in (P_M \cup T_M) \cap (P_{i_a} \cup T_{i_a}))(g(\tau_{i_a}, x) = (\tau_{i_b}, x)) \qquad \text{(DEF-}\iota\text{)}$$

La définition ci-dessus ignore les symétries locales de N_{i_a} et N_{i_b}. Dans le cas où H_{i_1} est trivial, alors elle nous donne l'action de G sur les composants de sortes O_i. Mais ce n'est pas suffisant dans le cas général où H_{i_1} n'est pas trivial. Nous prenons maintenant en compte les symétries locales des composants.

Supposons un noeud $(\tau_{i_a}, x) \in M$ et $g \in G$ tels que $g\tau_{i_a} \in \tau_{i_b} H_{i_1}$. Alors, comme chaque composant est associé à un coset, l'image de (τ_{i_a}, x) par g appartient à N_{i_b}. Cela définit l'image globale de N_{i_a} par g (ie N_{i_b}) (Déf. DEF-ι). Soit $\tau^{-1}_{i_a}(\tau_{i_a}, x)$ la projection de (τ_{i_x}, x) sur N_{i_1} et (τ_{i_b}, x) l'image de ce noeud par τ_{i_b}. Lorsque H_{i_1} n'est pas trivial, on a généralement $g(\tau_{i_a}, x) = (\tau_{i_b}, y) \neq (\tau_{i_b}, x)$. Mais comme $g\tau_{i_a} \in \tau_{i_b} H_{i_1}$ par hypothèse, il existe un $h \in H_{i_1}$ tel que $g\tau_{i_a} = \tau_{i_b} h$ et une symétrie locale à $N_{i,b}$ par laquelle l'image de (τ_{i_b}, x) est (τ_{i_b}, y).

Dans l'opérateur de composition symétrique, la relation entre le groupe G et les symétries locales des composants est capturée par les fonctions γ. Il est suffisant de donner γ pour les représentants de sorte uniquement. L'action de $h \in H_{i_1}$ sur N_{i_b}, c'est à dire la symétrie locale de N_{i_b} induite par h, notée $\gamma_{i_b}(h)$, s'obtient par conjugaison :

$$\gamma_{i_b}(h) = \iota(\tau_{i_b})\gamma_{i_1}(h)\iota(\tau^{-1}_{i_b})$$

où ι est l'isomorphisme définit dans DEF-ι. Les copies des N_{i_r} sont isomorphes. Donc toutes les symétries locales existant dans une copie de N_{i_1} existent aussi dans toutes les copies de N_{i_1}.

Lemme 3.7. *Pour tout* $h \in H_{i_1}$ *et tout* $i_a, i_b \in O_i$*, si il existe un* (τ_{i_a}, y) *tel que* $(\tau_{i_a}, y) = (\tau_{i_a}, \gamma_{i_a}(h)x)$ *alors il existe un* (τ_{i_b}, y) *unique tel que* $(\tau_{i_b}, y) = (\tau_{i_b}, \gamma_{i_b}(h)x)$ *et* $\gamma_{i_b} = \iota(\tau_{i_b}\tau^{-1}_{i_a})\gamma_{i_1}(h)\iota(\tau_{i_a}\tau^{-1}_{i_b})$.

Démonstration. N_{i_a} et N_{i_b} sont isomorphes (Définition 3.6). □

Nous savons maintenant qu'il existe un élément $h \in H_{i_1}$ tel que $g\tau_{i_a} = \tau_{i_b} h$ et $(\tau_{i_b}, y) = (\tau_{i_b}, \gamma_{i_b}(h)x)$. Il reste à trouver cet élément. Comme H_{i_1} peut être grand, il n'est pas possible d'énumérer ses éléments. Alors nous utilisons une propriété fondamentale des cosets pour le calculer : les deux éléments $g\tau_{i_a}$ et $\tau_{i_b} h$ appartiennent au même coset de H_{i_1} si et seulement si $g\tau_{i_a} H_{i_1} = \tau_{i_b} h H_{i_1}$. Sachant que $h \in H_{i_1}$, on a de plus $\tau_{i_b} h H_{i_1} = \tau_{i_b} H_{i_1}$. D'où on peut déduire :

$$g\tau_{i_a} H_{i_1} = \tau_{i_b} H_{i_1} \Leftrightarrow \tau^{-1}_{i_b} g\tau_{i_a} \in H_{i_1} \Leftrightarrow (\exists h \in H_{i_1})(\tau^{-1}_{i_b} g\tau_{i_a} = h)$$

Comme τ_{i_u}, τ_{i_r} et g sont connus, on obtient ainsi l'élément $h \in H_{i_1}$ tel que $g\tau_{i_a} = \tau_{i_b} h$. Nous illustrons la prise en compte des symétries locales par l'exemple 3.2.2

Exemple 3.2.2. *Cet exemple illustre la notion de symétrie locale. Soit* $G = S_3 = \langle (1,2), (2,3) \rangle$*. Le stabilisateur* H_1 *est le sous-groupe* $\langle (2,3) \rangle$ *et on choisit la transversale* $F_1 = \{(1,2,3), (1,3)\}$.

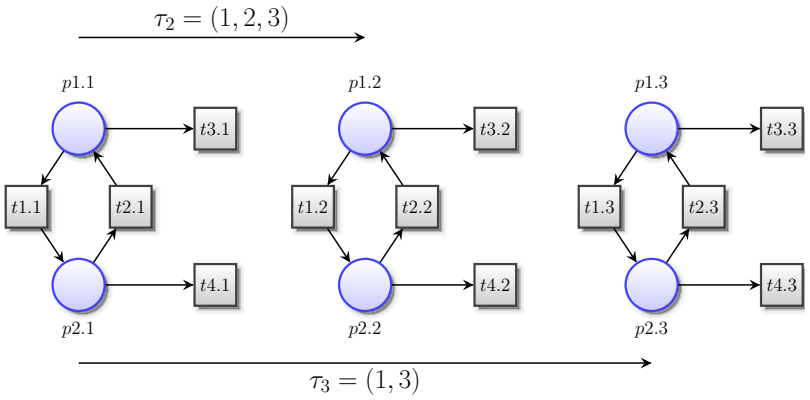


FIGURE 3.5: Exemple d'union disjointe

On suppose l'union disjointe illustrée dans la Figure 3.5. Cette union contient trois composants N_1, N_2, et N_3. Les places et transitions des composants sont suffixées par leur identifiant (eg: p1.1 appartient au composant N_1).

On suppose que l'action de H_1 sur le composant 1 est définie par :

$$\gamma_1 \overset{déf}{=} [(2,3)] \to [(p1.1, p2.1)(t1.1, t2.1)(t3.1, t4.1)]$$

On cherche l'image de la transition $((1,3), t3.3)$ par l'élément $g = (1,3,2)$ de G. On a $\tau_{i_a} = (1,3)$, $g = (1,3,2)$ et comme $(1,3,2) * (1,3) = (1,2) \in (1,2,3)H_1$ on a $\tau_{i_b} = (1,2,3)$. On en déduit l'élément $h = \tau_{i_b}^{-1} g \tau_{i_a} = (2,3)$.

L'action de $h = (2,3)$ sur N_2 est

$$\begin{aligned}\gamma_2((2,3)) &= \iota((1,2,3))\gamma_1((2,3))\iota((1,3,2)) \\ &= \iota((1,2,3))(p1.1, p2.1)(t1.1, t2.1)(t3.1, t4.1)\iota((1,3,2)) \\ &= (t1.2, t2.2)(t3.2, t4.2)\end{aligned}$$

On peut conclure en disant que :

$$(1,3,2)((1,3), t3.3) = ((1,2,3), (t1.2, t2.2)(t3.2, t4.2)t3.2) = ((1,2,3), t4.2)$$

et l'image de t3.3 par $(1,3,2)$ est t4.2.

Nous définissons formellement l'action de G sur M. Soit $\mu : G \to Sym(M)$ la fonction de G vers $Sym(M)$ définie, pour tout $g \in G$ et tout $x \in P_M \cup T_M$, avec $g\tau_{i_a} = \tau_{i_b} h$ et $h \in H_{i_1}$, par :

$$g \cdot (\tau_{i_a}, x) = (\tau_{i_b}, \gamma_{i_b}(h)x) \tag{GM}$$

L'image de G par μ est notée G^μ. Le Théorème 3.8 montre que μ est une action et que son image est un sous-groupe des symétries de M isomorphe à G.

Théorème 3.8. *La fonction $\mu : G \to Sym(M)$ (GM) est une action dont l'image est un sous-groupe des automorphismes de M isomorphe à G :*

$$G \simeq G^\mu \leq Aut(M)$$

Démonstration. 1. μ est une action.

Soit $g_1, g_2 \in G$ tels que $g_1\tau_{i_b} = \tau_{i_c} h_1$ et $g_2\tau_{i_a} = \tau_{i_b} h_2$. On en déduit que

$$g_1 g_2 \tau_{i_a} = \tau_{i_c} h_1 h_2$$

définition de μ, on a $g_2 \cdot (\tau i_a, x) = (\tau i_b, \gamma_{i_b}(h_2)x)$ et

$$\begin{aligned} g_1 \cdot (\tau i_b, \gamma_{i_b}(h_2)x) &= (\tau_{i_c}, \gamma_{i_c}(h_1)\gamma_{i_c}(h_2)x) \quad \text{(Lemme 3.7)} \\ &= (\tau_{i_c}, \gamma_{i_c}(h_1 h_2)x) \quad (\gamma_{i_c} \text{ est une action}) \\ &= g_1 g_2 \cdot (\tau_{i_a}, x) \end{aligned}$$

D'où $g_1 \cdot g_2 \cdot (\tau_{i_a}, x) = g_1 g_2 \cdot (\tau_{i_a}, x)$

2. $Img(\mu) \leq Aut(M)$.

   Nous montrons que pour tout $g \in G$ et tout $t, p \in P_M \cup T_M$, μ préserve les préconditions et les postconditions.

   Soit (τ_{i_a}, t) une transition de M et (τ_{i_b}, p) une place de M, avec $\tau_{i_a} \neq \tau_{i_b}$. Alors par définition de l'union disjointe, $\mathbf{Pre}(\tau_{i_a}, t)(\tau_{i_b}, p) = 0$. Or comme g est une bijection, il est clair que $g(\tau_{i_a}, t)$ et $g(\tau_{i_b}, p)$ n'appartiendront pas au même composant ($g\tau_{i_a} \neq g\tau_{i_b}$). D'où

   $$\mathbf{Pre}(\tau_{i_a}, t)(\tau_{i_b}, p) = 0 = \mathbf{Pre}(g(\tau_{i_a}, t))(g(\tau_{i_b}, p))$$

   Soit (τ_{i_a}, t) une transition de M et (τ_{i_a}, p) une place de M et $g\tau_{i_a} = \tau_{i_b} h$. On a $g = \tau_{i_b} h \tau_{i_a}^{-1}$. Comme μ est une action

   $$\begin{aligned} \mathbf{Pre}(\tau_{i_a}, t)(\tau_{i_a}, p) &= \mathbf{Pre}(\tau_{i_1}, t)(\tau_{i_1}, p) \quad (N_{i_1} \text{ et } N_{i_a} \text{ sont isomorphes}) \\ &= \mathbf{Pre}(\tau_{i_1}, \gamma_{i_1}(h)t)(\tau_{i_1}, \gamma_{i_1}(h)p) \quad (Img(\gamma_{i_1}) \leq Img(\kappa_{i_1}) \leq Aut(N_{i_1})) \\ &= \mathbf{Pre}(\tau_{i_b}, \gamma_{i_b}(h)t)(\tau_{i_b}, \gamma_{i_b}(h)p) \quad (N_{i_1} \text{ et } N_{i_b} \text{ sont isomorphes}) \\ &= \mathbf{Pre}(g(\tau_{i_a}, t))(g(\tau_{i_a}, p)) \end{aligned}$$

   Le même raisonnement tient pour **Post**.

   Nous montrons que les intervalles statiques sont préservés avec les mêmes arguments, avec $g = \tau_{i_b} h \tau_{i_a}^{-1}$:

   $$I_s(\tau_{i_a}, t) = I_s(\tau_{i_1}, t) = I_s(\tau_{i_1}, \gamma_{i_1} t) = I_s(\tau_{i_b}, \gamma_{i_b} t) = I_s(g(\tau_{i_a}, t))$$

   D'où μ préserve les préconditions, les postconditions et les intervalles statiques et donc $Img(\mu)$ est un sous-groupe des symétries de M.

3. $G \simeq Img(\gamma)$.

   Il faut montrer que l'action μ est une action fidèle. C'est à dire que :

   $$(\forall g_1, g_2 \in G)(\exists x \in P_M \cup T_M)(g_1 \cdot x \neq g_2 \cdot x) \tag{1}$$

   Nous notons $G_{|O}$ l'ensemble des éléments de G qui permutent un point d'une sorte. Soit $g_1 \neq g_2 \in G_{|O}$ deux éléments distincts et $i_a \in O_i$ tels que $g_1 i_a \neq g_2 i_a$. L'identifiant i_a existe puisque G est un sous-groupe de S_m et que l'action d'un sous-groupe de S_m sur $1..m$ est fidèle.

   Soit (τ_{i_a}, x) un point de N_{i_a}. Alors :

   $$\begin{aligned} g_1(\tau_{i_a}, x) = g_2(\tau_{i_b}, x) &\Rightarrow g_1 \tau_{i_a} = g_2 \tau_{i_a} \\ &\Leftrightarrow g_1 i_a = g_2 i_a \end{aligned}$$

   Or par hypothèse $g_1 i_a \neq g_2 i_a$. D'où pour tout $g_1 \neq g_2 \in G_{|O}$ il existe un point (τ_{i_a}, x) de M tel que $g_1(\tau_{i_a}, x) \neq g_2(\tau_{i_a}, x)$.

   La définition 1 revient à dire que pour tout $g \in G$ différent de $()$, il existe un point $x \in P_M \cup T_M$ tel que $g \cdot x \neq x$. Or par définition de la composition (Déf. 3.5), pour tout $g \in G \setminus G_{|O}$ il existe un γ_{j_1}, pour $j \in 1..k$, tel que $\gamma_{j_1}(g) \neq ()$.

   Donc μ est fidèle et son image est isomorphe à G.

□

3.2.2 Synchronisation

Cette section décrit les synchronisations de transitions de l'union disjointe et construit une action de G^μ sur le réseau ainsi obtenu. Les ensembles de transitions sont calculés à partir des spécifications de synchronisation Ψ. C'est la deuxième et dernière étape qui permet de construire le G-réseau spécifié par l'opérateur de composition symétrique.

Par le Théorème 3.8 on a G^μ un sous groupe des symétries de M isomorphe à G. Pour chaque spécification $\psi \in \Psi$, les transitions de chacun des ensembles de l'orbite de ψ par G^μ sont synchronisées (Déf. 3.4). Nous définissons l'action ν de G^μ vers les symétries du réseau synchronisé et montrons que ce réseau est un G-réseau (G, N, ν) tel que l'image de ν est isomorphe à G^μ.

Soit $\psi = (E, \mathcal{I}) \in \Psi$ une spécification de synchronisation. L'orbite ψ^{G^μ} est l'ensemble $\{(gE, \mathcal{I}) \mid g \in G^\mu\}$, où gE est l'image par g de E (ie, $\{gx \mid x \in E\}$). La construction de N consiste, pour tout $\psi \in \Psi$, à ajouter les transitions obtenues en synchronisant les transitions de chacun des éléments de ψ^{G^μ}.

Définition 3.9 (Synchronisation de l'union disjointe)**.** *Soit* $\Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ *une composition, M l'union disjointe et $G^\mu \leq Aut(M)$. Pour tout $\psi = (E, \mathcal{I}) \in \Psi$ l'orbite de ψ est définie, avec $gE = \{ge \mid e \in E\}$, par :*

$$\psi^G = \{(gE, \mathcal{I}) \mid g \in G^\mu\}$$

Notons Ψ^{G^μ} l'union des orbites ψ^{G^μ}, pour tout $\psi \in \Psi$:

$$\Psi^{G^\mu} = \bigcup_{\psi \in \Psi} \psi^{G^\mu}$$

Alors la synchronisation de l'union disjointe *est le réseau $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathcal{I}_s \rangle$, avec t_ψ le produit de $(E, \mathcal{I})$, tel que :*

— $P = P_M$ *; et*

— $T = \left(T_M \setminus \bigcup_{(E,\mathcal{I}) \in \Psi^{G^\mu}} E \right) \bigcup \{t_\psi \mid \psi \in \Psi^{G^\mu}\}$

L'action ν de G^μ sur N se déduit alors de l'action de G sur M. Pour chaque $\psi \in \Psi^{G^\mu}$, la synchronisation de ψ est une transition t_ψ et elle est ajoutée dans N. L'image de t_ψ par $g \in G^\mu$ est la synchronisation de l'image de ψ par g. Les places et les transitions non synchronisées de M peuvent être vues comme des singletons.

Soit $\nu : G^\mu \to Sym(N)$ l'action de G^μ sur N définie, pour tout $g \in G^\mu$ et tout $\psi \in \mathcal{O}$, par

$$g \cdot t_\psi = t_{g\psi} \tag{GN}$$

Théorème 3.10. *L'image de l'action $\nu : G^\mu \to Sym(N)$ est un sous-groupe des automorphismes de N :*

$$Img(\nu) \leq Aut(N)$$

Démonstration. Pour tout $p \in P = P_M$ et $g \in G^\mu$ nous avons, par Définition 3.9, $\nu(g)(p) = \mu(g)(p)$. De même, pour toutes les transitions non synchronisées $t \in T \cap T_M$, nous avons $\nu(g)(t) = \mu(g)(t)$. Comme G^μ est un groupe de symétries pour M, il suffit alors de montrer que les préconditions, postconditions et intervalles des transitions synchronisées sont préservés par ν.

Nous montrons que les préconditions des transitions synchronisées sont préservées par ν. Pour tout $\psi = (E, \mathcal{I}) \in \mathcal{O}$, tout $g \in G^\mu$ et tout $p \in P$:

$$\begin{aligned}
\mathbf{Pre}(t_\psi)(p) &= \sum_{t \in E} \mathbf{Pre}(t)(p) \\
&= \sum_{t \in E} \mathbf{Pre}(gt)(p) \quad \text{(Théorème 3.8)} \\
&= \sum_{t' \in gE} \mathbf{Pre}(t')(p) \\
&= \mathbf{Pre}(t_{g\psi})(p)
\end{aligned}$$

Le même argument est utilisé pour les postconditions et donc ν préserve les préconditions et postconditions. Les intervalles des transitions sont invariants par G^μ. D'où, avec $I : T \to I_s$ la fonction qui associe aux transitions leur intervalle statique, pour tout $t' \in t_\psi^{G^\mu}$ on a $I(t') = I(t_\psi)$ et donc ν préserve les intervalles statiques. Nous pouvons conclure que les éléments de l'image de ν sont des symétries de N. □

L'image de ν n'est pas isomorphe à G^μ dans le cas général. Par exemple, supposons que M n'ait aucune place et k transitions qui sont toutes dans la même orbite par G^μ. Si on synchronise ces k transitions alors tous les éléments de l'image de ν fixeront tous les points de N et l'image de ν sera triviale.

Si on rajoute une place dans un composant de M alors chaque copie de ce composant en aura une. Comme on ne synchronise pas les places et que μ est fidèle, les éléments de ν ne fixeront pas ces places et donc ν sera fidèle.

Le Théorème 3.11 montre que dès lors que M contient une place, alors ν est fidèle et donc $Img(\nu) \simeq G^\mu$.

Théorème 3.11. *Si M contient une place, alors l'image de μ est isomorphe à G^μ :*

$$|P_M| \geq 1 \Rightarrow Img(\nu) \simeq G^\mu$$

Démonstration. Supposons que ν n'est pas fidèle :

$$(\exists g \in G^\mu)(g \neq () \wedge (\forall x \in P \cup T)(\nu(g)x = x))$$

Soit x une place de M. Alors c'est aussi une place de N. L'action de μ est fidèle et, par construction, le seul élément de G^μ qui fixe toutes les places est $()$. Si x est une place M alors on a $\nu(g)x = gx$. Donc si g est différent de $()$ il existe au moins un x tel que $\nu(g)x \neq x$. Par contradiction, on a ν est une action fidèle.

D'où $G^\nu \simeq G^\mu$. □

Soit $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ un G-réseau obtenu par composition symétrique. L'action $\kappa : G \to Aut(N)$ de G sur N est définie comme la composition des actions μ et ν :

$$\kappa = \nu \circ \mu$$

Puisque μ et ν sont des actions et que la composition de deux actions est une action, alors κ est une action de G sur N. De plus comme ν et μ sont fidèles, alors κ l'est aussi.

3.3 Réseaux de composants disjoints

La composition symétrique permet de construire des réseaux symétriques. Les symétries d'un G-réseau sont utilisées pour réduire la taille de l'espace d'états, en construisant son quotient par la relation d'équivalence induite par les symétries.

La construction du quotient nécessite de trouver une solution au problème de l'orbite, pour lequel il n'existe pas de solution polynomiale dans le cas général. Autrement dit, nous ne pouvons pas exploiter les symétries de manière efficace dans le cas général.

Cependant, depuis les premiers travaux sur les symétries pour le model-checking, plusieurs classes de réseaux symétriques pour lesquelles des solutions efficaces existent ont été identifiées. Au début, ces cas étaient définis par rapport au groupe de symétries. Une classe de groupe simple a été proposée. Elle est composée des groupes d'ordre polynomial par rapport à leurs degrés, de groupes isomorphes à S_n, de produits disjoints et couronnes de ces groupes.

Plus récemment [108, 93] montrent que la condition sur le groupe n'est pas suffisante. Il y a aussi une condition nécessaire sur l'action du groupe. Intuitivement, il est nécessaire que les orbites soient de même longueur, égales au nombre de composants dans le modèle. Cette propriété n'a été définie que pour le cas où le groupe est isomorphe à S_n.

Cette section définit une classe de réseaux construits par composition, les réseaux à composants disjoints, qui vérifient la condition nécessaire sur l'action du G pour une réduction efficace, quelque soit le groupe G.

3.3.1 Définition

Soit $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ un G-réseau . Un *réseau à composants disjoints*, ou RCD est un cas particulier de la composition symétrique. Nous pouvons décider si un réseau est un RCD par des relations entre les stabilisateurs des noeuds de N dans G^κ et les stabilisateurs des identifiants de composants dans G.

Définition 3.12 (Réseau à composants disjoints (RCD))**.**
Soit $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ un G-réseau obtenu par composition symétrique.

(G, N, κ) est un réseau à composants disjoints *si , pour tout $x \in P \cup T$ tel que $|x^{G^\kappa}| > 1$ alors il existe une orbite i^G telle que :*

$$G^\kappa_x = \kappa(G_i) \text{ et } \forall y \in x^{G^\kappa} \begin{cases} G^\kappa_x = G^\kappa_y \\ (\exists! j \in i^G)(G^\kappa_y = \kappa(G_j)) \end{cases}$$

Si (G, N, κ) est un RCD alors nous pouvons définir une relation d'équivalence entre les noeuds de N à partir des conditions de la Définition 3.12. Dans la partition engendrée par cette relation d'équivalence chaque partie peut être associée à un composant et sera appelée état local du composant. La partie associée au composant i sera stabilisée par $\kappa(G_i)$. Cette partition est une condition nécessaire pour implémenter une réduction efficace. Intuitivement, on peut comparer ces états locaux et minimiser l'état global du système en minimisant le vecteur des états locaux. Cette propriété permet notamment de calculer des représentants canoniques. Nous rentrons dans le détail de cette représentation des états locaux et des réductions dans le Chapitre 4. L'exemple 3.3.1 illustre la définition des RCD.

Exemple 3.3.1. *Soit $G = \langle (1,2,3), (4,5), (5,6) \rangle$ un groupe et*
$(G, N, \kappa) = \Pi(2, G, N_1, N_2, \{\{t0.1, t1.2\}, \{t2.4, t3.5\}\}, \emptyset, \emptyset)$ une composition symétrique. Le réseau N est illustré dans la Figure 3.6 (les réseaux N_1 et N_2 comportent chacun une place avec une transition sortante et une entrante à cette place).

On observe les orbites $o_1 = \{t0.1|t1.2, t0.2|t1.3, t0.3|t1.1\}$, $o_2 = \{p0.4, p0.5, p0.6\}$ et $o_3 = \{t2.4|t3.5, t2.4|t3.6, t2.5|t3.4, t2.5|t3.6, t2.6|t3.4, t2.6|t3.5\}$.

Le stabilisateur des points de l'orbite o_1 est l'image par κ du groupe $\langle (4,5), (5,6) \rangle = G_1 = G_2 = G_3$. Il existe donc une orbite 1..3 dans G telle que pour tout $x \in o_1$, on a $G^\kappa_x = G^\kappa_y$.

