Chapitre 4
Symétries et logique temporelle

Serge HADDAD , Jean-Michel ILIÉ[1]

1. Introduction

Le modèle des réseaux de Petri bien formés permet de maîtriser la complexité des systèmes durant les phases de conception, de vérification et d'évaluation. Une de ses méthodes de vérification, la construction du graphe symbolique a de nombreuses applications [DIA 01]. Il fournit entre autres une représentation très condensée de l'espace d'états et dispose d'algorithmes de décision de certaines propriétés standard des réseaux de Petri.

Pour d'autres propriétés, on ne peut vérifier sur ce graphe que des conditions suffisantes ou nécessaires (e.g. la vivacité). Or le modélisateur souhaite le plus souvent tester des propriétés spécifiques de son système. Ces propriétés sont exprimées à l'aide d'un langage de formules comme ceux des différentes logiques temporelles [EME 90]. Dans ce cas, la construction du graphe symbolique n'est d'aucun secours. Il s'agit donc ici de présenter une extension de cette méthode applicable aux formules de logique temporelle.

Quelques travaux ont déjà porté sur l'exploitation de la symétrie pour diminuer la complexité de la vérification. Ces travaux s'appliquent à des systèmes symétriques d'un pouvoir d'expression plus restreint que celui des réseaux bien formés. Mais comme nous allons maintenant le détailler, leur véritable limite se situe dans la prise en compte de la symétrie des formules de logique temporelle.

[1]LAMSADE, Université Paris Dauphine, Place du Maréchal de Lattre de Tassigny, 75775 Paris cedex 16, France, (serge.haddad@lamsade.dauphine.fr)
LIP6, Université Pierre et Marie Curie, 75252 Paris cedex 05, France, (jean-michel.ilie@lip6.fr),

Nous nous restreindrons ici à la logique temporelle linéaire dont nous rappelons brièvement l'algorithme usuel de vérification (voir le premier chapitre ou encore [GER 93]) :

- La négation de la formule est traduite en automate de Büchi.
- On construit le produit synchronisé de l'automate et du graphe d'état du modèle (produit des états et des transitions compatibles).
- On recherche dans ce produit un chemin qui se termine par une boucle comprenant un état associé à un des états de succès de l'automate de Büchi. Nous appellerons un tel chemin, *chemin d'invalidation*.

La formule est valide si et seulement si un tel chemin n'existe pas. Bien que la taille de l'automate de Büchi soit exponentiellement plus grande que celle de la formule, cette taille reste très minime en regard du nombre d'états du sytème. Par conséquent dans la pratique, ce dernier paramètre conditionne la complexité de la vérification.

Dans une première approche [CLA 96], on restreint les propositions atomiques du langage à des propositions symétriques. Par exemple, des formules telles que "*Dans un état futur tous les processus seront au repos*" ou "*Si un quelconque des processus attend la ressource alors un quelconque des processus l'obtiendra*" sont des formules symétriques. Pour de telles formules, on substitue au graphe d'etat un graphe quotient du modèle (équivalent au graphe symbolique) et on applique de manière inchangée le reste de l'algorithme. Cette méthode s'applique directement au réseaux bien formés à l'aide du graphe symbolique. Malheureusement de nombreuses formules intéressantes ne sont pas symétriques au sens de cette définition. Par exemple une formule d'équité telle que "*Si un quelconque des processus attend la ressource alors ce processus l'obtiendra*" n'est pas symétrique.

La deuxième approche [EME 96] définit des automates de Büchi symétriques. A partir de cet automate et du modèle, on construit directement un produit synchronisé *quotient* pour lequel l'existence des chemins d'invalidation est équivalente à l'existence d'un tel chemin dans le produit synchronisé initial. Les formules symétriques de la méthode précédente engendrent des automates symétriques et d'autres formules dont la formule d'équité décrite précédemment deviennent symétriques.

Malheureusement cette nouvelle méthode ne couvre pas le cas de formules partiellement symétriques qui sont considérées comme asymétriques. Tel est le cas par exemple de : "*Si un quelconque des processus attend la ressource et si dans un futur aucun processus d'identité supérieure ne demande plus la ressource alors ce processus l'obtiendra*". Aussi les auteurs de ce chapitre (avec K. Ajami) [AJA 98] ont introduit une méthode reposant sur une définition du graphe quotient du produit synchronisé plus fine que la précédente :

- On détermine des relations d'équivalence entre états de l'automate de Büchi pour chaque élément du groupe des "*symétries*" du modèle telles

que deux états sont équivalents si ils induisent le même présent et futur à l'action de cette symétrie près.
– Le graphe quotient est construit en mettant en relation des couples (état du modèle, état de l'automate) à partir des relations d'équivalence de l'étape précédente.

Dans le cas d'un automate symétrique, on obtient le même graphe quotient mais la méthode permet de réduire la complexité de la vérification pour des automates partiellement symétriques. Autrement dit, cette méthode généralise les deux précédentes. Remarquant que les relations d'équivalence de l'automate sont parfois en nombre exponentiel, on se limite dans la pratique à un sous-ensemble de relations de taille polynomiale qui produit un graphe quotient plus large. Cependant le temps de calcul global est souvent plus faible que si l'on avait déterminé toutes les relations d'équivalence.

Des cas pratiques de modélisation mettent en évidence que cette troisième méthode (et donc aussi les deux autres) reste impuissante à réduire la complexité de la vérification dans un cas relativement fréquent pour les formules asymétriques : les propositions de la plupart des états de l'automate de Büchi possèdent des symétries mais l'automate est globalement asymétrique (i.e. les relations d'équivalence entre états sont presque toutes réduites à l'