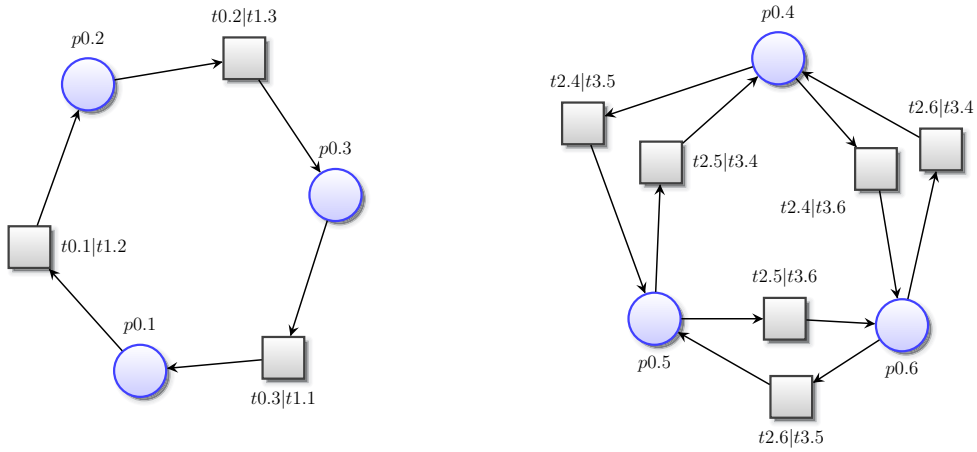


FIGURE 3.6: Illustration de la Définition 3.12

Le stabilisateur de la place $p0.4$ *de* o_2 *est* $G_4 = \langle (1,2,3),(5,6)\rangle$*, celui de la place* $p0.5$ *est* $G_5 = \langle (1,2,3),(4,6)$ *et celui de* $p0.6$ *est* $G_6 = \langle (1,2,3),(4,5)\rangle$*. Dans ce cas, il existe une orbite* $4..6$ *dans* G *telle que pour tout* $y \in o_2$ *il existe un* $j \in 4..6$ *unique tel que* $G_y^\kappa = \kappa(G_j)$*.*

Le stabilisateur de chacun des points de o_3 *est l'image par* κ *de* $H = \langle (1,2,3)\rangle$*. Or il n'existe pas de point* i *dont le stabilisateur est* H*. Les conditions de la Définition 3.12 ne sont pas vérifiées et donc* N *n'est pas un RCD.*

La Définition 3.12 repose sur le calcul de stabilisateurs d'un point et nécessite de tester si deux sous-groupes sont égaux. Le calcul du stabilisateur d'un point est polynomial par rapport au degré de G mais le test d'égalité de sous-groupe est exponentiel par rapport à l'ordre de G. Dans certains cas particuliers, on peut vérifier efficacement et a posteriori si un réseau est un RCD, en utilisant des algorithmes pour les groupes de permutations [108]. Nous ne choisissons pas cette approche et tirons profit de notre méthode par construction plus générale : nous proposons des compositions particulières qui permettent de décider a priori si un G-réseau est un RCD.

A cette fin, il est utile de présenter les relations entre les groupes G, G^μ et G^κ. Plus précisément, on définit les relations entre les stabilisateurs dans ces groupes. Ces relations vont nous permettre de prouver que des configurations particulières de l'opérateur de composition construisent des RCD.

Soit $x \in P \cup T$ un noeud non synchronisé de N tel que x appartient au réseau N_{i_a} de l'union disjointe (ie: $x \in P \cup T \ \cap \ P_{i_a} \cup T_{i_a}$). Alors, en notant (τ_{i_a}, x) ce point, le stabilisateur de x dans G^κ est égal au stabilisateur de (τ_{i_a}, x) dans G^μ :

$$\begin{aligned} G_x^\nu &= \{g \in G^\nu \mid gx = x\} \qquad \text{(DEF-}G_x^\nu\text{)} \\ &= \nu(\{g' \in G^\mu \mid g'(\tau_{i_a}, x) = (\tau_{i_a}, x)\}) \\ &= \nu(G_x^\mu) \end{aligned}$$

Le stabilisateur de x dans G^ν peut à son tour être caractérisé par rapport à G. Un élément g de G fixe le point (τ_{i_a}, x) de M si il fixe i_a et si il fixe localement x.

$$\begin{aligned} G_x^\mu &= \{g \in G^\mu \mid g(\tau_{i_a}, x) = (\tau_{i_a}, x)\} \qquad \text{(DEF-}G_x^\mu\text{)} \\ &= \mu(\{g \in G_{i_a} \mid (\tau_{i_a}, \gamma_{i_a}(h)x) = (\tau_{i_a}, x)\}) \quad (\text{avec } g\tau_{i_a} = \tau_{i_a} h \text{ et } h \in H_{i_1}) \end{aligned}$$

Ce que l'on peut remarquer par les équations DEF-G_x^ν et DEF-G_x^μ, c'est que si l'action γ_{i_a} est triviale, alors le stabilisateur de x dans G^κ est exactement l'image de G_{i_a} par κ.

Lemme 3.13. *Soit $x \in P \cup T$ un noeud non synchronisé appartenant au composant N_{i_a} de M.*

Si γ_{i_1} est triviale alors le stabilisateur de x dans G^κ est isomorphe à G_{i_a} :

$$x \in (P \cup T \ \cap \ P_{i_a} \cup T_{i_a}) \wedge \gamma_{i_1} = \emptyset \Rightarrow G^\kappa_x = \kappa(G_{i_a})$$

Démonstration. Par définition de l'action μ, on a γ_{i_a} triviale ssi γ_{i_1} triviale.

Si γ_{i_a} est triviale alors par l'équation DEF-G^μ_x, $\mu(G_{i_a}) = G^\mu_x$. Par l'équation DEF-G^ν_x, on a $\nu(G^\mu_x) = G^\nu_x$. D'où $G^\nu_x = \nu(\mu(G_{i_a}))$

□

L'exemple 3.3.2 montre un cas où les conditions du Lemme 3.13 ne sont pas vérifiées.

Exemple 3.3.2. *Soit N_{i_1} un réseau avec $P_{i_1} = \{p0, p1, p2\}$ et $T_{i_1} = \emptyset$.*
Soit $(G, N, \kappa) = (1, S_4, N_{i_1}, \emptyset, [(2,3),(2,4)] \to [(p0,p1),(p0,p2)])$. Dans cette composition, aucune synchronisation n'a lieu et l'action de γ_{i_1} n'est pas triviale.

Le groupe G^κ obtenu est isomorphe à S_4. Si on choisit un point x dans N_{i_1} son stabilisateur est isomorphe à S_2, or le stabilisateur de 1 dans S_4 est isomorphe à S_3.

Soit $x \in P \cup T$ un noeud synchronisé. Dans ce cas, il existe une spécification $\psi = (E, I)$, avec $E \subseteq T_M$ l'ensemble de transitions de M dont x est la synchronisation. On ne peut pas caractériser la relation entre un stabilisateur dans G et celui de x dans N aussi précisément que pour la cas où x n'était pas synchronisé.

Le stabilisateur de x dans G^κ est l'image par ν du stabilisateur de ψ dans G^μ.

$$\begin{aligned} G^\kappa_x &= \{g \in G^\kappa \mid gx = x\} \\ &= \nu(\{g \in G^\mu \mid (\forall(\tau_{i_a}, x) \in E)(g(\tau_{i_a}, x) \in E)\} \\ &= \nu(G^\mu_\psi) \qquad \text{(DEF-}G^\mu_\psi\text{)} \end{aligned}$$

On peut déduire de cette équation une relation utile: les éléments de G dont l'image dans G^μ stabilise ψ doivent stabiliser l'ensemble des identifiants des composants auxquels appartiennent les transitions de ψ. La conséquence est que G^κ_x est un sous-groupe de l'image du stabilisateur dans G de cet ensemble d'identifiants.

Lemme 3.14. *Soit $x \in P \cup T$ un noeud synchronisé de N et $\psi = (E, I)$ l'ensemble de transitions de T_M dont x est la synchronisation. Soit ψ_{id} la projection de E sur les identifiants de composants :*

$$\psi_{id} = \{i \in C \mid \exists(\tau_i, x) \in E\}$$

Alors le stabilisateur de x dans G^κ est un sous-groupe de l'image par κ du stabilisateur de ψ_{id} dans G :

$$G^\kappa_x \leq \kappa(G_{\psi_{id}})$$

Démonstration. Soit $g \in G$ et $i_a \in \psi_{id}$ tel que $gi_a = i_x \notin \psi_{id}$. Alors, par définition de μ et de G^μ_ψ, on a $g(\tau_{i_a}, x) \notin \psi$ et donc $\mu(g) \notin G^\mu_\psi$.

Pour tout $g \in G^\mu_\psi$ et tout $(\tau_{i_a}, x) \in \psi$, on a $g(\tau_{i_a}, x) \in \psi$ et donc $\mu^{-1}(g)i_a \in \psi_{id}$. D'où $\mu^{-1}(g) \in G_{\psi_{id}}$.

D'où la préimage par μ de tous les éléments de G^μ_ψ appartient à $G_{\psi_{id}}$. Par l'équation DEF-G^μ_ψ, on a :

$$G^\mu_\psi \leq \mu(G_{\psi_{id}}) \Rightarrow \nu(G^\mu_\psi) \leq \nu(\mu(G_{\psi_{id}}) \Rightarrow G^\kappa_x \leq \kappa(G_{\psi_{id}})$$

□

L'exemple 3.3.3 illustre le Lemme 3.14.

Exemple 3.3.3. *Soit N_{i_1} un réseau avec $P_{i_1} = \emptyset$ et $T_{i_1} = \{t0, t1, t2\}$.*
Soit $(G, N, \kappa) = (1, S_4, N_{i_1}, \{\{t0.1, t1.4\}\}, \emptyset)$. Dans cette composition on synchronise la transition $t0$ du composant 1 avec la transition $t1$ du composant 4. L'action de γ_{i_1} est triviale.

Le groupe G^κ obtenu est isomorphe à S_4. Soit $G^\kappa_{t0.1|t1.4}$ le stabilisateur de la transition synchronisée $t0.1|t1.4$. Ce stabilisateur est isomorphe à C_2.

La projection de $\{t0.1, t0.4\}$ sur les identifiants de composant donne $[1, 4]$. Le stabilisateur $G_{\{1,4\}}$ est un sous-groupe isomorphe à $C_2 \times C_2$. On peut vérifier que $G^\kappa_{t0.1|t1.4} \leq \kappa(G_{\{1,4\}})$.

3.3.2 Conditions sur la composition symétrique

Dans cette section nous donnons trois conditions sur la composition qui permettent de construire des RCD. Ces conditions s'appliquent sur des réseaux à sorte unique, c'est à dire où $k = 1$. Elles sont donc très restrictives.

Cependant, les réseaux obtenus peuvent servir de brique de base dans d'autres compositions, des produits, que nous présentons dans les sections suivantes. L'avantage de ces réseaux est qu'il existe des méthodes de réduction efficaces. De plus, ces symétries sont suffisantes pour la vérification d'architectures symétriques rencontrées en pratique (eg: token ring, architecture client/serveur, ...).

Le Théorème 3.15 donne trois conditions suffisantes sur la composition à sorte unique. Pour chacune d'elle, le G-réseau obtenu est un RCD .

Théorème 3.15. *Soit $(G, N, \kappa) = \Pi(1, G, N_i, \Psi, \gamma_{i_1})$ un G-réseau obtenu par composition symétrique et O_i la sorte unique.*

Si l'une des conditions suivantes est vérifiée alors (G, N, κ) est un RCD

1. *Ψ est l'ensemble vide et γ_{i_1} est triviale ; ou*
2. *Pour tout $\psi = (E, I) \in \Psi$, E coïncide avec une orbite de l'action de G^μ et γ_{i_1} est triviale ; ou*
3. *G est le groupe cyclique C_{n_i}.*

Démonstration. 1. Ψ est l'ensemble vide et γ_{i_1} est triviale.
On peut, sans perte de généralité, ne regarder que les $x \in P \cup T$ tels que $|x^{G^\kappa}| > 1$.
Si $\Psi = \emptyset$ alors tous les noeuds de $P \cup T$ sont des noeuds non synchronisés. Comme γ_{i_1} est triviale, alors par le Lemme 3.13, on a pour tout $x \in N_{i_a}$, avec $a \in [1..n_1]$:

$$G^\kappa_x = \kappa(G_{i_a})$$

Si G_{i_a} est trivial, alors par le Théorème 1.3 et pour tout $i_b \in i_1^G$, G_{i_b} est trivial.
Sinon, pour tout x il existe exactement un $i_a \in i_1^G$ tel que $\kappa^{-1}(G_x) = G_{i_a}$.
D'où N vérifie les conditions de la Définition 3.12 et donc N est un RCD.

2. Pour tout $\psi \in \Psi$, ψ coïncide avec une orbite de G^μ et γ_{i_1} est triviale.
Pour tout $\psi = (E, I) \in \Psi$ si $E = t^{G^\mu}$ avec $t \in T_M$, alors le stabilisateur G^μ_ψ est G.
Soit x la transition produit de ψ. Par l'équation DEF-G^μ_ψ :

$$G^\kappa_x = \nu(G^\mu_\psi) = \nu(G^\mu) = \kappa(G)$$

D'où l'orbite de x est triviale. Par hypothèse, toutes les spécifications de Ψ coïncident avec des orbites de G^μ. D'où toutes les orbites des noeuds synchronisés de $P \cup T$ sont triviales.

Comme γ_{i_1} est triviale, on peut se ramener au cas 1. pour les noeuds non synchronisés. D'où N vérifie les conditions de la Définition 3.12 et donc N est un RCD.

3. $G = C_{n_i}$ est un groupe cyclique.
   Comme $G = C_{n_i}$, le stabilisateur d'un point dans G est trivial :

$$(\forall i_a \in i_1^G)(G_{i_a} = \{()\})$$

   En conséquence, l'action de γ_{i_1} de G_{i_1} vers $Aut(N_{i_1})$ est nécessairement triviale.

   Soit $\psi = (E, I) \in \Psi$, ψ_{id} la projection de E sur les identifiants de composants et x la transition de $P \cup T$ produit de ψ.
   Si $\psi_{id} = 1..n_i$ alors $G_{\psi_i d} = G$ et l'orbite de x est triviale. Sinon $G_{\psi_i d} = \{()\}$ et l'orbite de x est de longueur n_i. Sans perte de généralité, on se limite au cas où $G_{\psi_i d} = \{()\}$.
   Par Le Lemme 3.14, on a $G_x^\kappa \leq \kappa(G_{\psi_{id}})$. Or $G_{\psi_{id}}$ est trivial, d'où

$$G_x^\kappa = \{()\}$$

   D'où il existe une orbite i_1^G telle que $G_x^\kappa = \kappa(G_{i_1}) = \{()\}$. Par le Théorème 1.3, tous les stabilisateurs des points de l'orbite de x sont aussi triviaux. D'où pour tous les noeuds synchronisés leur stabilisateur dans G^κ est trivial.
   Comme γ_{i_1} est triviale, on peut se ramener au cas 1. pour les noeuds non synchronisés.
   D'où N vérifie les conditions de la Définition 3.12 et donc N est un RCD.

□

La première condition du Théorème 3.15 permet de construire des réseaux sans synchronisation et n'a que peu d'utilité en tant que telle. Nous l'utilisons comme une construction intermédiaire pour des produits.

La deuxième condition permet de synchroniser tous les composants sur une même transition et d'obtenir un RCD. Cette condition est plus utile en pratique. L'exemple 3.3.4 montre une construction à partir de S_3.

Exemple 3.3.4. *Soit la composition symétrique* $(G, N, \kappa) = \Pi(1, S_3, N_1, \{\{t1.1, t1.2, t1.3\}\}, \emptyset)$. *Le réseau* N_1 *est contient deux places* $p0$ *et* $p1$*, une transitions* $t0$ *sortant de* $p0$ *et entrant dans* $p1$*, et une transition* $t1$ *sortant de* $p1$ *et entrante dans* $p0$.

Tous les composants vont se synchroniser sur leur transition $t1$*. L'ensemble* $\{t1.1, t1.2, t1.3\}$ *correspond à une orbite de* G^μ*. Le réseau obtenu est un RCD : l'orbite de* $t1.1|t1.2|t1.3$ *est triviale. Il est illustré dans la Figure 3.7. Le groupe de symétries de* N *est décrit par ses générateurs :*

$$G^\kappa = \langle (p1.1, p1.2, p1.3)(p0.1, p0.2, p0.3)(t0.1, t0.2, t0.3),$$
$$(p1.1, p1.2)(p0.1, p0.2)(t0.1, t0.2)\rangle$$

La troisième condition montre que tous les G-réseaux ayant une symétrie en anneau sont des RCD, et ce quelles que soient les synchronisations spécifiées. Même si le groupe cyclique est un groupe petit, les systèmes dont la structure est un anneau sont assez fréquents. Il est donc utile de savoir qu'un modèle construit par composition symétrique est un RCD. L'exemple 3.3.5 montre une construction à partir de C_4.

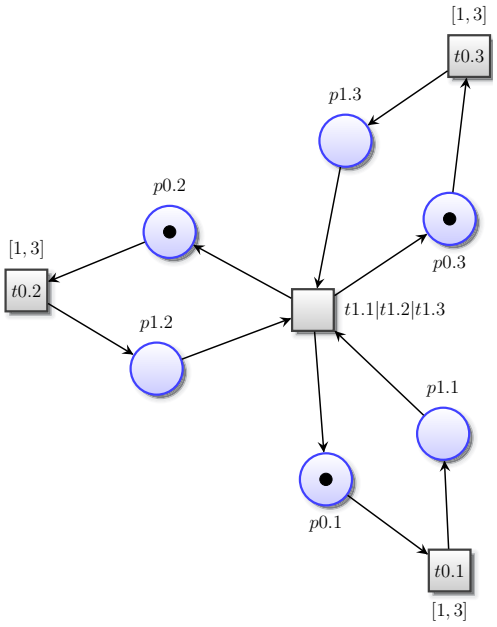


FIGURE 3.7: Exemple de composants synchronisés sur une même transition

Exemple 3.3.5. *Soit la composition symétrique* $(G, N, \kappa) = \Pi(1, C_4, N_1, \{\{t1.1, t0.2\}, \{t1.1, t0.3\}\}, \emptyset)$. *Le réseau* N_1 *contient une place* $p0$, *une transition* $t1$ *sortante de* $p0$ *et une transition* $t0$ *entrante dans* $p0$.

Deux synchronisations sont spécifiées. A partir de la spécification $\{t1.1, t0.2\}$, *pour tout* $i \in 1..4$, *un composant* i *va synchroniser sa transition* $t1$ *avec la transition* $t0$ *du composant* $i + 1 mod 4$.

A partir de la spécification $\{t1.1, t0.3\}$, *un composant* i *va synchroniser sa transition* $t1$ *avec la transition* $t0$ *de* $i + 2 mod 4$. *Le réseau obtenu est un RCD : tous les stabilisateurs sont triviaux. Il est illustré dans la Figure 3.8. Le groupe de symétries de* N *est décrit par un générateur :*

$$\begin{aligned} G^{\kappa} = \langle & (p0.1, p0.2, p0.3, p0.4) \\ & (t0.1|t1.4, t0.2|t1.1, t0.3|t1.2, t0.4|t1.3) \\ & (t0.1|t1.3, t0.2|t1.4, t0.3|t1.1, t0.4|t1.2) \rangle \end{aligned}$$

La condition qui permet aux compositions basées sur C_n d'être des RCD est que le stabilisateur d'un point dans G est trivial. Nous conjecturons que toutes les compositions construites à partir d'un groupe de G dont l'action sur $1..m$ est régulière permettent de construire des RCD. L'action d'un groupe sur un ensemble est régulière lorsque le stabilisateur de tous les points de l'ensemble est trivial.

3.3.3 Produits

Les RCD peuvent être utilisés comme briques de base dans la construction de réseaux plus grands. Nous définissons deux opérations de produits sur les RCD, le produit disjoint et le produit couronne.

Un produit disjoint est une composition où le groupe G peut se décomposer en un produit direct de facteurs disjoints. Il y a une sorte par facteur. Le produit disjoint est très utile pour construire des réseaux où certains sous-réseaux sont symétriques et d'autres sont asymétriques. En effet, lorsqu'un des facteurs est trivial (et si l'action γ correspondante est triviale), la sorte associée ne contiendra qu'un seul élément et aucune symétrie n'agira sur le composant de cette sorte.

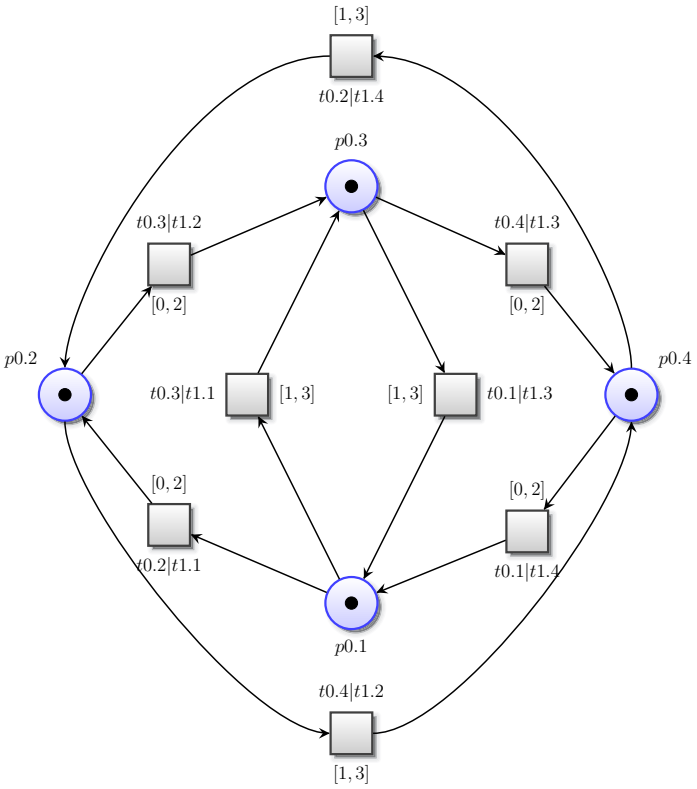


FIGURE 3.8: Exemple de symétries circulaires

Produits disjoints

Soit $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ un G-réseau obtenu par composition symétrique. tel que G peut se décomposer en un produit direct de l facteurs : $G = K_1 \times \ldots \times K_l$. Une condition nécessaire pour que (G, N, κ) soit un produit disjoint est que le nombre de facteurs soit égal aux nombre de sorte, eg $k = l$. Cette condition n'est pas vraie dans le cas général, comme l'illustre l'exemple 3.3.6.

Exemple 3.3.6. *Soit* $G = \langle (1,2)(3,4,5,6), (7,8) \rangle$ *et la composition symétrique :*

$$\Pi(3, G, N_1, N_2, N_3, \Psi, \gamma_{1_1}, \gamma_{2_1}, \gamma_{3_1})$$

Dans cette composition, on a $k = 3$, $n_1 = 2$, $n_2 = 4$ *et* $n_3 = 2$. *Or le groupe* G *contient deux facteurs* $\langle (1,2)(3,4,5,6) \rangle$ *et* $\langle (7,8) \rangle$ *que l'on ne peut pas décomposer comme un produit de trois facteurs* $K_1 \times K_2 \times K_3$.

Néanmoins les produits disjoints permettent de construire des réseaux utiles en pratique. Nous proposons la définition suivante des produits disjoints.

Définition 3.16 (Produit disjoint)**.** *Soit le* G*-réseau* $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ *obtenu par composition symétrique.*

Alors (G, N, κ) *est un* produit disjoint *si toutes les conditions suivantes sont vérifiées:*

1. *Pour tout* $i \in 1..k$, *on a* $n_i = 1$;
2. *le groupe* G *est le produit de* k *facteurs disjoints* $K_{(1)} \times \ldots \times K_{(k)}$*:*

$$G = \{k_1 \ldots k_k \mid (\forall i \in 1..k)(k_i \in K_{(i)})\}$$

3. *pour tout* $i \in 1..k$, $(K_{(i)}, N_{i_1}, \kappa_i)$ *un* G*-réseau obtenu par composition symétrique ;*

4. *pour tout $i \neq j \in 1..k$, $K_{(j)}$ n'agit pas sur les réseaux de la sorte O_i et l'image de $K_{(i)}$ par γ_{i_1} est le groupe G^{κ_i} :*

$$\gamma_{i_1}(G) = \gamma_{i_1}(K_{(i)}) = K_{(i)}^{\kappa_i}$$

5. *Il n'y a pas de synchronisation ou pour toute spécification $\psi = (E, I) \in \Psi$ telle qu'il existe une transition t_i dans l'intersection $E \cap N_{i_1}$ (notée E_i) avec $K_{(i)}$ non trivial, alors t_i est unique et pour tout $t \in N_{j_1} \cap E$ avec $i \neq j$, on a $K_{(j)}$ trivial :*

$$\begin{aligned} &\Psi = \emptyset \\ &\vee\ (\forall (E, I) \in \Psi)(\exists! t_{i_a} \in E_i \Rightarrow |E_i| = 1 \wedge (\forall i \neq j)(\forall t_j \in E_j)(K_{(j)} = \{()\})) \end{aligned}$$

La première condition implique que l'action de G sur l'union disjointe est uniquement déterminée par les actions locales. Une autre conséquence de cette condition est que le plus petit élément permuté par G est supérieur à k, puisque les orbites de G sont ordonnées.

La deuxième condition restreint la structure de G de sorte que l'on ait une bijection entre les sortes et les facteurs.

La troisième condition garantit que les symétries du réseaux N_{i_1} sont isomorphes aux i-ème facteur $K_{(i)}$. Le but d'un produit est de composer des G-réseaux plus "petit", construits à priori.

La quatrième condition empêche un facteur $K_{(j)}$ d'agir sur une sorte O_i. Comme les sortes sont de longueur 1, et que l'image de γ_{i_1} doit être le groupe $K_{(i)}^{\kappa_i}$, les seules symétries qui agiront sur N_{i_1} sont exactement celles définies lors de la construction du G-réseau $(K_{(i)}, N_{i_1}, \kappa_i)$.

Finalement, la cinquième condition limite les synchronisations possibles. Elle interdit les synchronisations locales à un composant (t_1 est unique). De plus, si deux transitions de deux composants i et j aux symétries non triviales se synchronisent, alors les facteurs $K_{(i)}$ et $K_{(j)}$ agiront sur cette transition : le stabilisateur de cette transition ne vérifiera pas les conditions de la Définition 3.12.

L'exemple 3.3.7 illustre une construction par produit disjoint.

Exemple 3.3.7. *Cet exemple montre comment utiliser les produits disjoints pour construire un réseau où le marquage initial est symétrique.*

Dans un premier temps on construit un RCD par une composition à sorte unique à partir du composant N_1, avec $G = S_3$. Le composant N_1 modélise un processus et il est illustré dans la Figure 3.9a :

$$(G, N_1, \kappa_1) = \Pi(1, G, N_1, \emptyset, \emptyset)$$

On renomme les éléments de G pour obtenir $K_{(1)}$ tel que le plus petit point permuté par $K_{(1)}$ soit 3. L'action γ_{i_1} est telle que $\gamma_{i_1}(K_{(1)}) = G^{\kappa_1}$.

Nous effectuons ensuite le produit disjoint de ce RCD avec un composant appelé initiateur et illustré dans la Figure 3.9b. Les éléments de G sont renommés

$$\Pi(2, K_{(1)} \times K_{(2)}, N_1, N_2, \{\{t0.1, start\}\}, \kappa_1, \emptyset)$$

Dans cette construction, $K_{(2)}$ est trivial. Le réseau obtenu est illustré dans la Figure 3.9c

Le Théorème 3.17 montre qu'un produit disjoint de RCD est un RCD. Intuitivement, tous les composants sont des RCD par définition. S'il n'y a pas de synchronisation alors le réseau obtenu peut être vu comme une union disjointe. Si une synchronisation est spécifiée entre deux composants, un seul des deux a des symétries. La transition synchronisée sera permutée comme une transition de ce composant. Comme c'est un RCD, le réseau obtenu est un RCD.

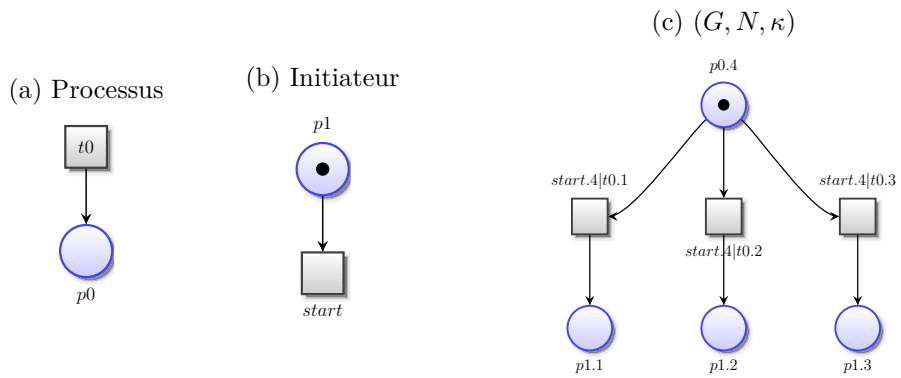


FIGURE 3.9: Exemple de produit disjoint

Théorème 3.17. *Soit* $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ *un produit disjoint tel que défini dans la Définition 3.16. Si pour tout* $i \in 1..k$*, le* G*-réseau* $(K_{(i)}, N_{i_1}, \kappa_i)$ *est un RCD alors* (G, N, κ) *est un RCD.*

Démonstration. 1. Il n'y a pas de synchronisation.

Dans ce cas, $\Psi = \emptyset$. G est un produit facteurs disjoints des $K_{(i)}$ et $\gamma_{i_1}(G) = \gamma_{i_1}(K_{(i)})$ d'où, pour tout $i \in 1..k$, G agit sur N_{i_1} comme $K_{(i)}$ le fait.

Donc comme R_i est un RCD , si les conditions de la Définition 3.12 sont satisfaites pour R_i alors elles le sont pour (G, N, κ).

2. Il y a des synchronisations.

Dans ce cas, pour tout $\psi = (E, I) \in \Psi$, il y a exactement une transition t_i d'un réseau R_i telle que $K_{(i)}$ n'est pas trivial. Soit $x \in P \cup T$ le produit de E.

Par l'équation DEF-G^{μ}_{ψ}, on a $G^{\kappa}_{x} = \nu(G^{\mu}_{\psi})$. Or par Définition du produit disjoint, tous les stabilisateurs des transitions de E différentes de t_i sont triviaux. D'où le stabilisateur de ψ dans M est le stabilisateur de t_i dans $K^{\kappa_i}_{(i)}$ (noté I_{t_i}).

$$G^{\kappa}_{x} = \nu(G^{\mu}_{\psi}) = \nu(I_{t_i})$$

Donc les conditions de la Définition 3.12 sont satisfaites pour R_i alors elles le sont pour (G, N, κ).

□

Produits couronnes disjoints

Les produits couronnes disjoints, ou plus simplement produits couronnes, permettent de modéliser des réseaux dont la structure est arborescente, tel que celui illustré dans la Figure 3.10. Plus généralement, lorsque l'action d'un groupe est imprimitive, alors ce groupe est isomorphe à un sous-groupe d'un produit couronne. Comme il existe des solutions efficaces pour réduire des réseaux avec ce type de symétrie, nous proposons cette construction.

Plus formellement, un produit couronne est un cas particulier de composition symétrique où le groupe G est un produit couronne : $G = K_{(1)} \wr L$. Dans ce cas, L agit transitivement sur le groupe d'automorphismes du produit direct $K_{(1)} \times \ldots \times K_{(l)}$, où pour tout $i, j \in 1..l$ on a $K_{(i)}$ et $K_{(j)}$ sont isomorphes.

Définition 3.18 (Produit couronne disjoint)**.** *Soit le* G*-réseau* $(G, N, \kappa) = \Pi(1, G, N_{1_1}, \Psi, \gamma_{i_1})$ *obtenu par composition symétrique. Alors* (G, N, κ) *est un* produit couronne disjoint *si toutes les conditions suivantes sont vérifiées:*

1. $(G_{(1)}, N_1, \kappa_1)$ *est un* G*-réseau obtenu par composition symétrique ;*

2. $G = K_{(1)} \times \ldots \times K_{(k)} \times L \simeq G_{(1)} \wr L'$ *avec* L' *qui agit transitivement sur* $1..k$;
3. *pour tout* $i \neq j \in 1..k$ *on a* $K_{(i)} \cap K_{(j)} = \emptyset$;
4. *pour tout* $i \in 1..k$, $G_i = K_{(1)} \times \ldots \times K_{(k)} \times L_i$ *et* $\gamma_{i_1}(G_i) = \gamma_{i_1}(K_{(i)}) = G_{(i)}^{\kappa_i}$;
5. *il n'y a pas de synchronisation :* $\Psi = \emptyset$

Les trois premières conditions garantissent que le groupe G est bien un produit couronne des symétries de N_{i_1} par L'.

La quatrième condition implique que le seul facteur de G qui permute la sorte O_1 est L. L'action de L sur la sorte va créer k copies de N_{1_1}, puisqu'il agit transitivement sur $1..k$. Cette condition impose aussi que pour tout $j \in 1..k$ seul $K_{(j)}$ agit sur la copie N_{1_j}. Comme l'image de cette action est égale à l'image de $G_{(j)}$ par κ_j, toutes les copies de N_{1_1} auront des groupes de symétries isomorphes, ce qui est une partie de la définition du produit couronne. L'autre partie de la définition concerne les symétries sur les copies vues comme des blocs, qui sont définies par l'action de L.

La dernière condition restreint les synchronisations. Si une synchronisation est spécifiée, même à l'intérieur d'une copie de N_{i_1} alors à cause de l'action de L, il y aura des éléments dans tous les G^{κ_i} qui permuteront la transition synchronisée et les composants ne seront plus disjoints.

Nous donnons un exemple de construction de produit couronne.

Exemple 3.3.8. *Soit* $(G_{(1)}, N_1, \kappa_1)$ *le G-réseau obtenu par produit disjoint et décrit dans l'exemple 3.3.7. Son groupe de symétries* $G_{(1)}$ *est isomorphe à* S_3.

Nous souhaitons montrer dans cet exemple comment construire un produit couronne à partir de $(G_{(1)}, N_1, \kappa_1)$ *avec* $L' = S_2$. *Tout d'abord, on définit :*

$$G = \langle (3,4), (4,5), (6,7), (7,8), (1,2)(3,6)(4,7)(5,8) \rangle$$

Le sous-groupe $K_{(1)} = \langle (3,4), (4,5) \rangle$ *est isomorphe à* $G_{(1)}$ *tout comme le sous-groupe* $K_{(2)} = \langle (6,7), (7,8) \rangle$. *Le sous-groupe* $L = \langle (1,2)(3,6)(4,7)(5,8) \rangle$ *est isomorphe à* L' *qui agit transitivement sur* $1..2$.

Le stabilisateur de 1 *dans* G *est le sous-groupe* $G_1 = \langle (3,4), (4,5), (6,7), (7,8) \rangle$. *L'action* γ_{a_1} *de* G_1 *sur* $G_{(1)}$ *est définie par :*

$$\gamma_{a_1} = [(3,4), (4,5), (6,7), (7,8)] \to [\kappa_1((1,2)), \kappa_1((2,3)), (), ()]$$

On peut alors définir la composition symétrique ;

$$\Pi(1, 2, G, N_1, \Psi, \gamma_{a_1})$$

Le réseau obtenu est illustré par la Figure 3.10, son groupe de symétries est un produit couronne de $G_{(1)} \wr L'$. *Il y a deux copies de* $(G_{(1)}, N_1, \kappa_1)$, *qui sont permutées par l'action de* L, *et par construction, chacun les noeuds de la première copie de* N_1 *sont permutés par* $K_{(1)} \simeq G_{(1)}$ *et ceux de la deuxième copies par* $K_{(2)} \simeq G_{(2)}$.

Le Théorème 3.19 montre que si N_{1_1} est un RCD, alors le produit couronne est aussi un RCD. Intuitivement, les permutations des copies de N_{1_1} induites par l'action de L préserve la définition des RCD. Les copies sont isomorphes et donc elles sont toutes des RCD. L'absence de synchronisation garantit que les copies resteront des RCD.

Théorème 3.19. *Soit* $(G, N, \kappa) = \Pi(1, G, N_{1_1}, \Psi, \gamma_{i_1})$ *un produit couronne tel que défini dans la Définition 3.18. Si le G-réseau* $(G_{(1)}, N_1, \kappa_1)$ *est un RCD alors* (G, N, κ) *est un RCD.*

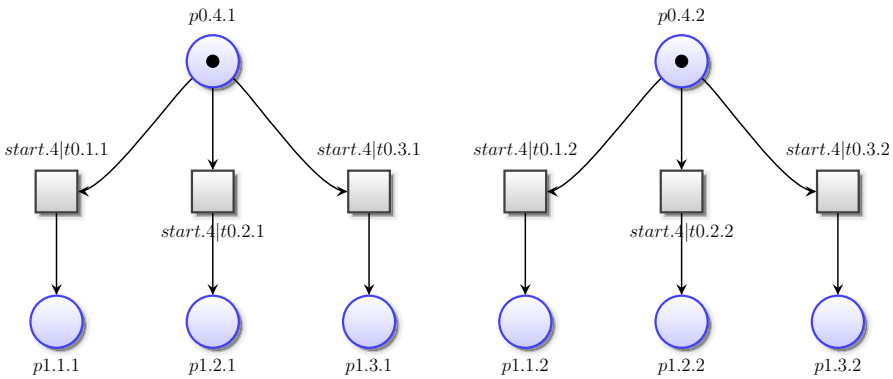


FIGURE 3.10: Produit couronne

Démonstration. On peut faire l'hypothèse, sans perte de généralité que L' n'est pas trivial et donc il n'y a pas d'orbite triviale. Le groupe L définit des isomorphismes entre les copies de N_{i_1}.

Le sous-groupe $D = K_{(1)} \times \ldots \times K_{(k)}$ vérifie les conditions du produit disjoint. D'où le G-réseau $(D, M = \bigcup_{i=1}^{k} N_{i_1}, \delta) = \Pi(k, 1, D, N_{i_1}, \ldots, N_{i_k}, \emptyset, \gamma_{i_1}, \ldots \gamma_{i_k}$ est un RCD et donc, par la Définition 3.12, pour tout $x \in P_M \cup T_M$, il existe une orbite i^D telle que :

$$D_x^\delta = \delta(D_i) \text{ et } (\forall y \in x^{D^\delta})(\exists! j \in i^D)(D_y^\delta = \delta(D_j))$$
$$\Leftrightarrow (\forall l' \in L')(D^\delta_{\lambda(l')x} = \delta(D_{li}) \text{ et } (\forall y \in (\lambda(l')x)^{D^\delta})(\exists! j \in (l'i)^D)(D_y^\delta = \delta(D_j)))$$

et donc (G, N, κ) est un RCD.

□

3.4 Conclusion

Dans ce chapitre nous avons défini les G-réseaux. Un G-réseau est un réseau de Petri équipé d'un groupe de permutations et d'une action de ce groupe sur les symétries du réseau.

Un opérateur de composition symétrique a été défini. Cet opérateur permet de construire des G-réseaux étant donné un groupe de permutation, k G-réseaux , des spécifications de synchronisation et des actions sur les symétries des k G-réseaux .

L'opérateur de composition est très général. Comme il utilise la notion de synchronisation de transition, qu'il ne contraint pas les synchronisations et que G peut être trivial, il permet de construire tous les réseaux qu'il est possible de construire par composition classique.

L'opérateur calcule un *sous-groupe* des automorphismes d'un réseau mais nous ne savons pas si il permet de décrire toutes les symétries du réseau. Un problème analogue est celui de déterminer le groupe des automorphismes d'un graphe de Cayley. Ce problème n'a pas de solution générale connue à ce jour [123].

Notre objectif est d'utiliser les symétries des réseaux pour réduire la taille de leurs espaces d'états. Il s'agit de construire un quotient de l'espace d'états par la relation d'équivalence induite par les symétries. Le problème fondamental à résoudre est le problème de l'orbite. Ce dernier n'a pas de solution efficace dans le cas général.

Nous avons introduit la classe des G-réseaux à composants disjoints et des conditions suffisantes sur la composition permettant de décider à priori si le G-réseau obtenu est un RCD. Cette classe de réseaux donne une condition nécessaire à l'application des solutions

efficaces, tirées de la littératures, au problème de l'orbite. Pour les réseaux de Petri, la condition est suffisante, mais pour les TPN d'autres conditions, objets du Chapitre 4, doivent être vérifiées pour pouvoir réduire efficacement le graphe des classes.

Chapitre 4

Exploitation des symétries des G-réseaux

Sommaire

Ce chapitre présente une méthode d'exploitation des symétries pour la construction d'espace d'états quotients des TPN. La méthode s'applique au graphe des classes.

Soit (G, N, κ) un G-réseau . Le chapitre montre que l'action de G sur N induit des automorphismes sur le graphe d'états de N. Ce graphe étant généralement infini ce résultat n'a pas d'utilité pratique mais est important pour montrer que l'action de G induit aussi des automorphismes du graphe de classes. L'existence de ces automorphismes prouve qu'il est possible de construire un quotient par la relation d'équivalence induite par les symétries.

Le chapitre discute ensuite de l'applicabilité aux G-réseaux des méthodes de réduction développées pour les modèles non temporisés. A cause de la dimension temporelle, la plupart des méthodes pour les modèles non temporisés ne s'appliquent pas (ou moins efficacement) sans une représentation particulière des domaines de tirs.

Parmi les stratégies de réduction possibles, on choisit le calcul du représentant canonique. Ce chapitre présente un ordre total entre les transitions équivalentes par symétrie dans un domaine de tir. Une représentation des états particulière est construite à partir de cet ordre. Cette représentation des états rend possible le calcul efficace du représentant canonique.

Finalement, la méthode de canonisation exploitant l'ordre total est appliquée aux RCD. Les RCD capturent les conditions suffisantes pour canoniser un état global en canonisant un vecteur d'états locaux. Le calcul se fait en temps polynomial pour les groupes simples, comme dans le cas des modèles non temporisés.

Les principaux résultats de ce chapitre ont fait l'objet d'une publication lors de la 30-ième édition du symposium ACM "Symposium on Applied Computing" sous le titre :

Symmetry reduced state classes for time petri nets [2]

4.1 Symétries du graphe des classes d'états

Soit le G-réseau (G, N, κ) obtenu par composition symétrique. Le groupe G^κ des symétries de N induit des automorphismes du graphe d'états de N. Ces automorphismes s'obtiennent par l'action de G^κ sur les marquages et les intervalles dynamiques.

Les symétries du graphe d'états n'ont pas d'utilité pratique car ce graphe est généralement infini. Nous montrons alors que les symétries de N induisent aussi des automorphismes sur le graphe des classes. Les automorphismes s'obtiennent par l'action de G^κ sur les marquages et les domaines de tirs des classes. La relation d'orbite permet de construire un quotient du graphe des classes. Ce quotient préserve l'accessibilité modulo la relation de symétrie lorsque le marquage initial est invariant par rapport à G^κ : une classe C est dans le graphe des classes si et seulement si il existe une classe de son orbite dans le graphe quotient. Ce résultat étend le Théorème 2.5 énoncé dans le contexte des réseaux de Petri aux réseaux de Petri temporels.

4.1.1 Symétries du graphe d'états

Les symétries d'un graphe sont définies comme des permutations de ses sommets qui préservent les arêtes. Dans le graphe d'états du réseau N, les sommets sont des états (m, I) où m est un marquage et I est une fonction partielle, appelée intervalle dynamique, qui associe un intervalle dans $\mathbb{R}_{\geq}$ à chacune des transitions sensibilisées par m. Il faut donc définir l'action de G^κ sur ces couples. Nous montrons ensuite que ces permutations induisent des symétries du graphe.

L'action de G^κ sur les marquages de N est définie comme pour les réseaux non temporisés : l'image du marquage d'une place par un élément g de G^κ est égale au marquage de la préimage par g de cette place. L'action sur les intervalles dynamiques est défini de manière similaire : l'image par g de l'intervalle d'une transition est égale à l'intervalle de la préimage par g de cette transition.

Soit $SG = \langle S, s_0, \to \rangle$ le graphe d'états, avec S l'ensemble des sommets, s_0 l'état initial et $\to$ l'ensemble des arêtes. Alors l'action ρ de G^κ sur S est définie, pour tout $g \in G^\kappa$ et tout $(m, I) \in S$, par :

$$g \cdot (m, I) = (g \cdot m, g \cdot I) \tag{G-SG}$$

avec :

1. $\forall p \in P, g \cdot m(p) = m(g^{-1}p)$
2. $\forall t \in T, g \cdot I(t) = I(g^{-1}t)$

Pour tout $g_1, g_2 \in G^\kappa$, $g_1 g_2 \cdot m(p) = m((g_1 g_2)^{-1} p) = m(g_2^{-1} g_1^{-1} p) = g_2 \cdot m(g_1^{-1} p) = g_1 \cdot g_2 \cdot m(p)$. De plus, $g_1 g_2 \cdot I(t) = I((g_1 g_2)^{-1} t) = I(g_2^{-1} g_1^{-1} t) = g_2 \cdot I(g_1^{-1} t) = g_1 \cdot g_2 \cdot I(t)$. Nous pouvons en déduire que $g_1 g_2 \cdot (m, I) = g_1 \cdot g_2 \cdot (m, I)$ et conclure que ρ est bien une action.

Le Lemme 2.2 montre que ρ induit des automorphismes du graphe d'états. Du point de vue de la sémantique de N, cela signifie que :

1. si une transition t est sensibilisée ou tirable depuis un marquage m, alors son image l'est aussi dans l'image de m ; et
2. les écoulements de temps possibles depuis un état (m, I) le sont aussi depuis l'image de (m, I).

Lemme 4.1. *Soit le G-réseau* (G, N, κ) *et* $SG = \langle S, s_0, \to \rangle$ *le graphe d'états de* N*. Soit* $\rho : G^\kappa \to Sym(S)$ *l'action de* G^κ *sur les états de SG définie par G-SG.*

Alors pour tout $g \in G^\kappa$*,* $t \in T$*,* $I \in \mathbb{I}$*,* $I' \in \mathbb{I}$*,* $\theta \in \mathbb{R}_\geq$ *et tout marquage* m *et* m' *de* N*, on a :*

(i) $(m, I) \xrightarrow{t} (m', I') \Leftrightarrow g(m, I) \xrightarrow{gt} g(m', I')$

(ii) $(m, I) \xrightarrow{\theta} (m', I') \Leftrightarrow g(m, I) \xrightarrow{\theta} g(m', I')$

Démonstration. La preuve consiste à montrer que les conditions de la Définition 1.11 des graphes d'états sont préservées par l'action de G^κ.

Pour tout $p \in P$:

(i) $(m, I) \xrightarrow{t} (m', I') \Leftrightarrow g(m, I) \xrightarrow{gt} g(m', I')$ si et seulement si:

1. t est sensibilisée dans m si et seulement si gt l'est dans gm.
   Nous vérifions $m(p) \geq \mathbf{Pre}(t)(p) \Leftrightarrow gm(gp) \geq \mathbf{Pre}(gt)(gp)$.
   Par définition de ρ, on a pour tout $p \in P$: $gm(p) = m(g^{-1}p) \Leftrightarrow gm(gp) = m(g^{-1}gp)) = m(p)$ et par définition des symétries d'un réseau, on a $\mathbf{Pre}(t)(p) = \mathbf{Pre}(gt)(gp)$, d'où :

$$gm(gp) \geq \mathbf{Pre}(gt)(gp) \Leftrightarrow m(p) = \mathbf{Pre}(t)(p)$$

2. Le marquage obtenu depuis m par le tir de t est l'image de celui obtenu depuis gm par le tir de gt.
   Nous vérifions $m'(p) = m(p) - \mathbf{Pre}(t)(p) + \mathbf{Post}(t)(p) \Leftrightarrow gm'(gp) = gm(gp) - \mathbf{Pre}(gt)(gp) + \mathbf{Post}(gt)(gp)$.
   On a, par les même arguments que le point 1. :

$$\begin{aligned} gm'(gp) &= gm(gp) - \mathbf{Pre}(gt)(gp) + \mathbf{Post}(gt)(gp) \\ &= m(p) - \mathbf{Pre}(t)(p) + \mathbf{Post}(t)(p) \\ &= m'(p) \end{aligned}$$

3. t est tirable ssi gt est tirable .
   Nous vérifions $0 \in I(t) \Leftrightarrow 0 \in gI(gt)$.
   Par définition de ρ, on a $gI(t) = I(g^{-1}t) \Leftrightarrow gI(gt) = I(g^{-1}gt) = I(t)$. D'où $0 \in I(t) \Leftrightarrow 0 \in gI(gt)$.
4. L'intervalle de tir d'une transition persistante $k \neq t$ est invariant par le tir de t.
   Soit le prédicat $A(k, t, p)$ définit par $A(k, t, p) \Leftrightarrow k \neq t \wedge m(p) - \mathbf{Pre}(t)(p) \geq \mathbf{Pre}(k)(p)$. Ce prédicat est vrai si la transition k est persistante. Alors :

$$\begin{aligned} &(\forall k)(m'(p) \geq \mathbf{Pre}(k)(p) \Rightarrow (A(k,t,p) \wedge I'(k) = I(k)) \vee (I'(k) = I_s(k))) \\ \Leftrightarrow\ &(\forall k)(gm'(gp) \geq \mathbf{Pre}(gk)(gp) \Rightarrow (A(gk, gt, gp) \wedge gI'(gk) = gI(gk)) \\ &\qquad\qquad \vee (gI'(gk) = g(I_s)(gk))) \end{aligned}$$

   Nous avons vu dans le point 1. que, pour tout k, $m'(p)(k) \geq \mathbf{Pre}(k)(p) \Leftrightarrow gm'(gp) \geq \mathbf{Pre}(gk)(gp)$. De plus, nous avons $A(k, t, p) \Leftrightarrow A(gk, gt, gp)$. Les intervalles dynamiques (point 2.) et statiques (définition des symétries) sont préservés : $I(k) = gI(gk)$ et $I_s(k) = gI_s(gk)$. D'où le résultat.

(ii) $(m, I) \xrightarrow{\theta} (m, I')$ si et seulement si:

5. Les écoulements de temps possibles depuis m le sont depuis gm.

$$(\forall k)(m(p) \geq \mathbf{Pre}(k)(p) \Rightarrow \theta \leq \uparrow I(k) \wedge I'(k) = I(k) \dot{-} \theta)$$
$$\Leftrightarrow (\forall k)(gm(p) \geq \mathbf{Pre}(gk)(gp) \Rightarrow \theta \leq \uparrow gI(gk) \wedge gI'(gk) = gI(gk) \dot{-} \theta)$$

Nous avons vu dans le point 1. que, pour tout k, $m'(p)(k) \geq \mathbf{Pre}(k)(p) \Leftrightarrow gm'(gp) \geq \mathbf{Pre}(gk)(gp)$. De plus, puisque $I(k) = gI(k)$, $gI'(gk) = gI(gk) \dot{-} \theta = I(k) \dot{-} \theta = I'(k)$. D'où le résultat.

□

4.1.2 Symétries du graphe des classes d'états

Le graphe des classe d'états est une abstraction finie du graphe d'états. Dans cette abstraction, une classe contient tous les états atteignables par une séquence de tir de transitions, quelque soit leurs dates de tir. Dans cette thèse, nous ne traitons que de la construction classique des classes d'états, telle que décrite par l'Algorithme 1.2.1.

Nous avons vu dans la section précédente que si une transition est tirable depuis un état du graphe d'états alors son image l'est aussi depuis l'image de cet état. De même, les écoulements de temps possibles depuis un état le sont aussi depuis son image. On peut avoir l'intuition que ces propriétés s'étendent aux graphes des classes, où les intervalles dynamiques sont remplacés par des domaines de tirs qui capturent l'ensemble des dates de tir possibles.

Dans cette section nous montrons que les symétries d'un réseau induisent des automorphismes du graphe des classes de ce réseau. Ces automorphismes nous permettent de construire un quotient du graphe des classes. De plus, nous montrons que si l'état initial du graphe des classes est symétrique, alors une classe existe dans le graphe des classes si et seulement si il existe une image de cette classe dans le quotient et ce dernier préserve l'accessibilité modulo les symétries.

Soit le G-réseau (G, N, κ), SG son graphe d'états et $SCG = \langle \mathcal{C}, \rightarrow \subseteq \mathcal{C} \times T \times \mathcal{C} \rangle$ le graphe des classes d'états de N, où $\mathcal{C}$ est l'ensemble des classes.

L'action de G^κ sur les marquages est définie comme pour les graphes d'états. Nous définissons maintenant l'action de G^κ sur les domaines de tirs. Soit (m, D) une classe du SCG et D en forme canonique. Le domaine de tir D est l'ensemble des solutions d'un système fini d'inéquations linéaires où chaque variable est associée à exactement une transition sensibilisée par m. Nous identifierons les variables et les transitions. L'image de D par un élément g de G^κ est l'ensemble des solutions du système d'inéquations où chaque transition est remplacée par son image par g.

Plus formellement, soit L un système fini d'inéquations linéaires de la forme

$$A \times \underline{\phi} \leq \underline{w}$$

où A est une matrice sur $\{-1, 0, 1\}$, $\underline{\phi}$ un vecteur avec ϕ_k la k-ème transition sensibilisée et $\underline{w}$ un vecteur d'entiers. Nous notons $\mathcal{F}(L)$ l'ensemble des solutions de L et $D = \mathcal{F}(L)$. L'action de G^κ est sur les domaines de tir est définie, pour tout $g \in G^\kappa$, par :

$$g \cdot (\mathcal{F}(A \times \underline{\phi} \leq \underline{w})) = \mathcal{F}(A \times (g^{-1} \cdot \underline{\phi}) \leq \underline{w}) \tag{GD}$$

où g agit sur $\underline{\phi}$ par permutation de ses composantes. Pour tout $g_1, g_2 \in G^\kappa$, nous avons :

$$\begin{aligned} g_1 g_2 \cdot \mathcal{F}(A \times \underline{\phi} \leq \underline{w}) &= \mathcal{F}(A \times ((g_1 g_2)^{-1} \cdot \underline{\phi}) \leq \underline{w}) \\ &= \mathcal{F}(A \times (g_2^{-1} g_1^{-1} \cdot \underline{\phi}) \leq \underline{w}) \\ &= g_2 \cdot \mathcal{F}(A \times (g_1^{-1} \cdot \underline{\phi}) \leq \underline{w}) \\ &= g_1 \cdot g_2 \cdot \mathcal{F}(A \times \underline{\phi} \leq \underline{w}) \end{aligned}$$

D'où nous pouvons déduire que GD définit effectivement une action.

Afin de simplifier la notation, nous noterons gD pour $g \cdot \mathcal{F}(L)$. Nous en déduisons l'action $\tau : G^\kappa \to Sym(\mathcal{C})$ de G^κ sur $\mathcal{C}$, définie pour tout $g \in G^\kappa$ et $(m, D) \in \mathcal{C}$ par :

1. $\forall p \in P$, $g \cdot m(p) = m(g^{-1}p)$;
2. $g \cdot D$ est définie par GD

Le Théorème 4.2 montre que l'action de G^κ sur le SCG définit un groupe d'automorphismes : si une transition t est tirée depuis une classe (m, D) et produit la classe (m', D') alors l'image gt de t par g est tirée depuis $g(m, D)$ et produit la classe $g(m', D')$.

Théorème 4.2. *Soit le G-réseau (G, N, κ) et $SCG = \langle \mathcal{C}, \to \subseteq \mathcal{C} \times T \times \mathcal{C} \rangle$ le graphe des classes d'états obtenu par l'Algorithme 1.2.1. Soit $\tau : G^\kappa \to Sym(\mathcal{C})$ l'action de G^κ sur $\mathcal{C}$ définie par GD.*

Alors pour tout $t \in T$, (m, D) et (m', D') deux classes du SCG, on a :

$$(m, D) \xrightarrow{t} (m', D') \Leftrightarrow g(m, D) \xrightarrow{gt} g(m', D')$$

Démonstration. Soit $\mathcal{E}$ l'ensemble des transitions sensibilisées par un marquage. Pour m, on a $\mathcal{E}(m) = \{t \in T \mid (\forall p \in P)(m(p) \geq \mathbf{Pre}(m)(p))\}$.

Nous montrons que $(m, D) \xrightarrow{t} (m', D') \Leftrightarrow g(m, D) \xrightarrow{gt} g(m', D')$ si et seulement si :

1. t est tirable depuis (m, D) si et seulement si gt est tirable depuis $g(m, D)$
   (a) t est sensibilisée dans m ssi gt est sensibilisée dans gm
   Par le Lemme 2.2 une transition est sensibilisée par m ssi son image par g l'est dans gm :
   $$(\forall k \in T)(k \in \mathcal{E}(m) \Leftrightarrow gk \in \mathcal{E}(gm))$$
   d'où t est sensibilisée dans m ssi gt l'est dans gm.
   (b) Soit F les contraintes sur les dates de tirs au plus tard de t : $F = \{\underline{t_t} \leq \underline{t_i} \mid t \neq i \wedge i \in \mathcal{E}(m)\}$. Le système $D \wedge F$ est satisfiable ssi le système $gD \wedge gF$ l'est aussi.
   Par définition de τ, nous avons
   $$gF = \{g\underline{t_t} \leq g\underline{t_i} \mid gi \neq gt \wedge gi \in \mathcal{E}(gm)\}$$
   D'où $D \wedge F$ admet une solution ssi $gD \wedge gF$ en admet une.
2. La classe obtenue en tirant t depuis (m, D) est (m', D') si et seulement si la classe obtenue en tirant gt depuis $g(m, D)$ est $g(m', D')$.
   Pour tout k,
   (a) k est nouvellement sensibilisée dans m' ssi gk l'est dans gm.
   Si $k \in \mathcal{E}(m')$ alors pour tout $p \in P$, $m'(p) \leq \mathbf{Pre}(t)(p)$. Comme g est une symétrie et que $gm'(gp) = m'(p)$ on peut déduire, pour tout p :
   $$(\forall k)((k \in \mathcal{E}(m') \Leftrightarrow gk \in \mathcal{E}(gm')) \wedge D' \Leftrightarrow gD')$$
   (b) k est persistante dans m' ssi gk l'est dans gm'.
   Si $k \in \mathcal{E}(m) \cap \mathcal{E}(m')$ alors pour tout $p \in P$, $m'(p) \leq m(p) - \mathbf{Pre}(t)(p) + \mathbf{Post}(t)(p)$. Comme g est une symétrie et que $gm'(gp) = m'(p)$ on peut déduire, pour tout p :
   $$m'(p) = m(p) - \mathbf{Pre}(t)(p) + \mathbf{Post}(t)(p) \Leftrightarrow gm'(p) = gm(p) - \mathbf{Pre}(gt)(p) + \mathbf{Post}(gt)(p)$$
   (c) L'image de D' par g est la classe obtenue en tirant gt depuis gD.
   Pour tout $k \in \mathcal{E}(m')$

Si k est nouvellement sensisibilisée alors gk l'est aussi dans gm' et par définition des symétries $I_s(k') = I_s(gk')$.

Si k est persistante, par définition de τ, on a $\phi'_k = \phi_k - \phi_t \Leftrightarrow g\phi'_k = g\phi_k - g\phi_t$. Soit $\Delta = D \cap F$ le système d'inéquations linéaires obtenu à l'étape *2.* de l'Algorithme 1.2.1. Par 1.(b). Alors $g\Delta = gD \cap gF$ est obtenu par permutations des variables et admet les même solutions. Pour chaque variable ϕ_k éliminée dans Δ, nous éliminons $g\phi_k$ dans $g\Delta$. D'où, pour chaque ϕ'_k dans D', il existe exactement une variable $g\phi'_k$ dans gD'. Et donc l'image de D' par g est bien le domaine obtenu en tirant gt depuis $g(m, D)$.

□

Graphe quotient

Dans le cas général, le quotient d'un graphe par une relation d'équivalence est le graphe induit sur l'ensemble quotient des sommets. En pratique, on utilise un représentant à la place d'une classe d'équivalence. Le graphe quotient est obtenu comme le sous graphe induit en ne conservant qu'un sommet par classe d'équivalence. Nous définissons cette construction dans notre contexte.

La relation d'équivalence utilisée est la relation d'orbite induite par l'action de G^κ sur le SCG. Chaque orbite est une classe d'équivalence sur $\mathcal{C}$. Pour $C_1, C_2 \in \mathcal{C}$, nous définissons la relation $\approx$ par :

$$C_1 \approx C_2 \Leftrightarrow (\exists g \in G^\kappa)(C_2 = gC_1) \quad \text{(EC)}$$

Nous dirons alors que C_1 et C_2 sont G-équivalents. La construction du graphe des classes présentée dans l'Algorithme 1.2.1 est adaptée afin d'obtenir le quotient du SCG, noté $SCG_\approx$, en insérant dans le SCG non plus la classe obtenue par le tir d'une transition mais un représentant de cette classe.

Soit $\theta : \mathcal{C} \to \mathcal{C}$ la fonction qui à chaque classe $(m, D) \in \mathcal{C}$ associe un représentant de son orbite. Nous ne précisons pas pour l'instant si ce représentant est unique et proposerons dans la Section 4.3.3 une spécification précise de θ. Le graphe des classes quotient, noté $SCG_\approx$, est construit en ne conservant qu'une seule classe par orbite. Étant donnée une classe C, telle que t est tirable, l'algorithme calcule la classe obtenue par le tir de t et insère la classe $\theta(C)$ dans le SCG. Comme $\theta(C)$ est généralement différent de C, la construction ne préserve pas les traces mais préserve l'accessibilité.

Algorithme 4.1.1 Calcul du quotient $SCG_{\approx}$

Le SCG est le plus petit ensemble de classes $\mathcal{C}$ contenant L_ϵ et tel que, lorsque $L_\sigma \in \mathcal{C}$ et $\sigma.t$ est tirable, alors $(\exists X \in \mathcal{C})(X \cong \theta(L_{\sigma.t}))$, où $\theta(L_{\sigma.t})$ est le représentant de l'orbite de $L_{\sigma.t}$.

1. La classe initiale est $L_\epsilon = (m_0, D_0)$, où D_0 est le domaine défini par le système d'inégalités $\{\downarrow I_s(t) \leq \phi_t \leq \uparrow I_s(t) \mid t \in \mathcal{E}(m_0)\}$.
2. Si σ est tirable et $L_\sigma = (m, D)$, alors $\sigma.t$ est tirable si et seulement si:
   (a) $m \geq \mathbf{Pre}(t)$ (t est sensibilisée m)
   (b) Le système $D \wedge F$ est satisfiable, avec $F = \{\phi_t \leq \phi_i \mid i \neq t \wedge i \in \mathcal{E}(m)\}$
3. Si $\sigma.t$ est tirable et que $L_\sigma = (m, D)$ alors $L_{\sigma.t} = (m', D')$ avec :
   $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
   D' est construit à partir de D en trois étapes :
   (a) Les inégalités de F sont ajoutées à D ;
   (b) Pour chaque k sensibilisée dans m' on introduit une nouvelle variable ϕ'_k telle que: $\begin{array}{ll} \phi'_k = \phi_k - \phi_t & \text{si } k \text{ est persistente} \\ \downarrow I_s(k) \leq \phi'_k \leq \uparrow I_s(k) & \text{si } k \text{ est nouvellement sensibilisée} \end{array}$
   (c) Les (anciennes) variables ϕ_i sont éliminées.

Le Théorème 4.3 montre que la construction du graphe quotient préserve l'accessibilité modulo les symétries.

Théorème 4.3. *Soit le G-réseau (G, N, κ), $SCG = \langle \mathcal{C}, \rightarrow \subseteq \mathcal{C} \times T \times \mathcal{C} \rangle$ le graphe des classes d'états obtenu par l'Algorithme 1.2.1 et $SCG_{\approx} = \langle \mathcal{C}_{\approx}, \rightarrow \subset \mathcal{C}_{\approx} \times T \times \mathcal{C}_{\approx} \rangle$ son quotient obtenu par l'Algorithme 4.1.1. Soit $\tau : G^\kappa \rightarrow Sym(\mathcal{C})$ l'action de G^κ sur $\mathcal{C}$ définie par GD et $\approx$ la relation définie par EC.*

Alors si m_0 est symétrique il existe pour toute classe $C \in \mathcal{C}$ une classe C^* dans $SCG_{\approx}$ telle que $\mathcal{C}^* \approx \mathcal{C}$:

$$C \in \mathcal{C} \Leftrightarrow (\exists C^* \in \mathcal{C}_{\approx})(C \approx C^*)$$

Démonstration. $\Rightarrow$ Si $C \in \mathcal{C}$ alors il existe $C^* \in \mathcal{C}_{\approx}$ telle que $C \approx C^*$.

Par induction sur les séquences de tir :

1. Soit $C_0 = (m_0, D_0)$. Par hypothèse, pour tout $g \in G^\kappa$, nous avons $gC_0 = C_0$. D'où
$$C_0 \in \mathcal{C} \Rightarrow gc_0 \in \mathcal{C}_{\approx}$$
2. Soit $C_\sigma \in SCG$ et $g \in G^\kappa$ tels que $gC_\sigma \in C_{\approx}$. Soit t une transition tirable depuis C_σ et $C_\sigma \xrightarrow{t} C_{\sigma.t}$.
   Par le Théorème 4.2, nous avons $gC_\sigma \xrightarrow{gt} gC_{\sigma.t}$ et nous pouvons déduire :
$$C_{\sigma.t} \in \mathcal{C} \Rightarrow gC_{\sigma.t} \in \mathcal{C}_{\approx}$$

$\Leftarrow$ Si $C^* \in \mathcal{C}_{\approx}$ alors il existe $C \in \mathcal{C}$ telle que $C \approx C^*$.

Par induction sur les séquences de tirs :

1. Soit $C_0^* \in \mathcal{C}_{\approx}$ et $g \in G^\kappa$, alors par hypothèse on a $gC_0^* = C_0^* = C_0 \in \mathcal{C}$.
2. Soit $C_\sigma^* \in \mathcal{C}_{\approx}$ telle que $gC_\sigma^* \in \mathcal{C}$. Soit t une transition tirable depuis C_σ^* et $C_\sigma^* \xrightarrow{t} C_{\sigma.t}^*$.
   Par le Théorème 4.2, nous avons $gC_\sigma^* \xrightarrow{gt} gC_{\sigma.t}^*$ et nous pouvons déduire :
$$C_{\sigma.t}^* \in \mathcal{C}_{\approx} \Rightarrow gC_{\sigma.t}^* \in \mathcal{C}$$

□

4.2 Réduction des G-réseaux

Cette section discute de l'applicabilité aux G-réseaux des méthodes de réduction présentées dans le Chapitre 2.

Soit $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ un G-réseau obtenu par composition symétrique, SCG son graphe des classes d'états et (m, D) une classe. L'ensemble P des places de N est équipé d'une ordre total $\leq_P$. De même l'ordre $\leq_T$ ordonne totalement les transitions de N. On suppose que les domaines de tirs sont en forme canonique et implémentés par des DBM indexées par les transitions sensibilisées dans m.

Toutes les méthodes de réduction nécessitent de pouvoir comparer les classes. Nous définissons donc un ordre $\leq_N$ à partir de $\leq_P$ et $\leq_T$. On ordonne les places suivant $\leq_N$:

$$x \leq_N y \stackrel{\text{déf}}{=} \begin{cases} x, y \in P & \text{et} \quad x \leq_P y \\ x, y \in T & \text{et} \quad x \leq_T y \\ x \in P & \text{et} \quad y \in T \end{cases} \qquad \text{(DEF-}\leq_N\text{)}$$

Le Lemme 4.4 montre que l'ordre $\leq_N$ est total.

Lemme 4.4. *L'ordre $\leq_N$ défini par DEF-$\leq_N$ est total*

Démonstration. Les ordres $\leq_P$ et $\leq_T$ sont totaux et un noeud est soit une place soit une transition. □

Il est aussi nécessaire de pouvoir comparer lexicographiquement deux domaines de tir. Pour ce faire, nous avons besoin de deux ordres : un sur les indices et l'autre sur les constantes. Comme les constantes sont des entiers, nous définissons uniquement l'ordre sur les indices. Nous appellerons cet ordre *un ordre linéarisant*, noté $\preceq_L$, et défini, avec γ_{t_1,t_2} et γ_{t_3,t_4} deux indices d'une même domaine de tir, par :

$$\gamma_{t_1,t_2} \preceq_L \gamma_{t_3,t_4} \stackrel{\text{déf}}{=} \begin{cases} t_1 = t_3 \wedge t_2 = t_4 \\ t_1 = t_3 \wedge t_2 \leq_T t_4 \\ t_1 \leq_T t_3 \end{cases} \qquad \text{(DEF-}\preceq_L\text{)}$$

On peut déduire de $\preceq_L$ un ordre lexicographique sur les classes, noté $\leq_L$ qui compare lexicographiquement d'abord les marquages, puis les domaines de tir en utilisant $\preceq_L$ pour ordonner les indices des contraintes de différences.

Comme discuté dans la Section 2.3, trois solutions sont possibles pour le problème de l'intégration :

1. Itérer sur les symétries de G
2. Itérer sur les états de M
3. Calculer, à partir de m_1, un représentant canonique de sa classe d'équivalence.

4.2.1 Méthodes par itération

Les deux premières approches, illustrées par les Algorithmes 2.3.1 et 2.3.2 utilisent le test d'égalité entre classes. Comme les domaines de tirs sont en forme canonique, il est toujours possible de tester l'égalité de deux classes en comparant les états par $\leq_L$.

On en déduit que les méthodes par itération peuvent s'appliquer en l'état sur les G-réseaux . Plus généralement, elles sont applicables quelque soit la représentation des états pourvu qu'on puisse tester l'égalité.

Néanmoins ces méthodes sont limitées par l'ordre du groupe ou le nombre d'états, ce qui les rend inapplicables dès lors que l'ordre du groupe ou l'espace d'états sont grands. L'optimisation basée sur les fonctions de hachages symétriques pourraient s'appliquer aux G-réseaux . Une fonction de hachage compatible avec les symétries peut être implémentée en calculant le déterminant d'un domaine de tir. En effet, si D_1 et D_2 sont deux domaines de tir tel qu'il existe une permutation g des lignes et colonnes de D_1 telle que $gD_1 = D_2$ alors D_1 et D_2 ont le même déterminant. Mais si le réseau est très symétrique les orbites seront grandes et l'efficacité de l'optimisation n'est pas garantie.

Si les méthodes par itération peuvent s'appliquer sur les G-réseaux , il est clair qu'elles sont limitées aux cas où les groupes ou les espaces d'états sont petits. Or les gains de réduction sont proportionnels à la taille du groupe. Il est donc nécessaire de proposer une solution de réduction efficace pour les grands groupes et en particulier S_n. Pour cette raison, nous avons choisi d'implémenter la solution du calcul du représentant canonique.

4.2.2 Représentant canonique

Le calcul du représentant canonique consiste à calculer l'élément le plus petit de son orbite. Cela implique de disposer d'un ordre total sur les états des orbites.

Dans cette section, nous montrons que l'ordre linéarisant sur les DBM n'est pas suffisant pour calculer le représentant canonique d'une classe, ce qui justifie l'ordre total sur les classes équivalentes proposées dans le reste de ce chapitre.

Symétrie totale

Notre méthode de calcul des représentants canoniques repose sur le tri lexicographique du vecteur des états locaux des composants. Puisque le groupe de symétries est S_n, nous pouvons utiliser l'ensemble des transpositions sur $1..n$ et utiliser un tri à bulles (cf ensembles minorants de [108]).

Cette implémentation fait l'hypothèse que les états locaux des composants sont disjoints. Lorsque le G-réseau est un RCD, il est possible de partitionner les places et transitions du réseau avec une partie par composants. Mais lorsqu'on chercher à canoniser une classe, il faut pouvoir partitionner l'état global en états locaux. Dans le cas non temporisé, il y a une bijection entre les places et le marquage : l'état est un vecteur du marquage des places. Cette propriété n'est plus vraie dans le cas temporisé puisque les contraintes de différences d'une DBM peuvent être associées à deux transitions appartenant à des composants différents. On ne peut donc pas partitionner l'état global en état locaux de composants lorsque la DBM est linéarisée par $\leq_L$. Cela n'empêche pas d'appliquer le tri à bulles, mais le représentant obtenu n'est pas pas nécessairement canonique.

Nous proposons un ordre total sur les classes d'états équivalentes qui permet d'obtenir un représentant canonique.

Recherche avec retours arrières

La méthode de réduction proposée par [124] vise à construire un quotient pour des réseaux symétriques quelconques. Cette méthode permet de lever les restrictions pour le calcul du représentant canonique.

L'approche repose sur une recherche avec retour arrière et le choix d'une BSGS de places et transitions. La méthode s'applique au cas non temporisé grâce à la bijection entre les places et les indices du vecteur des marquages. Cette propriété permet d'utiliser une base de places pour canoniser des états. Mais cette propriété n'est plus vraie dans le cas temporisé (à cause des contraintes de différence).

Dans le cas temporisé, il faudrait calculer une base pour chaque nouvelle classe et nous conjecturons qu'il y aurait plus d'éléments du groupe à explorer que dans le cas non temporisé. Cette conjecture repose sur l'observation suivante. Une base est une séquence de points stabilisée par l'identité. Dans le cas où le groupe est S_3, on peut choisir la base $[1, 2]$ et une forme canonique pour un état d'un réseau non temporisé peut s'obtenir en énumérant 4 éléments de S_3 . Supposons un G-réseau sans place et avec trois transitions équivalentes par symétrie, une par composant, sur lequel agit S_3. Ces transitions sont toujours sensibilisées. La DBM contient 6 contraintes de différences (en négligeant les contraintes avec la transition spéciale 0). L'état global est une DBM linéarisée par $\leq_L$. Le stabilisateur d'une contrainte $\gamma_{i,j}$ dans l'action de S_3 sur l'état global est trivial. Dans ce cas, la base la plus longue que l'on peut construire pour l'état global est de longueur 1. Il faudra donc énumérer les 6 éléments de S_3.

L'approche de réduction par recherche arrière n'a pas été explorée dans cette thèse.

4.3 Réduction des RCD

4.3.1 Un ordre sur les transitions de même intervalle statique

Soit (G, N, κ) un G-réseau , $SCG = \langle \mathcal{C}, \rightarrow \subseteq \mathcal{C} \times T \times \mathcal{C} \rangle$ le graphe des classes, (m, D) une classe du SCG. Tous les domaines de tirs D sont en forme canonique. Soit deux transitions i et j de T de même intervalle statique. Nous dirons que i et j sont I_s-équivalentes. Dans le reste de ce chapitre, nous réutilisons les notations introduites dans la Section 1.2.3 pour les contraintes de différences entre les transitions.

Il est possible de comparer deux transitions I_s-équivalentes i et j en comparant un sous-ensemble des inéquations où i et j interviennent. Plus précisément, il suffit de comparer $\gamma_{i,j}$ avec $\gamma_{j,i}$ et $\gamma_{i,k}$ avec $\gamma_{j,k}$, pour tout k différent de i, j. La relation $\preceq_D$ est définie, pour tout i, j I_s-équivalentes et sensibilisées dans m, par :

$$i \preceq_D j \quad =_{\text{def}} \quad \gamma_{i,j} \leq \gamma_{j,i} \ \wedge \ (\forall k \neq i, j)(\gamma_{i,k} \leq \gamma_{j,k}) \qquad \text{(DEF-}\preceq_D\text{)}$$

Lorsque D est implémentée par une DBM, comparer i et j par $\preceq_D$ revient à comparer uniquement les lignes de la DBM et en déduire les relations sur les colonnes : nous pouvons ainsi comparer deux transitions en temps linéaire.

Le Lemme 4.5 prouve un invariant sur les dates de tirs minimales et maximales de i et j : si $i \preceq_D j$ alors la date de tir maximale de i est inférieure à celle de j et la date de tir minimale de i est infèrieure à celle de j. La Figure 4.1 illustre le Lemme : les constantes à comparer sont représentées par des croix rouges et les comparaisons déduites par des cercles pleins bleus.

Lemme 4.5. *Soit $\preceq_D$ la relation définie par DEF-$\preceq_D$ et i, j deux transitions I_s-équivalentes sensibilisées par m.*

Si $i \preceq_D j$ alors on a $\alpha_i \leq \alpha_j$, $\beta_i \leq \beta_j$ et pour toute transition k différente de i, j, $\gamma_{k,j} \leq \gamma_{k,i}$:

$$i \preceq_D j \Rightarrow \alpha_i \leq \alpha_j \wedge (\forall k)(\gamma_{k,i} \geq \gamma_{k,j})$$

Démonstration. Soit $C = (m, D)$ une classe obtenue par construction du SCG et deux transitions i, j de même intervalle statique ($I_s(i) = I_s(j)$). Nous montrons que si $i \preceq_D j$ alors

(i) $\alpha_i \leq \alpha_j$;

(ii) $\beta_i \leq \beta_j$; et

(iii) $(\forall k \neq i, j)(k \in \mathcal{E} \Rightarrow \gamma_{k,j} \leq \gamma_{k,i})$.

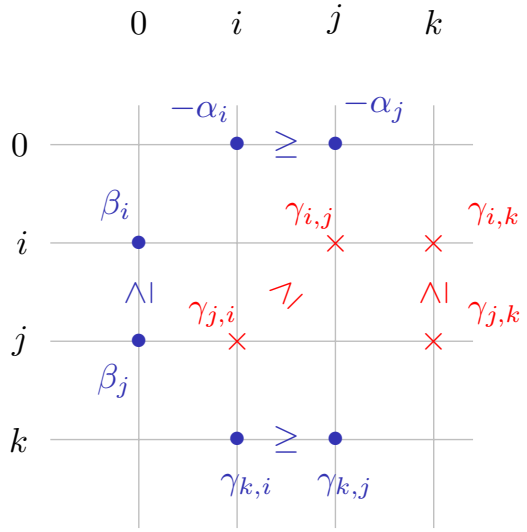


FIGURE 4.1: Illustration du Lemme 4.5

Dans le reste de la preuve, le point (iii) est noté F.

La preuve est une analyse des cas pour i, j.

Si i et j sont nouvellement sensibilisées:

$\alpha_i = \alpha_j$ puisque, par hypothèse, $\alpha_i = \alpha_i^s$, $\alpha_j = \alpha_j^s$ et $\alpha_i^s = \alpha_j^s$

$\beta_i = \beta_j$ puisque, par hypothèse, $\beta_i = \beta_i^s$, $\beta_j = \beta_j^s$ et $\beta_i^s = \beta_j^s$

$\gamma_{k,i} = \gamma_{k,j}$ puisque :

si k est persistante

Par hypothèse, $\gamma_{k,i} = \beta_k - \alpha_i^s$, $\gamma_{k,j} = \beta_k - \alpha_j^s$ et $\alpha_i^s = \alpha_j^s$

si k est nouvellement sensibilisée

Par hypothèse, $\gamma_{k,i} = \beta_k^s - \alpha_i^s$, $\gamma_{k,j} = \beta_k^s - \alpha_j^s$, et $\alpha_i^s = \alpha_j^s$

D'où $i =_D j$ et donc F est vraie

Si i est persistante et j nouvellement sensibilisée (ou l'inverse, par symétrie):

$\alpha_i \leq \alpha_j$

Puisque j est nouvellement sensibilisée, nous avons $\alpha_j = \alpha_j^s$, et donc $\alpha_i > \alpha_j \Leftrightarrow \alpha_i > \alpha_j^s$, ce qui est impossible car, par hypothèse, $\alpha_j^s = \alpha_i^s$ et par le Lemme 1.12, $0 \leq \alpha_i \leq \alpha_i^s$.

$\beta_i \leq \beta_j$

De la même manière, puisque j est nouvellement sensibilisée, nous avons $\beta_j = \beta_j^s$ et donc $\beta_i > \beta_j \Leftrightarrow \beta_i > \beta_j^s \Leftrightarrow \beta_i > \beta_i^s$, ce qui contredit le Lemme 1.12.

$(\forall k \neq i, j)(\gamma_{k,j} \leq \gamma_{k,i})$

Si k est persistante,

nous avons

$\gamma_{k,j} = \beta_k - \alpha_j^s$ car j est nouvellement sensibilisée,

$\alpha_j^s = \alpha_i^s$ par hypothèse,

$\beta_k - \alpha_i^s \leq \beta_k - \alpha_i$ puisque, par le Lemme 1.12, $0 \leq \alpha_i \leq \alpha_i^s$ et $0 \leq \beta_k$.

Maintenant, si $\gamma_{k,i} = \beta_k - \alpha_i$ alors

$\gamma_{k,j} = \beta_k - \alpha_j^s = \beta_k - \alpha_i^s \leq \beta_k - \alpha_i = \gamma_{k,i}$
et $\gamma_{k,j} \leq \gamma_{k,i}$

sinon, soit $(C_x)_{0 \leq x \leq p}$ une séquence de classes depuis la classe initiale jusqu'à la classe C (i.e. $C_p = C$) ; nous noterons $\alpha_i^x, \beta_i^x, \gamma_{k,i}^x$ les coefficients de la classe C_x ;

Soit C_n la dernière classe de cette séquence telle que $\gamma_{k,i}^n = \beta_k^n - \alpha_i^n$. Par le Lemme 1.12, C_n existe nécessairement, de plus pour tout $u \in]n, p]$, i et k sont persistantes dans la classe C_u et $\gamma_{k,i}^u = \gamma_{k,i}^{u-1}$.
Maintenant, nous avons $\beta_k \leq \beta_k^n$, $0 \leq \alpha_i \leq \alpha_i^s$, et par le Lemme 1.12, $0 \leq \beta_k$ et donc $\beta_k - \alpha_i^s \leq \beta_k^n - \alpha_i^n$
D'où $\gamma_{k,j} = \beta_k - \alpha_j^s = \beta_k - \alpha_i^s \leq \beta_k^n - \alpha_i^n = \gamma_{k,i}^n = \gamma_{k,i}^p$
et $\gamma_{k,j} \leq \gamma_{k,i}$

Si k est nouvellement sensibilisée

$\gamma_{k,j} > \gamma_{k,i} \Leftrightarrow \beta_k^s - \alpha_j > \beta_k^s - \alpha_i$
ce qui est impossible car $\alpha_i \leq \alpha_j$ (voir cas précédents)

Si i et j sont persistantes:

Nous raisonnons par induction sur les séquences de tir.

Supposons que $I_s(i) = I_s(j)$, $i \preceq_D j$, F, $(m, D) \xrightarrow{f} (m', D')$, i et j persistantes dans D', alors $\alpha'_i \leq \alpha'_j \wedge \beta'_i \leq \beta'_j \wedge (\forall k \neq i, j)(\gamma'_{k,j} \leq \gamma'_{k,i})$ (F')
$\alpha'_i \leq \alpha'_j$

Par le Lemme 1.12, nous avons $\alpha'_i = -\min_k \gamma_{k,i}$ et $\alpha'_j = -\min_k \gamma_{k,j}$,
$\gamma_{i,j} \leq \gamma_{j,i}$ et $(\forall l \neq i, j)(\gamma_{li} \geq \gamma_{lj})$ par hypothèse d'induction,
donc $-\min_k \gamma_{k,i} \leq -\min_k \gamma_{k,j}$ et $\alpha'_i \leq \alpha'_j$

$\beta'_i \leq \beta'_j$

nous avons $\beta'_i = \gamma_{i,f}$ et $\beta'_j = \gamma_{j,f}$ par le Lemme 1.12, $\gamma_{i,f} \leq \gamma_{j,f}$ par hypothèse d'induction, et donc $\beta'_i \leq \beta'_j$

$(\forall k \neq i, j)(\gamma'_{k,j} \leq \gamma'_{k,i})$

k est persistante

Par le Lemme 1.12,
$\gamma'_{k,i} = \min(\gamma_{k,i}, \beta'_k - \alpha'_i)$ et $\gamma'_{k,j} = \min(\gamma_{k,j}, \beta'_k - \alpha'_j)$
Il y a quatre cas à considérer:

1. $\gamma'_{k,i} = \gamma_{k,i}$ et $\gamma'_{k,j} = \gamma_{k,j}$: et donc $\gamma'_{k,j} \leq \gamma'_{k,i}$ puisque $\gamma_{k,j} \leq \gamma_{k,i}$ par hypothèse d'induction.
2. $\gamma'_{k,i} = \gamma_{k,i}$ et $\gamma'_{k,j} = \beta'_k - \alpha'_j$: nous avons, par le Lemme 1.12, $\gamma'_{k,j} = \beta'_k - \alpha'_j \leq \gamma_{k,j}$ et, par hypothèse d'induction, $\gamma_{k,j} \leq \gamma_{k,i}$
   d'où $\gamma'_{k,j} \leq \gamma_{k,j} \leq \gamma_{k,i} \leq \gamma'_{k,i}$ et finalement $\gamma'_{k,j} \leq \gamma'_{k,i}$
3. $\gamma'_{k,i} = \beta'_k - \alpha'_i$ et $\gamma'_{k,j} = \gamma_{k,j}$: Puisque $\alpha'_i \leq \alpha'_j$ (montré ci-avant) et que β_k, α'_i et α'_j sont positifs ou nuls (Lemme 1.12), nous avons $\beta'_k - \alpha'_i \geq \beta'_k - \alpha'_j$
   D'où $\gamma'_{k,j} \leq \beta'_k - \alpha'_j \leq \beta'_k - \alpha'_i = \gamma'_{k,i}$, et finalement $\gamma'_{k,j} \leq \gamma'_{k,i}$
4. $\gamma'_{k,i} = \beta'_k - \alpha'_i$ et $\gamma'_{k,j} = \beta'_k - \alpha'_j$: alor $\gamma'_{k,i} \geq \gamma'_{k,j}$ car $\alpha'_i \leq \alpha'_j$ (par les cas précédents)

k est nouvellement sensibilisée

Par le Lemme 1.12, $\gamma'_{k,i} = \beta^s_k - \alpha'_i$, $\gamma'_{k,j} = \beta^s_k - \alpha'_j$
Nous avons montré plus haut que $\alpha'_i \leq \alpha'_j$, et par le Lemme 1.12 nous savons que les alphas et betas sont positifs ou nuls finalement $\beta^s_k - \alpha'_i \geq \beta^s_k - \alpha'_j$ et par conséquent $\gamma'_{k,i} \geq \gamma'_{k,j}$

□

La relation $\preceq_D$ est réflexive, antisymétrique et transitive. Elle définit un ordre sur les transitions I_s-équivalentes sensibilisées par m, ce que prouve le Lemme 4.6.

Lemme 4.6. *La relation $\preceq_D$ définie par DEF-$\preceq_D$ est une relation d'ordre sur les transitions I_s-équivalente sensibilisées par m.*

Démonstration. Nous montrons que $\preceq_D$ est réflexive, transitive et antisymétrique.

1. $\preceq_D$ est réflexive.

$$(\forall i \in \mathcal{E}(m))(\gamma_{i,i} \preceq_D \gamma_{i,i} \wedge (\forall k \neq i)(\gamma_{i,k} \preceq_D \gamma_{i,k})$$

2. $\preceq_D$ est transitive.
   Soit $i, j, l \in \mathcal{E}(m)$ avec $i \neq j \neq l$. Par définition et comme l est différente de i et j, on a $\gamma_{i,l} \leq \gamma_{j,l}$ et $\gamma_{l,i} \geq \gamma_{l,j}$.
   Comme $j \preceq_D l$, on a $\gamma_{i,l} \leq \gamma_{j,l}$ et $\gamma_{l,i} \geq \gamma_{l,j}$.
   Or, puisque $j \preceq_D l$, nous avons $\gamma_{j,l} \leq \gamma_{l,j}$. Et donc

$$\gamma_{i,l} \leq \gamma_{j,l} \leq \gamma_{l,j} \leq \gamma_{l,i}$$

3. $\preceq_D$ est antisymétrique.
   Finalement, $\gamma_{i,j} \leq \gamma_{j,i} \wedge \gamma_{j,i} \leq \gamma_{i,j} \Leftrightarrow \gamma_{i,j} = \gamma_{j,i}$. De plus si pour tout k, $\gamma_{i,k} \leq \gamma_{j,k} \wedge \gamma_{j,k} \leq \gamma_{i,k}$ alors $\gamma_{i,k} = \gamma_{j,k}$; et $\gamma_{k,i} \geq \gamma_{k,j} \wedge \gamma_{k,j} \geq \gamma_{k,i}$ alors $\gamma_{k,i} = \gamma_{k,j}$

□

Deux transitions I_s-équivalentes sensibilisées par m sont toujours comparables par $\preceq_D$: c'est un ordre total. Le Lemme 4.7 prouve que $\preceq_D$ est un ordre total en considérant les trois cas possibles pour une paire de transitions dans D. Le premier cas correspond à l'introduction simultanée de i et j dans le système D (elles ont le même âge). Dans le second cas, elles sont introduites à des instants différents (une est plus vieille que l'autre). Finalement, le troisième cas énonce que les relations entre i et j sont préservées, quelques soient les transitions introduites, tant que i et j persistent dans D.

Lemme 4.7. *Soit (m, D) une classe du SCG et i, j deux transitions I_s-équivalentes sensibilisées par m. Alors, on a*

$$i \preceq_D j \vee j \preceq_D i$$

Démonstration. Les deux transitions i et j sont nouvellement sensibilisées dans (m, D):

D'où $\gamma_{i,j} = \beta^s_i - \alpha^s_j$, $\gamma_{j,i} = \beta^s_j - \alpha^s_i$ et $\gamma_{i,j} = \gamma_{j,i}$ puisque $I_s(i) = I_s(j)$.
Pour tout $k \neq i, j$, $\gamma_{i,k} = \beta^s_i - \alpha_k$, $\gamma_{j,k} = \beta^s_j - \alpha_k$ et donc $\gamma_{i,k} = \gamma_{j,k}$
puisque $I_s(i) = I_s(j)$.
D'où $i =_D j$; $i \preceq_D j$ et $j \preceq_D i$ sont vraies.

i est persistante et j nouvellement sensibilisée (ou inversement, par symétrie):

$\gamma_{i,j} \leq \gamma_{j,i}$

puisque j est nouvellement sensibilisée, nous avons $\gamma_{i,j} = \beta_i - \alpha_j^s$ et $\gamma_{j,i} = \beta_j^s - \alpha_i$
et donc $\gamma_{i,j} \leq \gamma_{j,i} \Leftrightarrow \beta_i + \alpha_i \leq \beta_j^s + \alpha_j^s$
mais $I_s(i) = I_s(j)$ par hypothèse,
donc $\gamma_{i,j} \leq \gamma_{j,i} \Leftrightarrow \beta_i + \alpha_i \leq \beta_i^s + \alpha_i^s$.
par le Lemme 1.12 nous avons $\alpha_i \leq \alpha_i^s$ et $\beta_i \leq \beta_i^s$
et ainsi $\beta_i + \alpha_i \leq \beta_i^s + \alpha_i^s$ et $\gamma_{i,j} \leq \gamma_{j,i}$

$(\forall k \neq i, j)(\gamma_{i,k} \leq \gamma_{j,k})$:

Puisque j est nouvellement sensibilisée, nous avons $\gamma_{j,k} = \beta_j^s - \alpha_k$
et donc $\gamma_{i,k} > \gamma_{j,k} \Leftrightarrow \beta_j^s - \alpha_k < \gamma_{i,k}$
mais $\gamma_{i,k} \leq \beta_i - \alpha_k$ par le Lemme 1.12
Ainsi $\gamma_{i,k} > \gamma_{j,k} \Leftrightarrow \beta_j^s - \alpha_k < \beta_i - \alpha_k \Leftrightarrow \beta_j^s < \beta_i$,
ce qui est impossible car $\beta_j^s = \beta_i^s$ par hypothèse et $\beta_i \leq \beta_i^s$ par le Lemme 1.12.

Ainsi $i \prec_D j$ (ou $j \prec_D i$, par symétrie)

Les deux transitions i et j sont persistantes et $i \preceq_D j$ (ou $j \preceq_D i$, par symétrie):

par induction, en s'appuyant sur le Lemme 1.12:

Initialement: Si j est nouvellement sensibilisée dans D, où i est persistante, alors $i \preceq_D j$ est vraie (voir cas précédent)

Etape d'induction: Supposons que i et j sont persistantes dans (m, D) et (m', D'), $i \preceq_D j$ et $(m, D) \xrightarrow{f} (m', D')$

$\gamma'_{i,j} \leq \gamma'_{j,i}$

Par le Lemme 1.12 nous avons
$\gamma'_{i,j} = \min(\gamma_{i,j}, \beta'_i - \alpha'_j) = \min(\gamma_{i,j}, \gamma_{i,f} - \alpha'_j)$
$\gamma'_{j,i} = \min(\gamma_{j,i}, \beta'_j - \alpha'_i) = \min(\gamma_{j,i}, \gamma_{j,f} - \alpha'_i)$
Il y a quatre cas à considérer

1. $\gamma'_{i,j} = \gamma_{i,j}$ et $\gamma'_{j,i} = \gamma_{j,i}$: $\gamma_{i,j} \leq \gamma_{j,i}$ par hypothèse, d'où $\gamma'_{i,j} \leq \gamma'_{j,i}$
2. $\gamma'_{i,j} = \gamma_{i,j}$ et $\gamma'_{j,i} = \gamma_{j,f} - \alpha'_i$: $\gamma_{i,f} \leq \gamma_{j,f}$ par hypothèse d'induction et $\alpha'_j \geq \alpha'_i$ par le Lemme 4.5 d'où $\gamma'_{i,j} \leq \gamma_{i,f} - \alpha'_j \leq \gamma_{j,f} - \alpha'_i = \gamma'_{j,i}$, c'est à dire $\gamma'_{i,j} \leq \gamma'_{j,i}$
3. $\gamma'_{i,j} = \gamma_{i,f} - \alpha'_j$ et $\gamma'_{j,i} = \gamma_{j,i}$: alors par hypothèse d'induction $\gamma'_{i,j} \leq \gamma_{i,j} \leq \gamma_{j,i} = \gamma'_{j,i}$, et ainsi $\gamma'_{i,j} \leq \gamma'_{j,i}$
4. $\gamma'_{i,j} = \gamma_{i,f} - \alpha'_j$ et $\gamma'_{j,i} = \gamma_{j,f} - \alpha'_i$: d'où par hypothèse d'induction et le Lemme 4.5: $\gamma'_{i,j} = \gamma_{i,f} - \alpha'_j \leq \gamma_{j,f} - \alpha'_i = \gamma'_{j,i}$, et ainsi $\gamma'_{i,j} \leq \gamma'_{j,i}$

$(\forall k \neq i, j)(\gamma'_{i,k} \leq \gamma'_{j,k})$

si k est persistante

$\gamma'_{i,k} = \min(\gamma_{i,k}, \beta'_i - \alpha'_k) = \min(\gamma_{i,k}, \gamma_{i,f} - \alpha'_k)$
$\gamma'_{j,k} = \min(\gamma_{j,k}, \beta'_j - \alpha'_k) = \min(\gamma_{j,k}, \gamma_{j,f} - \alpha'_k)$
Nous avons quatre cas à considérer :

1. $\gamma'_{i,k} = \gamma_{i,k}$ et $\gamma'_{j,k} = \gamma_{j,k}$: $\gamma_{i,k} \leq \gamma_{j,k}$ par hypothèse d'où $\gamma'_{i,k} \leq \gamma'_{j,k}$
2. $\gamma'_{i,k} = \gamma_{i,k}$ et $\gamma'_{j,k} = \gamma_{j,f} - \alpha'_k$: $\gamma_{i,f} \leq \gamma_{j,f}$ par hypothèse d'induction, donc
   $\gamma'_{i,k} \leq \gamma_{i,f} - \alpha'_k \leq \gamma_{j,f} - \alpha'_k = \gamma'_{j,k}$, et ainsi $\gamma'_{i,k} \leq \gamma'_{j,k}$

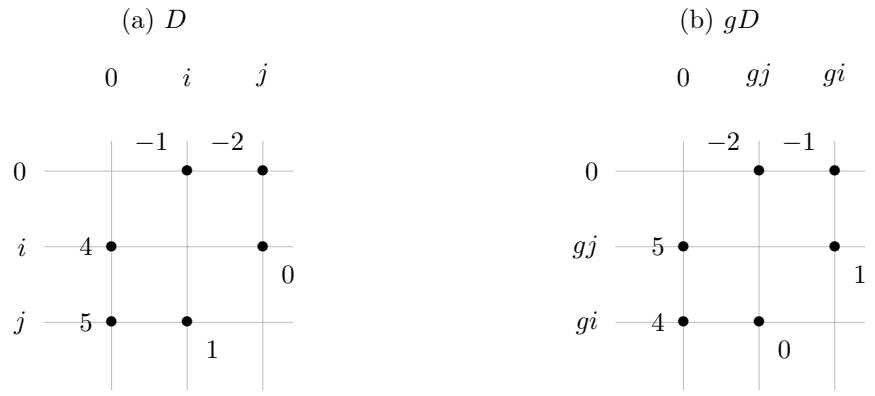


FIGURE 4.2: Exemple de permutation de domaines de tir (avec $g = (i, j)$)

3. $\gamma'_{i,k} = \gamma_{i,f} - \alpha'_k$ et $\gamma'_{j,k} = \gamma_{j,k}$: alors par hypothèse d'induction nous avons
   $\gamma'_{i,k} \leq \gamma_{i,k} \leq \gamma_{j,k} = \gamma'_{j,k}$, ainsi $\gamma'_{i,k} \leq \gamma'_{j,k}$
4. $\gamma'_{i,k} = \gamma_{i,f} - \alpha'_k$ et $\gamma'_{j,k} = \gamma_{j,f} - \alpha'_k$: alors par hypothèse d'induction
   $\gamma'_{i,k} = \gamma_{i,f} - \alpha'_k \leq \gamma_{j,f} - \alpha'_k = \gamma'_{j,k}$, d'où $\gamma'_{i,k} \leq \gamma'_{j,k}$

si k est nouvellement sensibilisée

$\gamma'_{i,k} = \beta'_i - \alpha'_k = \gamma_{i,f} - \alpha'_k$
$\gamma'_{j,k} = \beta'_j - \alpha'_k = \gamma_{j,f} - \alpha'_k$
ainsi $\gamma'_{i,k} \leq \gamma'_{j,k}$ est vraie par hypothèse d'induction ($\gamma_{i,f} \leq \gamma_{j,f}$)

en conséquence $i \preceq_{D'} j$.

□

L'ordre $\preceq_D$ permet donc de comparer des transitions I_s-équivalentes dans un domaine de tir D. Par définition, les transitions équivalentes par symétries sont I_s-équivalente. L'ordre $\preceq_D$ nous permet donc de comparer des transitions équivalentes par symétries. A partir de cette ordre, nous construisons dans la section suivante un ordre sur les classes d'états équivalentes.

4.3.2 Un ordre total sur les classes d'états équivalentes

Soit (G, N, κ) un G-réseau , $SCG = \langle \mathcal{C}, \rightarrow \subseteq \mathcal{C} \times T \times \mathcal{C} \rangle$ le graphe des classes, (m, D) une classe du SCG. Tous les domaines de tirs D sont en forme canonique. Soit deux transitions I_s-équivalentes i et j de T sensibilisées par m. Nous dirons de deux transitions appartenant à une même orbite qu'elles sont G-équivalentes. Par définition des symétries pour les TPN (Déf. 3.1), deux transitions G-équivalentes sont aussi I_s-équivalentes.

Cette section définit un ordre total sur les classes G-équivalentes.

La construction de l'ordre repose sur l'observation que l'action de G^κ sur les domaines préservent les relations d'ordre : i est plus petite par $\leq_D$ que j dans D si et seulement si l'image de i est plus petite que l'image de j dans l'image de D. Cette propriété est illustrée dans la Figure 4.2 et prouvée dans le Lemmme 4.8.

Lemme 4.8. *Soit (G, N, κ) un G-réseau , $SCG = \langle \mathcal{C}, \rightarrow \subseteq \mathcal{C} \times T \times \mathcal{C} \rangle$ le graphe des classes, (m, D) une classe du SCG et i et j deux transitions G-équivalentes sensibilisées par m. Alors :*

$$i \preceq_D j \Leftrightarrow gi \preceq_{gD} gj$$

Démonstration. Supposons que $D = \mathcal{F}(A \times \underline{\phi} \leq \underline{w})$ (Définition GD). Alors

$$\begin{aligned} & g\mathcal{F}(A \times \underline{\phi} \leq \underline{w}) = \mathcal{F}(A \times g^{-1}\underline{\phi} \leq \underline{w}) \\ \Leftrightarrow\ & g\mathcal{F}(A \times g\underline{\phi} \leq \underline{w}) = \mathcal{F}(A \times g^{-1}g\underline{\phi} \leq \underline{w}) \\ \Leftrightarrow\ & g\mathcal{F}(A \times g\underline{\phi} \leq \underline{w}) = \mathcal{F}(A \times \underline{\phi} \leq \underline{w}) \end{aligned}$$

Soit $\gamma'_{i,j}$ la contrainte de différences entre i et j dans gD (eg: $i - j \leq \gamma'_{i,j}$). Alors il est clair que :

$$\gamma'_{gi,gj} = \gamma_{i,j} \wedge \gamma'_{gj,gi} = \gamma_{j,i} \wedge (\forall k)(\gamma'_{gi,k} = \gamma'_{gj,k})$$

Et donc d'après la définition de $\preceq_D$, on a bien $i \preceq_D j \Leftrightarrow gi \preceq_{gD} gj$. □

Le Lemme 4.8 est vrai en particulier pour $j = g^{-1}i$:

$$i \preceq_D g^{-1}i \Leftrightarrow gi \preceq_{gD} i$$

Supposons que $i \preceq_D g^{-1}i$. Dans ce cas, la transition i dans D sera plus petite que la transition i dans gD et nous dirons alors que D est plus petite que gD. Plus précisément on a par le Lemme 4.8, avec les constantes de gD marquées par une cote (eg: $\gamma'_{i,j}$):

$$\begin{aligned} & i \preceq_D g^{-1}i = \gamma_{i,g^{-1}i} \leq \gamma_{g^{-1}i,i} \wedge (\forall k)(\gamma_{i,k} \leq \gamma_{g^{-1}i,k}) \\ \Leftrightarrow\ & gi \preceq_{gD} i = \gamma'_{gi,i} \leq \gamma'_{i,gi} \wedge (\forall k)(\gamma'_{gi,k} \leq \gamma'_{i,k}) \end{aligned}$$

On peut en deduire que $\gamma_{i,g^{-1}i} \leq \gamma_{gi,i} = \gamma'_{i,gi}$. De même pour tout k, on a $\gamma_{i,k} \leq \gamma'_{i,k}$. A partir de cette observation, nous définissons un ordre $\preceq_T$ qui permet de comparer les classes G-équivalentes.

Dans un premier temps nous nous limitons aux classes G-équivalentes de même marquage. Soit G_m le stabilisateur de m dans G. Alors pour tout $g \in G_m$, on a $g(m, D) = (m, gD)$. De plus, on suppose l'existence d'un ordre total $\leq_T$ sur les transitions de T (eg: l'ordre lexicographique sur les noms des transitions). Pour tout $g \in G_m$, on définit l'ordre $\preceq_T$ comme un ordre lexicographique sur le vecteur des transitions ordonnées par $\leq_T$ et les transitions comparées par $\preceq_D$. Plus formellement :

$$D \preceq_T gD \stackrel{\text{déf}}{=} (D = gD) \vee (\exists i)\left(i \prec_D g^{-1}i \wedge (\forall j <_T i)(j =_D gj)\right) \qquad \text{(DEF-}\preceq_T\text{)}$$

Le Lemme 4.9 montre que $\preceq_T$ est un ordre total sur les domaines de tirs G-équivalents.

Lemme 4.9. *La relation $\preceq_T$ définie par DEF-$\preceq_T$ ordonne totalement les domaines de tir G-équivalents.*

Démonstration. Nous montrons que $\preceq_T$ est réflexive, transitive et antisymétrique.

1. $\preceq_T$ est réflexive.
   Comme $D = ()D$ on a $D \preceq_T ()D$ par définition de $\preceq_T$.
2. $\preceq_T$ est transitive.
   Soit $a, b \in G_m$ et supposons que $D \preceq_T aD \preceq_T baD$. Par hypothèse, il existe i tel que $i \preceq_{aD} b^{-1}i$. Par le Lemme 4.8 on a :

   $$i \preceq_{aD} b^{-1}i \Leftrightarrow a^{-1}i \preceq_D a^{-1}b^{-1}i = (ba)^{-1}i$$

   Par hypothèse, il existe un i tel que $i \preceq_D a^{-1}i$ et donc il existe i tel que

   $$i \preceq_D a^{-1}i \preceq_D (ba)^{-1}i$$

   Par définition de $\preceq_T$ on a donc $D \preceq_T baD$ et donc :

   $$D \preceq_T aD \preceq_T baD \Rightarrow D \preceq_T baD$$

   D'où $\preceq_T$ est transitive.

3. $\preceq_T$ est antisymétrique.
   Si $D = gD$ il n'y a rien à prouver. Donc supposons que $D \prec_T gD$ et $gD \prec_T D$. Alors par transitivité, on a $D \prec_T D$, ce qui est une contradiction, d'où $D \preceq_T gD \wedge gD \preceq_T D \Rightarrow D = gD$
4. $\preceq_T$ est totale : $\leq_T$ et $\preceq_D$ sont des ordres totaux, d'où $\preceq_T$ est un ordre total.

□

Nous pouvons maintenant définir un ordre sur les classes G-équivalentes et plus uniquement sur les domaines équivalents. Cela revient à considérer l'action de G sur les classes plutôt que l'action de G_m sur les domaines et à comparer les marquages en plus des domaines. La comparaison de marquages se résume à une comparaison de vecteurs d'entiers.

Nous définissons donc un ordre lexicographique $\preceq_P$ sur les marquages, en supposant un ordre total $\leq_P$ sur les places (eg: un ordre lexicographique sur les noms des places) :

$$m \preceq_P m' \stackrel{\text{def}}{=} ((\forall p)(m(p) = m'(p))) \vee (\exists p)\left(m(p) \leq m'(p) \wedge (\forall k \leq_P P)(m(k) = m'(k))\right) \quad \text{(DEF-}\preceq_P\text{)}$$

Associé à $\preceq_T$, $\preceq_P$ permet de construire un ordre total $\preceq$ entre les classes G-équivalentes. Soit $g \in G$, $C = (m, D)$ et $C' = (m', D')$ deux classes G-équivalentes.

L'intuition ici est que lorsqu'on compare deux classes, on commence d'abord par comparer les marquages et si ceux-ci sont égaux, alors on compare les domaines. Nous définissons donc l'ordre $\preceq$ comme un ordre lexicographique :

$$(m, D) \preceq (m', D') \stackrel{\text{déf}}{=} (m, D) = (m', D') \vee (m \prec_P m') \vee m =_P m' \wedge D \preceq_T D' \quad (\preceq\text{-DEF})$$

Le Théorème 4.10 conclut cette section. Il montre que $\preceq$ est un ordre total sur les classes équivalentes par l'action de G.

Théorème 4.10. *La relation $\preceq$ est un ordre total sur les classes d'états équivalentes par l'action de G.*

Démonstration. Les ordres $\preceq_P$ et $\preceq_T$ sont totaux. □

L'ordre $\preceq$ permet d'ordonner les classes G-équivalentes. Il est donc possible de trouver une plus petite parmi ces classes G-équivalentes. Néanmoins cette propriété n'est pas suffisante en pratique. Il faut proposer un algorithme qui permet de calculer la plus petite classe.

4.3.3 Canonisation des RCD

Soit $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ un RCD obtenu par composition symétrique, $SCG = \langle \mathcal{C}, \rightarrow \subseteq \mathcal{C} \times T \times \mathcal{C} \rangle$ le graphe des classes et (m, D) une classe du SCG. Nous appelerons (m, D) un *état global*. On veut calculer le représentant canonique de l'état global. Comme discuté dans le Chapitre 2, le problème est NP-complet. Nous privilégions l'efficacité à la généralité en nous limitant à la classe des RCD.

L'ordre $\preceq_D$ ordonne totalement les transitions G-équivalentes de D. On peut assigner à chaque transition sa position dans l'ordre et donc représenter l'état global comme un vecteur d'entiers. Il existe alors une bijection entre les noeuds de N et les indices de l'état global.

Cette propriété, appliquée à des RCD, permet de représenter l'état global par un vecteur d'états locaux. La définition des RCD permet de construire une partition des noeuds de N où chaque partie est associée à un seul composant. Par la bijection entre les noeuds

de N et l'état global, on peut construire une partition de l'état global et sa représentation en un vecteur d'états locaux.

Un ordre total $\preceq_C$ sur les états locaux est défini. Le vecteur d'états locaux est trié lexicographiquement en comparant les états locaux par $\preceq_C$. En ordonnant correctement les noeuds de N, nous pouvons garantir que si le vecteur d'états locaux est le plus petit de son orbite, alors l'état global l'est aussi.

Noeuds d'un composant La définition des noeuds d'un composant se déduit de celle des RCD. Pour tout $x \in P \cup T$, si $G_x^\kappa = \kappa(G_i)$ alors on dira que x appartient au composant i. Lorsque les noeuds de l'orbite de x ont le même stabilisateur que x, ce dernier sera assigné à un composant en fonction de sa position dans l'orbite par $\leq_N$.

Soit $pos_\sim(x, X)$ la fonction d'un ensemble X vers $\mathbb{N}$ qui étant donné un ordre total $\sim$ sur X retourne la position dans X de x par $\sim$.

Définition 4.11 (Noeuds d'un composant)**.** *Pour tout $i \in C$ l'ensemble N_i des* noeuds du composant *i est défini par :*

$$N_i = \begin{cases} \{x \in P \cup T \mid G_x^\kappa = \kappa(G_i)\} & \text{si } G_i \neq G_j \text{ pour tout } j \in i^G \\ \{x \in P \cup T \mid pos_{\leq_N}(x, x^{G^\kappa}) = i - min(i^G)\} & \text{si } G_i = G_j \text{ pour tout } j \in i^G \end{cases}$$

L'action de G^κ sur N induit une action de G sur les composants N_i définie, pour tout $g \in G$ et $i \in C$, par :

$$\kappa(g) \cdot N_i = N_{gi} \qquad (G\text{-act-}N_i)$$

où N_{gi} est l'image de l'ensemble N_i par $\kappa(g) \in G^\kappa$.

Soit deux composants G-équivalents. Il faut ordonner les éléments de ces composants de telle manière que les éléments à la même position soient dans la même orbite. Nous définissons, pour chaque composant i, un ordre total $\leq_i$ sur ses noeuds tel que :

$$(\forall x \in N_i)(pos_{\leq_i}(x, N_i) = k \Leftrightarrow (\forall g \in G)(pos_{\leq_{gi}}(\kappa(g)x, N_{gi}) = k) \qquad (\text{DEF-}\leq_i)$$

Cette ordre existe puisque tous les composants G-équivalents à N_i sont isomorphes donc de même taille et, par le Théorème 4.12, toutes les orbites auxquelles un élément de N_i appartient sont de même longueurs. De plus, pour tout $x \in N_i$ et $g \in G$, on a $\kappa(g) \in N_{gi}$ par G-act-N_i.

Théorème 4.12. *Soit (G, N, κ) un RCD. Pour tout $x, y \in P \cup T$, si x et y ont le même stabilisateur dans G^κ alors les orbites de x^{G^κ} et y^{G^κ} sont de mêmes longueurs.*

$$(\forall x, y \in P \cup T)(G_x^\kappa = \kappa(G_i) = G_y^\kappa \Rightarrow |x^{G^\kappa}| = |y^{G^\kappa}|)$$

Démonstration. Par le Théorème 1.3, si x et y ont le même stabilisateurs, alors ils ont la même orbite. □

En utilisant la définition de l'ordre $\preceq_i$, on peut définir une action ξ de G sur les noeuds de composants, pour tout $g \in G$ et $i \in C$, par :

$$g \cdot N_i = N_{gi} \qquad (\text{DEF-}\xi)$$

L'intérêt de ξ est qu'elle évite d'appliquer les éléments de G^κ et de de permuter les états en permutant uniquement les indices des états locaux. Cela peut être efficace lorsque les composants contiennent beaucoup de noeuds.

Nous définissons un ordre total $\leq_e$ sur les noeuds de N. Il ordonne les noeuds de N par rapport à leur orbite et leur composant. Les noeuds des plus petits composants dans les plus petites orbites des plus petites sortes sont placés devant.

Soit la relation $\leq_e$ définie, pour tout $i, j \in C$, $x_i \in N_i$, $y_i \in N_j$, par :

$$x_i \leq_e y_j \stackrel{\text{déf}}{=} \begin{cases} x_i \leq_i y_j & \text{si } i = j \\ i \leq j & \text{si } x_i \in y_j^{G^\kappa} \\ x_i \leq_i g y_j & \text{si } x_i \notin y_j^{G^\kappa} \text{ et } i \in j^G \text{ et } gj = i \\ i \leq j & \text{si } x_i \notin y_j^{G^\kappa} \text{ et } i \notin j^G \end{cases} \qquad \text{(DEF-}\leq_e\text{)}$$

L'exemple 4.3.1 illustre cette définition.

Exemple 4.3.1. *Soit $G = \langle (1,2),(3,4) \rangle$ et (G, N, κ) un réseau obtenu par composition symétrique avec deux sortes O_1 et O_2. La Figure 4.3 montre la décomposition en état locaux.*

O_1		O_2	
1	2	3	4
a	d	g	i
b	e	h	j
c	f		

FIGURE 4.3: Exemple de partition en états locaux

Les orbites sont

$$\{\{a,d\},\{b,e\},\{c,f\},\{g,i\},\{h,j\}\}$$

L'ordre $\leq_e$ parcourt les orbites de haut en bas et de gauche à droite. On obtient :

$$a \leq_e d \leq_e b \leq_e e \leq_e c \leq_e f \leq_e g \leq_e i \leq_e h \leq_e j$$

Le Théorème 4.13 montre que $\leq_e$ est un ordre total.

Théorème 4.13. *Le relation $\leq_e$ définie par DEF-$\leq_e$ est un ordre total*

Démonstration. 1. $\leq_e$ est réflexif.

Pour tout x_i, $x_i \leq_e x_i$ si $x_i \leq_i x_i$. Comme $\leq_i$ est total dans N_i, $\leq_e$ est réflexif.

2. $\leq_e$ est transitif.

Supposons $x_i \leq_e y_j$ et $y_j \leq_e z_k$.

(a) si $i = j = k$ alors $x_i \leq_i y_j \leq_i z_k$ et $x_i \leq_e z_k$.

(b) si $i = j \neq k$ alors on a $x_i \leq_i y_j$ et :

— $y_j \in z_k^{G^\kappa}$. Alors on a $i = j \leq k$. Comme $y_j \in z_k^{G^\kappa}$ et $i = j$, on a $x_i \notin z_k^{G^\kappa}$, $j \in k^G$ et il existe g telque $y_i = gz_k$. Comme $x_i \leq_i y_i$ on a $x_i \leq gz_k$. D'où $x_i \leq_e z_k$.

— $y_j \notin z_k^{G^\kappa}$, $j \in k^G$, $gj = k$. Il existe g tel que $y_j \leq_i gz_k$. Comme $x_i \leq_i y_j$ on ne peut pas avoir $x_i \in z_k^{G^\kappa}$. D'où $x_i \leq_i gz_k$ et donc $x_i \leq_e z_k$.

— $y_j \notin z_k^{G^\kappa}$, $j \notin k^G$. D'où $i = j \leq k$ et $x_i \leq_e z_k$.

(c) si $i \neq j \neq k$:

— $x_i \in y_j^{G^\kappa}$ et $y_j^{G^\kappa} \in z_k^{G^\kappa}$. Alors $i \leq j \leq k$ et $x_i \in z_k^{G^\kappa}$ d'où $x_i \leq_e z_k$.

— $x_i \in y_j^{G^\kappa}$ et $y_j^{G^\kappa} \notin z_k^{G^\kappa}$ et $j \in k^G$. Alors il existe g tel que $y_j \leq_j g z_k$ et comme $i \in j^G$ on a un h tel que $x_i \leq_i hgz_k$. D'où $x_i \leq_e z_k$.

— $x_i \in y_j^{G^\kappa}$ et $y_j^{G^\kappa} \notin z_k^{G^\kappa}$ et $j \notin k^G$. On a $j \leq k$, $i \leq j$ et $i \notin k^G$. D'où $x_i \leq_e z_k$.

(d) $i \neq j = k$ se déduit de $i = j \neq k$.

3. $\leq_e$ est antisymétrique.
Si $x_i \leq_e y_j$ et $y_j \leq_e x_i$ alors

— $i = j$ et $x_i = y_j$

— $x_i \in y_j^G$ alors $i \leq j$ et $j \leq i$ alors $i = j$ et $x_i = y_j$ (Déf. DEF-$\leq_i$)

— $x_i \notin y_j^{G^\kappa}$, $i \in j^G$. Il existe g, h tel que $x_i \leq_i gy_j$ et $y_j \leq_j hx_i$. Par la Définition DEF-$\leq_i$ on $x_i = y_j$.

— $x_i \notin y_j^{G^\kappa}$, $i \notin j^G$. Alors on a $i = j$ et $x_i = y_j$.

□

Ordre sur les états locaux des RCD L'ordre $\preceq_D$ permet de représenter un état global par un vecteur d'entiers. Formellement, l'état global est un vecteur d'entiers σ indicé par les places et transitions sensibilisées ordonnées par $\leq_e$. Pour tout $x \in P \cup \mathcal{E}(m)$, on a :

$$\sigma(x) \stackrel{\text{déf}}{=} \begin{cases} m(x) & \text{si } x \in P \\ pos_{\preceq_D}(x, x^{G^\kappa}) & \text{si } x \in \mathcal{E}(m) \end{cases}$$

L'état local d'un composant i, noté σ_i, est la projection de σ sur les places et transitions du composant i. Par définition de $\leq_e$, les noeuds du composant i sont ordonnés par $\leq_i$. La comparaison des états locaux de deux composants G-équivalents se fait via l'ordre lexicographique sur les vecteurs d'entiers. S'ils ne sont pas G-équivalents, on compare les identifiants.

Soit $\preceq_C \subseteq C \times C$ définie, pour tout $i, j \in C$, par:

$$i \preceq_C j \stackrel{\text{déf}}{=} \begin{cases} i \leq j & \text{si } i \notin j^G \\ \sigma_i \leq \sigma_j & \text{sinon} \end{cases} \qquad \text{(DEF-}\preceq_C\text{)}$$

Par la définition de $\leq_i$ (Déf. DEF-$\leq_i$), si x est la position k dans $\leq_i$, alors pour tout g tel que $gi = j$, gx est à la position k dans $\leq_j$. La conséquence est qu'on ne compare que des noeuds G-équivalents. L'ordre $\preceq_C$ est total puisque $\leq$ est total et qu'un ordre lexicographique sur un ordre total est total.

Soit $\underline{\sigma}$ un vecteur d'états locaux des composants indicés par C ordonné par $\leq$. Nous cherchons à minimiser $\underline{\sigma}$ en permutant les états locaux. Pour ce faire, on définit un ordre lexicographique pour comparer deux vecteurs d'états locaux. Soit $\leq_C$ la relation définie pour tout vecteur $\underline{\sigma}$ par :

$$g\underline{\sigma} \leq_C \underline{\sigma} \Leftrightarrow (\exists i \in C)(g^{-1}i \prec_C i \wedge (\forall j < i)(i =_C j)) \qquad \text{(DEF-}\leq_C\text{)}$$

L'ordre $\leq_C$ est total puisque $\preceq_C$ est total. Le Théorème 4.14 montre qu'en minimisant le vecteur d'états locaux on minimise l'état global.

Théorème 4.14. *Pour tout $g \in G$, tout état global σ et le vecteur d'états locaux $\underline{\sigma}$ correspondant :*

$$g\underline{\sigma} \leq_C \underline{\sigma} \Leftrightarrow g\sigma \leq \sigma$$

Démonstration. Si $g = ()$ alors $g\underline{\sigma} \leq_C \underline{\sigma} \Leftrightarrow \underline{\sigma} = \underline{\sigma}$ et $\sigma = \sigma$. Si $g \neq ()$:

1. $g\underline{\sigma} \leq_C \underline{\sigma} \Rightarrow g\sigma \leq \sigma$
   Par la Définition DEF-$\leq_C$ on a :
   $$(\exists i \in C)(g^{-1}i \prec_C i \wedge (\forall j < i)(i =_C j))$$
   On peut, sans perte de généralité, fixer $i = 1$ et $g^{-1}i > i$. Soit $p = g^{-1}i$. Par la Définition de $\preceq_C$ et comme i et p sont dans la même orbite, on a :
   $$\sigma_p < \sigma_i$$
   On en déduit de l'ordre lexicographique que :
   $$(\exists k \in N_i)(\sigma_p(g^{-1}k) < \sigma_i(k) \wedge (\forall l <_i k)(\sigma_p(g^{-1}l) = \sigma_i(l))$$
   On peut, sans perte de généralité, fixer $pos_{\leq_i}(k, k^{G^\kappa}) = 1$.
   Comme $p > i$, $g^{-1}k \in k^{G^\kappa}$ et les états locaux sont des projections de l'état global, on a par la Définition DEF-$\leq_e$:
   $$g^{-1}k <_e k \wedge \sigma(g^{-1}k) < \sigma(k)$$
   Ce qui équivaut à $k <_e gk \wedge \sigma(k) < \sigma(gk)$. D'où $g\sigma \leq \sigma$
2. $g\sigma \leq \sigma \Rightarrow g\underline{\sigma} \leq_C \underline{\sigma}$
   Par définition de l'ordre lexicographique on a :
   $$(\exists x_i \in P \cup T)(\sigma(g^{-1}x_i) < \sigma(x_i) \wedge (\forall y_j <_e x_i)(\sigma(y_j) = \sigma(x_i)))$$
   On peut, sans perte de généralité, fixer $pos_{\leq_e}(x_i, P \cup T) = 1$. On a $x_i <_e g^{-1}x_i$ et on pose $g^{-1}x_i = x_p \in N_p$ et $g^{-1}i = p$. Comme $x_i <_e x_p$ et $x_p \in x_i^{G^\kappa}$, on a $i < p$ par la Définition DEF-$\leq_e$. D'où :
   $$\sigma_p(x_p) < \sigma_i(x_i)$$
   Comme $pos_{\leq_e}(x_i, P \cup T) = 1$, on déduit de la Définition DEF-$\leq_C$ que $g\underline{\sigma} <_C \underline{\sigma}$.

□

Canonisation des vecteurs d'états locaux Le Théorème 4.14 nous permet de canoniser l'état global en canonisant le vecteur d'états locaux. La stratégie de réduction pour canoniser le vecteur d'état locaux dépend maintenant de la structure du groupe. Un des intérêts de l'ordre total sur les transitions G-équivalentes est que l'on peut réutiliser les solutions pour les systèmes non temporisées (Chap. 2).

En particulier nous pouvons utiliser les méthodes de réduction pour les groupes simples (Section 2.3.4) pour les réseaux de la classe des RCD.

Lorsque G est isomorphe à $Sym(C)$, nous appliquons la méthode des ensembles minorants. Cette stratégie consiste à minimiser le vecteur d'états locaux à la manière d'un tri à bulle en utilisant successivement toutes les transpositions sur C.

Si G est un groupe cyclique, on peut énumérer tous les éléments de G jusqu'à obtenir le représentant canonique.

Si G est un produit disjoint $K_{(1)} \times \ldots \times K_{(k)}$ et que pour tout $i \in 1..k$ les $K_{(i)}$ sont des groupes simples, alors on applique la stratégie de réduction adaptée pour chacun des $K_{(i)}$.

Si G est un produit couronne $K_{(1)} \times \ldots \times K_{(k)} \times L \simeq G_{(1)} \wr L'$ et que $G_{(1)}$ et L sont des groupes simples, on applique la stratégie de réduction adaptée à $G_{(1)}$ sur les k facteurs $K_{(i)}$ puis la stratégie adaptée à L.

Pour les autres groupes, il est possible d'utiliser une recherche avec retour arrière, toujours en minimisant le vecteur d'états locaux.

4.4 Conclusion

Dans ce chapitre, nous avons proposé une méthode de réduction pour les TPN.

Nous avons montré que les symétries des G-réseaux induisent des symétries dans les graphes des classes. Ces symétries prouvent l'existence d'une relation d'équivalence sur les classes. A partir de cette relation d'équivalence, il est possible de construire un quotient du graphe qui préserve l'accessibilité (modulo les symétries).

Les TPN sont des modèles temporisés. La dimension temporelle des réseaux restreint l'application des méthodes de réduction développées pour les modèles non temporisés. Nous avons défini un ordre total sur les transitions équivalentes par symétrie à partir duquel est construit une représentation particulières des classes. Dans cette représentation, l'information temporelle associée à une transition est abstraite par sa position dans l'ordre total. Cela permet de décider si une transition est sensibilisée depuis plus longtemps qu'une autre transition équivalente et de calculer une plus petite classe dans une orbite.

Finalement, nous avons défini une méthode de canonisation pour la classe des RCD. Cette classe de G-réseaux , définie dans le Chapitre 3, capture les conditions suffisantes à l'application des méthodes de canonisation efficaces. En particulier, lorsqu'un G-réseau est un RCD, il est possible de calculer le représentant canonique d'un état en permutant les états locaux.

Si elle permet des réductions efficaces, la classe des RCD est relativement restreinte, par rapport à la classe des réseaux qu'il est possible de construire par composition symétrique. Néanmoins, elle permet d'implémenter sur les TPN une grande partie des méthodes de réduction de l'état de l'art.

Étendre la classe des réseaux pour laquelle nous proposons des méthodes de réduction est un travail en cours. En particulier, nous souhaiterions optimiser des méthodes de réductions basées sur la recherche avec retour arrière. Il est important de noter que l'ordre total sur les transitions équivalentes que nous avons proposé permet de s'abstraire de la dimension temporelle des G-réseaux . Cette propriété rend possible la mise en oeuvre de méthodes de réductions qui supposent que l'état est représenté par un vecteur d'entiers.

Chapitre 5

Implémentation et expérimentations

Sommaire

Ce chapitre présente une implémentation de la méthode décrite dans les Chapitres 3 et 4.

Les deux problèmes d'une méthode de réduction par symétrie ont été traités. Nous avons réalisé un prototype qui implémente la construction des réseaux par composition symétriques. Ce prototype utilise l'outil GAP(Groups,Algorithms and Programming)[125] pour les calculs de groupes. Il permet de construire un réseau symétrique et de calculer le groupe de symétries. L'algorithme de la construction est décrit dans le Chapitre 3. Des exemples de constructions sont présentés.

Comme discuté dans le Chapitre 4, le problème de l'orbite n'a pas de solution polynomiale dans le cas général. Nous proposons une implémentation dans Tina de la méthode du calcul du représentant canonique. L'implémentation est restreinte à un classe de RCD pour lesquels le calcul des symétries est polynomial dans le cas général. Des résultats expérimentaux encourageants sont présentés.

5.1 Construction des G-réseaux

Cette section présente un prototype qui implémente la méthode de composition symétrique proposé dans le Chapitre 3. Ce prototype est implémenté en Python et dans le langage de script de GAP. Il a été utilisé pour construire les réseaux symétriques illustrés dans cette thèse.

5.1.1 GAP

GAP est un logiciel sous licence libre pour les calculs de groupes. GAP propose un interprète en ligne de commande, une librairie pour la manipulation des groupes et un

langage de programmation.

GAP propose de nombreux algorithmes pour les groupes de permutations. Étant donné un groupe de permutations G, spécifié par ses générateurs ou des fonctions dédiées (eg `G:=SymmetricGroup(3)`), GAP propose des fonctions permettant de calculer notamment les orbites de G (`Orbits(G)`), les cosets d'un sous-groupe, les actions (`Action(G,Ω)`) et de tester l'isomorphisme de groupe(`IsomorphismGroups(A,B)`).

GAP implémente les algorithmes à l'état de l'art et il est possible de l'intégrer à d'autres outils, via une librairie C ou des appels systèmes.

5.1.2 Implémentation d'un prototype

La méthode de composition symétrique est implémentée dans un prototype écrit en Python et GAP. Ce prototype a été utilisé pour la construction de tous les G-réseaux illustrant cette thèse. Il ne traite pas à ce jour les actions γ, uniquement par manque de temps de son développeur principal, l'auteur de cette thèse.

Le prototype est un script Python, appelé depuis la ligne de commande , qui communique avec une librairie de fonctions GAP via des appels systèmes. Il est paramétré par un groupe de permutations et une liste de couples identifiant/réseau où l'identifiant est le représentant de la sorte et le réseau le composant.

Le groupe peut être spécifié par une liste de générateur, ou via un des constructeurs proposés. Ces constructeurs génèrent des groupes particuliers. Dans sa version actuelle le prototype propose :

— Un constructeur de groupe symétrique, cyclique ou diédral, paramétré par le degré ;

— Un constructeur d'hypercubes, paramétré par la dimension.

L'exemple des trains de Genrich (Exemple 3.2.1) est construit en spécifiant le groupe par ses générateurs. Le réseau de la Figure 3.8 est construit via le constructeur de groupe cyclique. Le réseau de la Figure 3.7 est construit via le constructeur de groupe symétrique.

Les réseaux sont lus à partir de descriptions aux formats gérés par Tina (ie: `.net` ou `.ndr`). Ils sont représentés en mémoire comme des réseaux de la librairie SNAKE [1], une libraire Python pour la manipulation des réseaux de Petri.

Les spécifications de synchronisation sont calculées à partir des étiquettes du réseau. Une syntaxe est définie qui permet de référencer des composants de l'union disjointe depuis les réseaux fournis en paramètre.

L'exemple 5.1.1 illustre la construction d'un hypercube d'agents.

Exemple 5.1.1. *Le réseau construit dans cet exemple est tiré de [94]. Il modélise un hypercube de dimension d composé de n agents. Chaque agent a deux états, un critique et l'autre non critique. Il peut passer de l'état non critique à l'état critique si aucun de ses trois voisins n'est en section critique. Le réseau est illustré dans la Figure 5.1.*

Le groupe G calculé par le constructeur d'hypercube est isomorphe aux symétries de l'hypercube de dimension 3. La construction de ce groupe de permutations, paramétré par la dimension, est tiré de [126]. G est transitif : il n'y a donc qu'une seule sorte dans la composition.

Le réseau est obtenu par composition symétrique de 8 agents. L'agent associé à l'identifiant 1 est illustré dans la Figure 5.2.

Les étiquettes permettent de calculer une spécification de synchronisation. Une étiquette est une paire (α, i) où α est une constante et i un entier d'une sorte. La constante définit une portée tandis que l'entier désigne un identifiant de composant dans l'union disjointe.

1. `https://www.ibisc.univ-evry.fr/~fpommereau/SNAKES/`

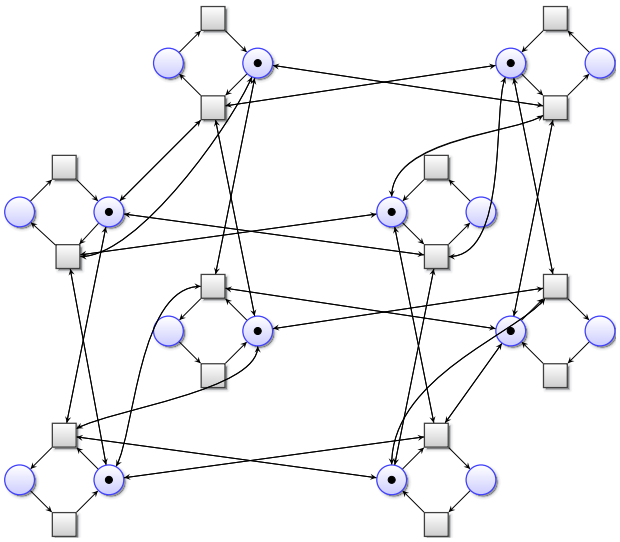


FIGURE 5.1: Un hypercube de dimension 3 avec 8 agents

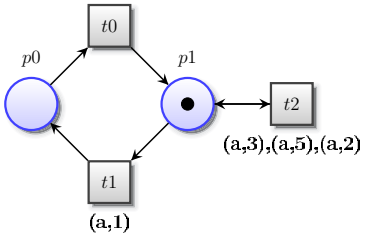


FIGURE 5.2: Un agent

Notons $t.j$ la transition t du composant j. Alors, dans la Figure 5.2, l'étiquette $(a, 1)$ sur la transition $t1$ et l'étiquette $(a, 3), (a, 5), (a, 6)$ sur la transition $t2$ génèrent la spécification de synchronisation $\{t1.1, t2.3, t2.5, t2.6\}$. Dans G, les trois voisins de 1 sont 3, 5 et 6. Cette spécification de synchronisation correspond à la description informelle du réseau.

La ligne de commande qui permet d'obtenir ce réseau est :

```
pi.py --Cube 3 1 agent.ndr
```

où `pi.py` *est le nom du script Python,* `--Cube 3` *spécifie un hypercube de dimension* 3 *et* 1 *est l'identifiant du composant* `agent.ndr` *(ie: le représentant de la sorte des agents). Le réseau construit est affiché sur la sortie standard au format textuel de description des réseaux de Tina (*`.net`*). Le groupe de symétries calculé est écrit dans le fichier. Il est présenté comme un sous groupe du groupe de symétries des places et transitions et défini par ses générateurs.*

Les calculs sur les groupes sont délégués à GAP. Tous les algorithmes que nous utilisons sont des algorithmes standards implémentés dans GAP. La Figure 5.3 donne un exemple d'interaction pour la construction d'un G-réseau à partir d'un produit couronne.

La ligne 1 de la Figure 5.3 lance le script interactif. La ligne 2 demande le groupe de permutations G. Dans cet exemple, le groupe est un produit couronne construit à la manière de la Section 3.3.3. Le groupe G est transitif, il n'y a qu'une sorte, dont le représentant est saisie à la ligne 3. Dans le cas général, la liste doit contenir un représentant par sorte. La ligne 4 demande de décrire le réseau. Dans GAP, nous ne considérons que les index de places et transitions. Ici le réseau associé à la sorte 1 est composé de 6 noeuds, numérotés de 1 à 6. L'action du stabilisateur de 1 dans G est généré par $\langle (3, 4), (4, 5), (6, 7), (7, 8) \rangle$.

```
1   gap> Pi();
2   Saisir le groupe : Group((3,4),(4,5),(6,7),(7,8),(1,2)(3,6)(4,7)(5,8));
3   Saisir la listes des representants : [1]
4   Saisir le réseau 1 (premier point 1) : [1..6]
5   Saisir une action de [ (3,4), (4,5), (6,7), (7,8) ] sur [ 1 .. 6 ] ([] si trivial) : [(1
6   Saisir une synchronisation ([[repr,noeud,g]+] ou [] pour terminer): [[1,1,()],[1,1,(1,2)
7     [(1,1),(2,7)] , [(1,1),(2,9)] , [(1,1),(2,11)] , [(1,3),(2,7)] , [(1,3),(2,9)] , [(1,3
8   Saisir une synchronisation ([[repr,noeud,g]+] ou [] pour terminer): []
```

FIGURE 5.3: Exemple d'interaction avec GAP

Le script demande à l'utilisateur de définir l'action de G sur le réseau 1 par ses générateurs. GAP vérifie alors que les paramètres saisis définissent bien une action. Ici l'action est conforme aux contraintes définies pour les produits couronnes.

Dans cet exemple, nous définissons des synchronisations qui impliquent que le réseau obtenu n'est pas un RCD (Section 3.3.3).

Les spécifications de synchronisation sont données une par une. Une spécification de synchronisation est une liste de triplets $[i, p, g]$ où :

— i est un identifiant de composant ;

— j un identifiant de noeud de ce composant. Les indices des copies de composants sont calculés par le script (eg: les indices du 3-ème composant commencent à $6 \times 2 + 1$) ; et

— g est un élément de G.

Le noeud désigné par le triplet $[i, j, p]$ est l'image par g du noeud j du composant i dans l'action de G sur l'union disjointe.

A partir de ces informations, le script construit le groupe G^κ tel que défini dans le Chapitre 3. Des fonctions compagnon du script GAP permettent d'afficher ce groupe, le script ne calcule que l'action de G pas son image. La Figure 5.4 donne une interaction avec GAP où l'on affiche le réseau synchronisé N et l'image de κ.

```
1.  gap> res := GAct();
[ [ [ rec( S_idx := 1, tau_idx := 1, x := 2 ) ], [ rec( S_idx := 1, tau_idx := 1, x := 4 ) ]
      [ rec( S_idx := 1, tau_idx := 1, x := 6 ) ], [ rec( S_idx := 1, tau_idx := 2, x := 8 )
    . . . ],
    <action epimorphism> ]
2.  gap> Image(res[2]);
3.    Group([ (1,2)(7,10)(8,11)(9,12), (2,3)(10,13)(11,14)(12,15), (4,5)(7,8)(10,11)(13,14),
(11,12)(14,15), (1,4)(2,5)(3,6)(8,10)(9,13)(12,14) ])
```

FIGURE 5.4: Représentation GAP du G-réseau (G, N, κ)

La fonction `GAct()` retourne la liste des noeuds de N et une action de G sur ce réseau (Figure 5.4). Un noeud est représenté par un triplet `[ S_idx, tau_idx, x ]` où :

— `S_idx` est l'indice de la sorte S dans le tableau des représentants (Fig 5.3, ligne 3) ;

— `tau_idx` est l'indice d'un élément de la transversale F, hérité de l'union disjointe ; et

— `x` est l'indice du noeud dans le réseau

L'action de G sur N est un homomorphisme de G vers les symétries de N. Son image est un groupe de permutations des noeuds de N.

5.1.3 Un mot sur la complexité

L'implémentation de la composition symétrique fait appel à plusieurs algorithmes classiques des groupes de permutations. Tous ces algorithmes sont implémentés dans GAP.

Une partie de ces traitements inclut le calcul d'orbite de G, du stabilisateur H d'un point dans G et d'une transversale de H. Ces calculs se font en temps polynomial.

Il faut ensuite calculer des actions (ie: μ,ν et κ). Le noyau d'une action et son image peuvent être calculés en temps polynomial par l'algorithme de Schreir-Sims [86], aussi utilisé pour le calcul des bases.

Le calcul du stabilisateur d'une spécification de synchronisation pose problème. Une spécification est un ensemble de transitions de l'union disjointe et il n'existe pas de solution polynomiale au problème du calcul du stabilisateur d'un ensemble. Néanmoins, l'utilisation de la recherche avec retour arrière, en choisissant comme base les éléments de l'ensemble, permet d'obtenir des résultats satisfaisants en moyenne.

5.2 Expérimentations avec Tina

Cette section présente une première implémentation dans Tina [2] de notre méthode d'exploitation des symétries pour les TPN.

L'implémentation se limite à une classe de RCD appelée RCD simples. Cette classe permet d'éviter le calcul des stabilisateurs des transitions, pour lequel il n'existe pas de solutions polynomiales dans le cas général.

La section présente des résultats expérimentaux encourageants.

5.2.1 Description des symétries

Tina a été étendu pour permettre la spécification et la réduction des RCD simples, temporisés ou non temporisés.

La Définition 5.1 définit la classe des RCD qu'il est possible de construire avec Tina.

Définition 5.1 (RCD simple)**.** *Soit un* $(G, N, \kappa) = \Pi(k, G, N_{1_1}, \ldots, N_{k_1}, \Psi, \gamma_{1_1}, \ldots, \gamma_{k_1})$ *un RCD obtenu par composition symétrique. Alors* (G, N, κ) *est un RCD simple si l'une des conditions suivantes est vérifiée :*

- $G \simeq S_n$ *et* $\Psi = \emptyset$ *;*
- $G \simeq C_n$ *et chaque noeud est synchronisé avec son voisin ;*
- (G, N, κ) *est un produit disjoint ou un produit couronne de RCD simples.*

Les réseaux construits sont décrits hiérarchiquement par composition de réseaux plus petits. Les constructions reposent sur une extension de `tpn` , un langage de description des réseaux par composition proposé par Tina. Deux opérateurs ad-hoc ont été ajoutés. Ils permettent respectivement de définir une symétrie totale ou circulaire à partir de n composants symétriques.

La construction repose sur une opération de produit de TPN étiquetés définie dans [121]. Nous rappelons cette définition. Soit S_1 et S_2 deux TPN étiquetés (Def. 1.9) et $S = \{(t_1, t_2) \in T_1 \times T_2 \mid \lambda(t_1) = \lambda(t_2)$ l'ensemble des paires de transitions de même étiquette. Le produit synchronisé de S_1 et S_2 est construit comme suit :

2. `http://projects.laas.fr/tina/symmetries`

1. Pour chaque paire de transitions $(t_1, t_2) \in S$ on ajoute une nouvelle transition ayant l'effet combiné de t_1 et t_2, le même label que t_1 et l'intersection des intervalles statiques de t_1 et t_2 ;
2. On supprime les transitions de S.

Les compositions sont obtenues par des produits synchronisés de TPN, notés $|$, des produits libres, notés $||$, et un opérateur de renommage des étiquettes. Le produit libre entrelace les transitions de ses composantes sans aucune synchronisation.

Deux opérateurs de composition, notés $\mathbf{P}(pool)$ et $\mathbf{R}(ring)$, permettent la spécification de symétries. Les deux opérateurs sont paramétrés par un réseau et le degré du groupe :

— $\mathbf{P}(n, C)$ spécifie un produit libre de n copies de C et une symétrie totale sur ces copies ;

— $\mathbf{R}(n, C)$ spécifie un produit synchronisé de n copies de C, où chaque composant est synchronisé avec ses voisins, et une symétrie circulaire sur ces copies.

Ces compositions permettent de construire les RCD simples et simplifient la construction du groupe de symétrie. Ce groupe se déduit des symétries de l'union disjointe. L'action des γ est ignorée, ce qui limite les symétries de l'union disjointe aux isomorphismes entre les copies.

L'exemple 5.2.1 illustre quelques compositions possibles avec le langage.

Exemple 5.2.1. — $(\boldsymbol{R}(8, P) \mid Q \mid \boldsymbol{P}(3, R))$ *décrit le TPN obtenu en synchronisant un anneau de 8 instances de P avec Q et un pool de 3 instances de R ;*

— $\boldsymbol{R}(6, P \mid \boldsymbol{P}(4, Q))$ *décrit un anneau de 6 instances du composant $P \mid \boldsymbol{P}(4, Q)$, ce dernier synchronisant P avec un pool de 4 instances de Q ;*

— $T \mid \boldsymbol{P}(2, S \mid \boldsymbol{P}(3, C))$ *décrit l'architecture client serveur présenté dans la Figure 2.6.*

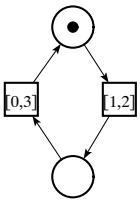
5.2.2 Expérimentations avec Tina

Cette section présente quelques expérimentations réalisées avec l'implémentation dans Tina de notre méthode d'exploitation des symétries. Elle présente d'abord une expérimentation où le groupe de symétries est isomorphe à S_n, puis une expérimentation où les groupes de symétries sont des produits disjoints ou couronnes.

Symétrie totale. Le premier exemple est un modèle de passage à niveau, dans sa version temporisée et non temporisée. La version temporisée est extraite de [18]. La version non temporisée présentée est un modèle simplifié tiré de [127]. Dans les deux cas, le réseau modélise un passage à niveau: un certain nombre de voies traversent une route, protégées par un passage à niveau ; lorsqu'un train approche ou quitte le croisement, il déclenche un signal émis vers le contrôleur qui lève ou ferme la barrière. Il faut s'assurer que les barrières du passage à niveau soient toujours baissées lorsqu'un train traverse la route. Dans la version temporisée, la propriété est assurée par des contraintes temporelles sur les évènements pertinents, tandis que la version non temporisée utilise des acquittements et un drapeau partagé.

Dans les deux cas, le modèle est obtenu en synchronisant un modèle des barrières (droite), le contrôleur (milieu) et un ensemble de voies. Le modèle temporisé est illustré dans la Figure 5.7 et le modèle non temporisé dans la Figure 5.6.

Les résultats pour la version non temporisée sont donnés dans la Figure 5.6. La colonne $R(nosym.)$ contient les tailles des espaces d'états et les temps de calculs, sans réduction, pour le nombre de voies indiqué dans la première colonne. La colonne $R_{\approx}(our)$ donne les tailles et temps de calculs de l'espace d'états quotient. La table montre que le gain en taille est exponentiel, ce qui est conforme à la symétrie du réseau. Les temps de calculs sont



description	$R_{\mathcal{N}}/\approx$		$SCG/\approx$		$SCG_{\subseteq}/\approx$	
	markings	transitions	classes	transitions	classes	transitions
$\mathbf{P}(6, N)$	7	42	5404	30694	7	42
$\mathbf{D}(\mathbf{P}(3, N)\ ,\ \mathbf{P}(3, N))$	16	96	72234	411960	69	414
$\mathbf{P}(2, \mathbf{P}(3, N))$	10	60	36154	206185	35	210
$\mathbf{R}(6, N)$	14	84	328984	1881380	327	1962
$\mathbf{D}(\mathbf{R}(3, N)\ ,\ \mathbf{R}(3, N))$	16	96	221600	1266984	225	1350
$\mathbf{R}(2, \mathbf{R}(3, N))$	10	60	110860	633824	113	678
$\mathbf{D}(\mathbf{R}(3, N)\ ,\ \mathbf{P}(3, N))$	32	192	661296	3781368	663	3978
$\mathbf{R}(2, \mathbf{P}(3, N))$	10	60	36154	206185	35	210
$\mathbf{P}(2, \mathbf{R}(3, N))$	10	60	110860	633824	113	678

FIGURE 5.5: Composant N et exemples de symétries.

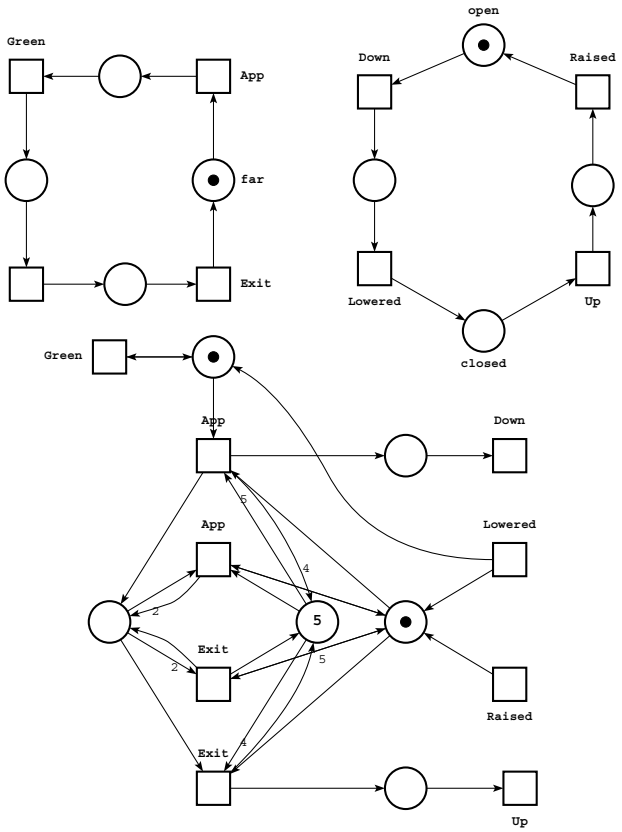
aussi significativement réduits. Les valeurs grisées ont été obtenues par une extrapolation manuelle en sommant les tailles des orbites des marquages du quotient. Il n'est pas possible de construire des espaces d'états de cette taille avec Tina qui est basé sur une approche énumérative.

La dernière colonne montre les résultats obtenus sur le même modèle avec la dernière version de LoLA (V2.0), qui implémente les techniques décrites dans [92]. Comme on peut le constater, LoLA utilise des représentants minimaux et non canoniques, ce qui produit plusieurs représentants pour une même orbite. En conséquence, le gain sur le nombre d'états obtenus est inférieur à celui obtenu par notre méthode. Cependant, LoLA est capable d'extraire plus de symétries que nous ne pouvons en exprimer avec le langage de composition de Tina. De plus, il les extrait automatiquement.

Comme discuté en conclusion du Chapitre 3, il est possible que LoLA, basé sur la détection automatique, extrait plus de symétries que l'opérateur de composition symétrique ne peut en décrire. L'avantage de notre méthode est la taille des réseaux qu'elle peut construire (et donc des symétries qu'elle permet de détecter). Dans tous les cas, l'opérateur de composition symétrique permet d'exprimer bien plus de symétries et de réseaux que le langage de composition de Tina que nous avons implémenté à ce jour. Des travaux en cours doivent permettre une implémentation plus complète.

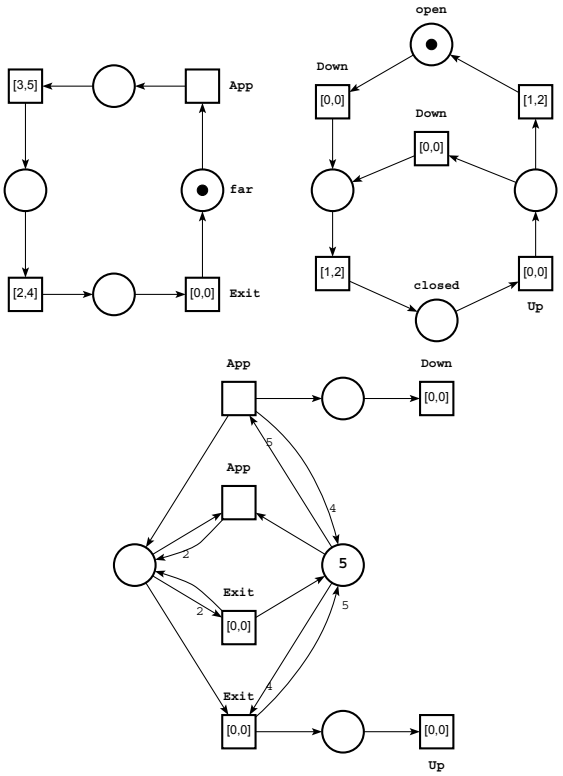
Les résultats obtenus sont donnés dans la Figure 5.6, pour l'abstraction des classes d'états (SCG) et celle de l'inclusion de classes ($SCG^{\subseteq}$). Comme pour le cas non temporisé, les valeurs grisées ont été obtenues par extrapolation.

Pour un même nombre de voies, le nombre de classes d'états dans le modèle temporisé (colonne SCG et $SCG_{\approx}$) est bien plus grand que le nombre de marquages du modèle non temporisé. Le gain obtenu par la réduction reste similaire à celui du cas non temporisé, autant pour la taille des espaces d'états que les temps de calcul. On peut remarquer que l'abstraction de l'inclusion des classes bénéficie elle aussi d'un important facteur de réduction.



tracks		R (*no sym.*)	$R/\approx$ (*our*)	$R/\approx$ (*lola*)
5	*size*	1036	60	92
	time	0.00	0.00	0.01
10		1048598	290	18836
		14.40	0.00	0.67
20		1.099×10^{12}	1775	303830981
		–	0.13	76221
50		1.267×10^{30}	23430	–
		–	9.42	–
100		1.606×10^{60}	176855	–
		–	225.18	–

FIGURE 5.6: Passages à niveau non temporisés



tracks		SCG	$SCG/\approx$	$SCG^{\subseteq}$	$SCG^{\subseteq}/\approx$
3		3101	578	172	41
		0.02	578	0.00	0.00
4	*size*	134501	6453	1175	76
	time	1.67	0.15	0.02	0.00
5		8557621	84510	10972	143
		179.33	2.61	0.38	0.01
6		697913229	1183782	128115	274
		24346.77	46.95	8.37	0.02
7		7.278×10^{10}	18143796	1772722	533
		–	1060.21	614.38	0.07
8		9.262×10^{12}	297205635	28208543	1048
		–	25105.87	177602.90	0.16
10		–	–	–	4126
		–	–	–	1.10
12		–	–	–	16420
		–	–	–	8.29
14		–	–	–	65578
		–	–	–	95.07
16		–	–	–	262192
		–	–	–	1418.38
18		–	–	–	1048630
		–	–	–	22407.25

FIGURE 5.7: Passages à niveau temporisés.

Produits de groupes La Table 5.5 montre les résultats obtenus pour des produits de groupes. Dans cette table, le réseau est un produit libre de 6 copies du composant illustré dans la Figure 5.5. La colonne de gauche donne, pour chaque ligne, la spécification des symétries. En ignorant les symétries et les informations temporelles, l'espace d'états de ces réseaux compte 64 marquages et 384 transitions. Lorsqu'on prend en compte l'information temporelle (mais pas les symétries) les réseaux comptent 1973488 classes d'états et 11285976 transitions. La colonne $R_{\mathcal{N}}/\approx$ donne le nombre d'états et de transitions dans le cas non temporisé avec réduction par symétrie. Les autres colonnes donnent les résultats pour les quotients du graphe des classes ($SCG/\approx$) et du graphe des classes avec inclusion ($SCG^{\subseteq}/\approx$).

5.3 Conclusion

Dans ce chapitre nous avons présenté une implémentation de la méthode de construction des réseaux symétriques et une implémentation de la méthode de réduction dans Tina. Les résultats sont encourageants.

La composition symétrique de réseaux utilise des algorithmes pour les groupes de permutations implémentés dans GAP. Un programme en langage Python permet de lire et écrire des réseaux de Petri et d'exécuter les calculs des groupes de symétries.

Notre méthode de réduction par calcul du représentant canonique est implémentée dans Tina, pour une classe des RCD simple à construire. Cette implémentation valide l'efficacité de l'ordre total sur les classes équivalentes défini dans le Chapitre 4 et de notre méthode de calcul du représentant canonique.

Conclusion générale, perspectives

Cette thèse a présenté mes travaux autour de la vérification formelle par model-checking dans un contexte avionique.

Nous avons présenté le contexte des systèmes avioniques, les méthodes de vérification et un exemple de vérification par model-checking de contraintes temporelles dans ce contexte. Plusieurs méthodes formelles sont utilisées aujourd'hui pour la vérification des systèmes avioniques, notamment la preuve de programme et l'interprétation abstraite. A notre connaissance, le model-checking s'est moins diffusé dans l'industrie aéronautique. Une des difficultés est le problème de l'explosion combinatoire. Nous pensons que le model-checking et le model-checking temps réel ont un rôle à jouer dans la vérification des systèmes avioniques parce qu'il est bien adapté à la vérification des systèmes distribués temps réel.

En particulier, les asynchronismes des architectures GALS peuvent activer des fautes de concurrence. De plus, la dimension temporelle est très importante dans le contexte avionique. C'est pour cette raison que nous avons proposé une expérimentation de vérification de contraintes temporelles dans ce contexte. Cette expérimentation contribue très modestement à montrer que le model-checking peut s'appliquer et aider à détecter au plus tôt des erreurs spécifiques aux systèmes distribués temps réel.

La contribution principale de cette thèse est une méthode de réduction par symétrie pour les TPN. Les méthodes de réduction par symétrie font face à deux problèmes : l'identification des symétries et la réduction des espaces d'états.

Un chapitre a été consacré à l'état de l'art des méthodes de réduction par symétrie. Deux approches pour l'identification des symétries y ont été présentées. La première repose sur des annotations du modèle (les scalarsets) et la deuxième sur la détection automatique. La détection automatique est la méthode la plus générale, précise et simple mais au détriment de l'efficacité. Les scalarsets offrent l'avantage de la simplicité pour l'utilisateur et de l'efficacité au détriment de la généralité et de la précision sur les groupes construits.

Plusieurs méthodes de réduction par symétrie ont été présentées. Les méthodes qui consistent à itérer sur les symétries d'un groupe ou sur les états, sont limitées par la taille exponentielle des groupes ou des espaces d'états. Les méthodes exploitant la recherche avec retour arrière comme les méthodes de calcul du représentant canonique nécessitent, pour être efficaces, de disposer d'un ordre total sur les états. Dans le cas des modèles non temporisés, cet ordre total est l'ordre lexicographique sur les vecteurs d'entiers. Les choses se compliquent pour les modèles temporisés à cause de la dimension temporelle et de sa représentation par une matrice de différences. Une méthode pour les automates temporisés a été présentée.

Il n'existe pas à notre connaissance de méthode de réduction par symétrie pour les TPN. Mes travaux de thèse comblent ce vide.

Notre méthode identifie les symétries par construction via un opérateur de composition symétrique. Cet opérateur de composition est très général. Il permet de construire un réseau symétrique par composition de sous-réseaux, sans contrainte sur les synchronisa-

tions et quelque soit le groupe de symétrie. Un groupe des symétries du réseau est calculé, isomorphe à celui spécifié par l'utilisateur. Nous ne savons pas s'il permet de décrire toutes les symétries du réseau construit. Un problème analogue est celui de déterminer le groupe des automorphismes d'un graphe de Cayley. Ce problème n'a pas de solution générale connue à ce jour.

L'opérateur de composition offre une solution efficace, générale et précise au problème de l'identification des symétries, au détriment de la simplicité d'utilisation. Néanmoins, il est assez simple de proposer à l'utilisateur des constructions de plus haut niveau, correspondant à des architectures de systèmes dont les symétries sont connues, par exemple des pools, des anneaux, des hypercubes,.... Ces constructions seraient, du point de vue de l'utilisateur, aussi simples d'utilisation que les scalarsets.

L'objectif est d'utiliser les symétries des réseaux pour réduire la taille de leurs espaces d'états. Il s'agit de construire un quotient de l'espace d'états par la relation d'équivalence induite par les symétries. Le problème fondamental pour cette construction est le problème de l'orbite. Ce dernier n'a pas de solution efficace dans le cas général.

Nous avons ensuite montré que les symétries d'un réseau induisent des symétries dans le graphe des classes. Il est possible de construire un quotient d'un graphe des classes à partir de la relation d'équivalence induite par les symétries du réseau. Ces quotients préservent l'accessibilité modulo les symétries. Nous proposons une méthode de construction de ces quotients.

Les TPN sont des modèles temporisés. La dimension temporelle des réseaux restreint l'application des méthodes de réduction développées pour les modèles non temporisés. Nous avons défini un ordre total sur les transitions équivalentes par symétrie, à partir duquel est construite une représentation particulière des classes. Dans cette représentation, l'information temporelle associée à une transition est abstraite par sa position dans l'ordre total. Cela permet de décider si une transition est sensibilisée depuis plus longtemps qu'une autre transition équivalente et de calculer une plus petite classe dans une orbite. La définition de cet ordre est une des contributions de cette thèse.

Nous avons défini une méthode de canonisation pour la classe des réseaux à composants disjoints (RCD). Cette classe est définie grâce à notre opérateur de composition. Décider si un réseau appartient à cette classe de réseaux se ramène au problème de l'isomorphisme de groupe, au moins aussi difficile que celui de l'isomorphisme de graphes. Cependant, la précision de notre construction permet de définir des conditions suffisantes sur l'opérateur permettant d'obtenir des RCD. La définition des RCD généralise des résultats de la littérature obtenus dans le contexte du model-checking non temporisé et des automates temporisés.

Un des avantages de notre ordre total sur les transitions équivalentes est qu'il permet l'application de méthodes développées pour les modèles non temporisés à la réduction du graphe des classes des TPN. En particulier, pour un groupe de symétries isomorphe à S_n, nous pouvons calculer le représentant canonique d'une classe, en temps polynomial.

Le dernier chapitre de cette thèse présente une implémentation de l'opérateur de composition symétrique ainsi qu'une implémentation de notre méthode de réduction dans TINA.

L'opérateur de composition exploite des algorithmes pour les groupes de permutations. Ces algorithmes sont implémentés dans l'outil GAP et nous les appelons pour construire les réseaux symétriques et calculer leurs symétries. Tous les exemples de cette thèse ont été construits par cette implémentation.

Notre méthode de réduction par calcul du représentant canonique est implémentée dans TINA, pour la classe des réseaux que nous savons traiter efficacement. Cette implémentation valide l'efficacité de notre ordre total et de notre méthode de calcul du représentant canonique. Des résultats expérimentaux encourageants ont été présentés.

Perspectives Plusieurs évolutions de mes travaux sont envisageables. Ces perspectives s'articulent sur trois axes : améliorer la phase de détection, élargir la classe des réseaux pour lesquels des solutions efficaces de réduction existent et étendre la méthode à d'autres modèles de vérification ou de symétrie.

Identification des symétries Si l'opérateur de composition est très général, il n'est pas en revanche assez simple d'utilisation. Le langage de composition décrit dans le Chapitre 5 propose une couche d'abstraction sur cet opérateur qui simplifie son utilisation. Un première amélioration est de proposer d'autres constructeurs de groupe : cube, diédral, ... qui correspondent à des architectures dont les symétries sont connues. L'autre possibilité est de définir un langage d'étiquettes symboliques qui permette de construire simplement les spécifications de synchronisation. Par exemple dans le cas d'une symétrie en anneau on pourrait proposer des étiquettes `next, prev` et pour des symétries totales des étiquettes `others, self` qui désignent, respectivement, tous les autres composants et le représentant de la sorte.

Une autre piste d'amélioration serait de proposer une approche mixte entre détection automatique des symétries et construction. Une première solution serait d'utiliser la détection automatique sur des petits composants puis d'utiliser l'opérateur de composition pour construire de plus grand réseaux à partir de ces composants en utilisant les symétries détectées. On pourrait aussi utiliser la détection automatique a posteriori de la construction, pour détecter si le réseau construit possède plus de symétries que celles spécifiées.

Exploitation des symétries La classe des RCD, si elle permet des réductions efficaces, est relativement restreinte par rapport à la classe des réseaux qu'il est possible de construire par compositions symétriques. Une perspective est d'élargir les constructions pour lesquelles nous proposerions des solutions de réduction efficaces.

Une première amélioration serait d'étendre la classe des groupes simples. Par exemple, la condition suffisante permettant de décider qu'un RCD est un anneau, repose sur le fait que dans une symétrie circulaire le stabilisateur d'un point dans G est trivial. Or, la théorie des groupes dit que tous les groupes commutatifs vérifient cette propriété. Il serait peut être possible d'étendre la classe des RCD aux groupes commutatifs. Plus généralement, il existe des résultats de la théorie des groupes où certains problèmes qui n'ont pas de solution efficace dans le cas général en ont pour certaine classes de groupes. Le problème de l'orbite étant lié à des problèmes de la théorie des groupes, nous pourrions éventuellement profiter de ces résultats pour obtenir des méthodes de réduction efficaces.

Une deuxième amélioration serait d'utiliser des algorithmes de recherche avec retour arrière. Ces méthodes sont générales puisqu'elles s'appliquent à tous les groupes de permutation. Naturellement, cela se fait au détriment de l'efficacité. Néanmoins, des optimisations de la recherche sont possibles, qui consistent à élaguer l'arbre de recherche au plus tôt. Peut être que certaines architectures de système pourraient donner des hypothèses nous évitant l'exploration de certaines branches et ainsi obtenir une réduction efficace.

Logiques temporelles Notre méthode de réduction pourrait être étendue à d'autres classes de propriétés. Nous nous sommes concentrés sur les propriétés d'accessibilité.

Notre construction s'applique aujourd'hui à l'abstraction des classes dites linéaires qui préserve LTL. Nous pourrions exploiter nos travaux afin de vérifier des formules de LTL.

De plus, TINA propose une abstraction dite des classes atomiques qui préserve la logique CTL. Nous conjecturons que notre méthode de réduction peut s'étendre à cette abstraction. Il serait alors possible de faire évoluer notre méthode de réduction pour qu'elle permette de vérifier des formules de CTL.

Symétries de données Notre approche est basée sur l'exploitation des symétries structurelles d'un système pour les TPN. Elle permet de réduire des systèmes temporisés. Des travaux existent sur l'exploitation des symétries de données. Nous pourrions étendre notre méthode pour y inclure les symétries de données et, ainsi, proposer une méthode de réduction pour les modèles FIACRE.

Bibliographie

[1] Pierre-Alain Bourdil, Bernard Berthomieu, and Eric Jenn. Model-checking real-time properties of an auto flight control system function. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 120–123. IEEE, 2014.

[2] Pierre-Alain Bourdil, Bernard Berthomieu, Silvano Dal Zilio, and François Vernadat. Symmetry reduced state classes for time petri nets. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1751–1758. ACM, 2015.

[3] Edmund M Clarke and E Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*. Springer, 1982.

[4] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on Programming*, pages 337–351. Springer, 1982.

[5] Philippe Schnoebelen, Béatrice Bérard, Michel Bidoit, and François Laroussinie. *Vérification de logiciels: techniques et outils du model-checking*. Vuibert, 1999.

[6] Rodrigo Tacla Saad. *Parallel model checking for multiprocessor architecture*. PhD thesis, INSA de Toulouse, 2011.

[7] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium in Logic in Computer Science (LICS)*, pages 322–331. IEEE Computer Society, 1986.

[8] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[9] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[10] Markus Müller-Olm, David Schmidt, and Bernhard Steffen. Model-checking. In *Static Analysis*, pages 330–354. Springer, 1999.

[11] Pierre-Alain Reynier. *Vérification de systèmes temporisés et distribués: modèles, algorithmes et implémentabilité*. PhD thesis, École Normale Supérieure de Cachan, 2007.

[12] Rajeev Alur and David Dill. Automata for modeling real-time systems. In *Automata, languages and programming*, pages 322–335. Springer, 1990.

[13] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[14] Carl A Petri. Communication with automata: Volume 1 supplement 1. Technical report, DTIC Document, 1966.

[15] Philip M Merlin and David J Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, 1976.

[16] Bernard Berthomieu, PO Ribet, and François Vernadat. L'outil tina–construction d'espaces d'états abstraits pour les réseaux de petri et réseaux temporels. *Proc. Modélisation des Systèmes Réactifs, Metz, France*, 2003.

[17] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In *Proceedings IFIP*, pages 41–46. Elsevier Science Publishers, 1983.

[18] B. Berthomieu and F. Vernadat. State class constructions for branching analysis of Time Petri Nets. In *TACAS2003*, volume LNCS2619, page 442. Springer verlag, 2003.

[19] H. Boucheneb and J. Mullins. Analyse des Réseaux Temporels: Calcul des Classes en O(n[2]) et des Temps de Chemin en O(m×n). *TSI. Technique et science informatiques*, 22(4):435–459, 2003.

[20] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 10 20 states and beyond. In *Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 428–439. IEEE, 1990.

[21] Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[22] Beate Bollig and Ingo Wegener. Improving the variable ordering of obdds is np-complete. *Computers, IEEE Transactions on*, 45(9):993–1002, 1996.

[23] Kenneth L McMillan. *Symbolic model checking*. Springer, 1993.

[24] Gerd Behrmann, Kim G Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Efficient timed reachability analysis using clock difference diagrams. In *Computer aided verification*, pages 341–353. Springer, 1999.

[25] Farn Wang. Symbolic verification of complex real-time systems with clock-restriction diagram. In *Formal Techniques for Networked and Distributed Systems*, pages 235–250. Springer, 2002.

[26] Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, pages 491–515. Springer, 1991.

[27] Patrice Godefroid, J van Leeuwen, J Hartmanis, G Goos, and Pierre Wolper. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer Heidelberg, 1996.

[28] Doron Peled. Combining partial order reductions with on-the-fly model-checking. In *Computer aided verification*, pages 377–390. Springer, 1994.

[29] François Vernadat, Pierre Azéma, and François Michel. Covering step graph. In *Application and theory of Petri nets 1996*, pages 516–535. Springer, 1996.

[30] P-O. Ribet, F. Vernadat, and B. Berthomieu. On combining the persistent sets method with the covering steps graph method. In *Proc. of FORTE 2002, Springer LNCS 2529*, pages 344–359, 2002.

[31] J. Rosen. *Symmetry discovered: concepts and applications in nature and science*. Dover Pubns, 1975.

[32] Khalil Ajami, Serge Haddad, and J-M Ilie. Exploiting symmetry in linear time temporal logic model checking: One step beyond. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 52–67. Springer, 1998.

[33] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 15 July 2004.

[34] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants*. Masson, Paris, 1974.

[35] J-C. Fernandez, H. Garavel, R. Mateescu, L. Mounier, and M. Sighireanu. Cadp, a protocol validation and verification toolbox. In *8th Conf. Computer-Aided Verification, Springer LNCS 1102*, pages 437–440, 1996.

[36] S. Chaki, M E, Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *4th Int. Conf. on Integrated Formal Methods, Springer LNCS 2999*, pages 128–147, 2004.

[37] P. Gastin and D. Oddoux. Fast ltl to buchi automata translation. In *13th Conference Computer-Aided Verification, CAV'2001, Springer LNCS 2102*, pages 53–65, jul 2001.

[38] Patrick Farail, Pierre Gaufillet, Florent Peres, Jean-Paul Bodeveix, Mamoun Filali, Bernard Berthomieu, Rodrigo Saad, François Vernadat, Hubert Garavel, and Frédéric Lang. FIACRE: an intermediate language for model verification in the TOPCASED environment. In *European Congress on Embedded Real-Time Software (ERTS), Toulouse, 29/01/2008-01/02/2008*, http://www.see.asso.fr, janvier 2008. SEE.

[39] P.M. Merlin and David J. Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, Sep 1976.

[40] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.

[41] Hubert Garavel. On the introduction of gate typing in E-LOTOS. In Piotr Dembinski and Marek Sredniawa, editors, *Proceedings of the 15th IFIP International Workshop on Protocol Specification, Testing and Verification (Warsaw, Poland)*. IFIP, Chapman & Hall, Ltd., June 1995.

[42] Mihaela Sighireanu. LOTOS NT user's manual (version 2.1). INRIA projet VASY., November 2000.

[43] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM*, pages 3–12, 2006.

[44] George S. Avrunin Matthew B. Dwyer and James C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE '99 – Proceedings of the 21st international conference on Software engineering ACM New York, NY, USA, 1999*, 1999.

[45] Silvano Dal Zilio, Nouha Abid, and Bernard Berthomieu. Who Checks the Model-Checkers ? Technical Report 12367, LAAS, July 2012.

[46] John Rushby. New Challenges In Certification For Aircraft Software.

[47] SAE ARP. 4761. *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*, 12, 1996.

[48] SAE ARP. 4754. *Certification considerations for highly-integrated or complex aircraft systems*, 1996.

[49] RTCA Do. 178b: Software considerations in airborne systems and equipment certification. *December, 1st*, 1992.

[50] RTCA Do. Design assurance guidance for airborne electronic hardware. *December, 1st*, 1992.

[51] Michaël Lauer. *Une méthode globale pour la vérification d'exigences temps réel: application à l'Avionique Modulaire Intégrée*. PhD thesis, Institut National Polytechnique de Toulouse-INPT, 2012.

[52] Aeronautical Radio Inc. *ARINC 653 : Avionics Application Software Standard Interface*. 1997.

[53] Aeronautical Radio Inc. *ARINC 664 : Aircraft Data Network Part 7 : "Avionics Full Duplex Switched Ethernet (AFDX) Network"*. 1997.

[54] Albert Benveniste, Paul Caspi, Stephen A Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert De Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[55] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. springer, 2004.

[56] Sam Owre, John M Rushby, and Natarajan Shankar. Pvs: A prototype verification system. In *Automated Deduction—CADE-11*, pages 748–752. Springer, 1992.

[57] Lawrence C Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer Science & Business Media, 1994.

[58] P COUSOT. Interprétation abstraite. *TSI. Technique et science informatiques*, 19(1-3):155–164, 2000.

[59] Patrick Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. PhD thesis, Institut National Polytechnique de Grenoble-INPG ; Université Joseph-Fourier-Grenoble I, 1978.

[60] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[61] Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. Formal verification of avionics software products. In *FM 2009: Formal Methods*, pages 532–546. Springer, 2009.

[62] Famantanantsoa Randimbivololona, Jean Souyris, Patrick Baudin, Anne Pacalet, Jacques Raguideau, and Dominique Schoen. Applying formal proof techniques to avionics software: A pragmatic approach. In *FM99 - Formal Methods*, pages 1798–1815. Springer, 1999.

[63] Patrick Cousot, Radhia Cousot, Jerôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. The astrée analyzer. In *Programming Languages and Systems*, pages 21–30. Springer, 2005.

[64] Jean Souyris, Erwan Le Pavec, Guillaume Himbert, Guillaume Borios, Victor Jégu, and Reinhold Heckmann. Computing the worst case execution time of an avionics program by abstract interpretation. In *OASIcs-OpenAccess Series in Informatics*, volume 1. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[65] Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park, and Flavio Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, 2003.

[66] John Penix, Willem Visser, Eric Engstrom, Aaron Larson, and Nicholas Weininger. Verification of time partitioning in the deos scheduler kernel. In *Proceedings of the 22nd international conference on Software engineering*, pages 488–497. ACM, 2000.

[67] Gerard J. Holzmann. Mars code. *Commun. ACM*, 57(2):64–73, February 2014.

[68] Klaus Havelund, Mike Lowry, SeungJoon Park, Charles Pecheur, John Penix, Willem Visser, Jonathan White, et al. Formal analysis of the remote agent before and after flight. In *Proceedings of the 5th NASA Langley Formal Methods Workshop*, volume 134, 2000.

[69] Guy L Steele. *Common LISP: the language*. Digital press, 1990.

[70] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K Rajamani. Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In *Integrated formal methods*, pages 1–20. Springer, 2004.

[71] Thomas Bochot. *Vérification par Model Checking des Commandes de Vol : Applicabilité Industrielle et Analyse de Contre-Exemples*. PhD thesis, Doctorat de l'université de Toulouse.

[72] M Whalen, J Innis, S Miller, and L Wagner. Adgs-2100 adaptive display & guidance system window manager analysis. *NASA Contractor Report*, 213952, 2006.

[73] Michael Whalen, Darren Cofer, Steven Miller, Bruce H Krogh, and Walter Storm. Integration of formal analysis into a model-based software development process. In *Formal Methods for Industrial Critical Systems*, pages 68–84. Springer, 2008.

[74] Thomas Bochot, Pierre Virelizier, Hélene Waeselynck, and Virginie Wiels. Model checking flight control systems: The airbus experience. *ICSE Companion*, 2009:18–27, 2009.

[75] Steven P Miller, Alan C Tribble, and Mats PE Heimdahl. Proving the shalls. In *FME 2003: Formal Methods*, pages 75–93. Springer, 2003.

[76] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. From informal requirements to property-driven formal validation. *Formal Methods for Industrial Critical Systems*, pages 166–181, 2009.

[77] Thomas Ball, Mayur Naik, and Sriram K Rajamani. From symptom to cause: localizing errors in counterexample traces. In *ACM SIGPLAN Notices*, volume 38, pages 97–105. ACM, 2003.

[78] Ning Ge, Marc Pantel, and Xavier Crégut. Automated failure analysis in model checking based on data mining. In *Model and Data Engineering*, pages 13–28. Springer, 2014.

[79] Pascal Traverse, Isabelle Lacaze, and Jean Souyris. Airbus fly-by-wire: A total approach to dependability. In *Building the Information Society*, pages 191–212. Springer, 2004.

[80] Alessandro Fantechi and Stefania Gnesi. On the adoption of model checking in safety-related software industry. In *Computer Safety, Reliability, and Security*, pages 383–396. Springer, 2011.

[81] Abdoulaye Gamatié and Thierry Gautier. Synchronous modeling of avionics applications using the signal language. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 144–151. IEEE, 2003.

[82] Aeronautical Radio Inc. *ARINC specification 429-ALL: Mark 33 Digital Information Transfer System (DITS) Parts 1, 2, 3*. 2001.

[83] Hussein Charara, J Scharbarg, Jerome Ermont, and Christian Fraboul. Methods for bounding end-to-end delays on an afdx network. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10–pp. IEEE.

[84] Peter J Cameron. *Permutation groups*, volume 45. Cambridge University Press, 1999.

[85] Alastair Donaldson and Alice Miller. On the constructive orbit problem. *Annals of Mathematics and Artificial Intelligence*, 57:1–35, 2009.

[86] Á. Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003.

[87] Charles C Sims. Computational methods in the study of permutation groups. In *Computational problems in abstract algebra*, pages 169–183. Pergamon Press, 1970.

[88] Alexander Hulpke. Notes on computational group theory, 2010.

[89] C Norris Ip and David L Dill. Verifying systems with replicated components in murϕ. In *Computer aided verification*, pages 147–158. Springer, 1996.

[90] Giovanni Chiola. Manual and automatic exploitation of symmetries in SPN models. In *Application and Theory of Petri Nets 1998*, pages 28–43. Springer, 1998.

[91] A. Prasad Sistla, Viktor Gyuris, and E. Allen Emerson. Smc: a symmetry-based model checker for verification of safety and liveness properties. *ACM Trans. Softw. Eng. Methodol.*, 9(2):133–166, April 2000.

[92] Karsten Schmidt. LOLA – A low level analyser. In *Application and Theory of Petri Nets 2000*, pages 465–474. Springer, 2000.

[93] M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. W. Vaandrager. Adding symmetry reduction to Uppaal. In *Formal Modeling and Analysis of Timed Systems*, pages 46–59. Springer LNCS 2791, 2004.

[94] Tommi Junttila. On the symmetry reduction method for petri nets and similar formalisms. Research Report A80, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, September 2003.

[95] Martijn Hendriks. *Enhancing uppaal by exploiting symmetry*. Nijmegen Institute for Computing and Information Sciences, Faculty of Science, University of Nijmegen, 2002.

[96] Peter H Starke. Reachability analysis of petri nets using symmetries. *Systems Analysis Modelling Simulation*, 8(4-5):293–303, 1991.

[97] T.A. Junttila. Computational complexity of the place/transition-net symmetry reduction method. *Journal of Universal Computer Science*, 7(4):307–326, 2001.

[98] Peter Huber, Arne M Jensen, Leif O Jepsen, and Kurt Jensen. *Towards reachability trees for high-level Petri nets*. Springer, 1985.

[99] Kurt Jensen. Condensed state spaces for symmetrical coloured petri nets. *Formal Methods in System Design*, 9(1-2):7–40, 1996.

[100] C. Norris IP and David L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9:41–75, 1996.

[101] Dragan Bosnacki, Dennis Dams, and Leszek Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer (STTT)*, 4:92–106, 2002.

[102] Yann Thierry-Mieg, Béatrice Bérard, Fabrice Kordon, Didier Lime, and Olivier H Roux. Compositional analysis of discrete time petri nets. In *1st workshop on Petri Nets Compositions (CompoNet 2011)*, volume 726, pages 17–31.

[103] Louise Elgaard. The symmetry method for coloured petri nets. *DAIMI Report Series*, 31(564), 2002.

[104] Giovanni Chiola, Claude Dutheillet, Giuliana Franceschinis, and Serge Haddad. On well-formed coloured nets and their symbolic reachability graph. In *High-level Petri Nets*, pages 373–396. Springer, 1991.

[105] Gerard J Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[106] A. Donaldson and A. Miller. Automatic symmetry detection for model checking using computational group theory. In John Fitzgerald, Ian Hayes, and Andrzej Tarlecki, editors, *FM 2005: Formal Methods*, volume 3582 of *Lecture Notes in Computer Science*, pages 631–631. Springer Berlin / Heidelberg, 2005. 10.1007/11526841-32.

[107] A. Donaldson, A. Miller, and M. Calder. Finding symmetry in models of concurrent systems by static channel diagram analysis. In *AVoCS'04, ENCTS 128(6)*, 2005.

[108] A. Donaldson and A. Miller. Exact and approximate strategies for symmetry reduction in model checking. In *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 541–556. Springer Berlin / Heidelberg, 2006.

[109] E. Clarke, E. Emerson, S. Jha, and A. Sistla. Symmetry reductions in model checking. In Alan Hu and Moshe Vardi, editors, *Computer Aided Verification*, pages 147–158. Springer LNCS 1427, 1998.

[110] Karsten Schmidt. How to calculate symmetries of petri nets. *Acta Inf.*, 36(7):545–590, January 2000.

[111] Karsten Schmidt. Integrating low level symmetries into reachability analysis. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 315–330, 2000.

[112] M AdelsonVelskii and Evgenii Mikhailovich Landis. An algorithm for the organization of information. Technical report, DTIC Document, 1963.

[113] TommiA. Junttila. New canonical representative marking algorithms for place/transition-nets. In Jordi Cortadella and Wolfgang Reisig, editors, *Applications and Theory of Petri Nets 2004*, volume 3099 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.

[114] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9:77–104, 1996. 10.1007/BF00625969.

[115] T.A. Junttila. New orbit algorithms for data symmetries. In *Application of Concurrency to System Design, 2004. ACSD 2004. Proceedings. Fourth International Conference on*, pages 175 – 184, june 2004.

[116] L. Lorentsen and L.M. Kristensen. Exploiting stabilizers and parallelism in state space generation with the symmetry method. In *Application of Concurrency to System Design, 2001. Proceedings. 2001 International Conference on*, pages 211 – 220, 2001.

[117] Somesh Jha. *Symmetry and induction in model checking*. PhD thesis, Carnegie Mellon University, 1996.

[118] E. Emerson and Thomas Wahl. Dynamic symmetry reduction. In Nicolas Halbwachs and Lenore Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *Lecture Notes in Computer Science*, pages 382–396. Springer Berlin / Heidelberg, 2005. 10.1007/978-3-540-31980-1-25.

[119] Alaistair F. Donaldson. *automatic technic for detecting and exploiting symmetry in model-checking*. PhD thesis, university of glasgow, 2007.

[120] Christoph M Hoffmann. *Group-theoretic algorithms and graph isomorphism*, volume 136. Springer Heidelberg, 1982.

[121] Bernard Berthomieu, Florent Peres, and François Vernadat. Bridging the gap between timed automata and bounded time petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 82–97. Springer, 2006.

[122] Hartmann J Genrich. Predicate/transition nets. In *Petri Nets: Central Models and Their Properties*, pages 207–247. Springer, 1987.

[123] Štefko Miklavic. Phd course algebraic combinatorics, computability and complexity in tempus project see doctoral studies in mathematical sciences. 2011.

[124] TommiA. Junttila. New canonical representative marking algorithms for place/transition-nets. In *Applications and Theory of Petri Nets 2004*, volume 3099 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004.

[125] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.7.7*, 2015.

[126] Frank Harary. The automorphism group of a hypercube. *Journal of Universal Computer Science*, 6(1):136–138, 2000.

[127] Franck Pommereau. Algebras of coloured petri nets. *LAP LAMBERT Academic Publishing*, 2010.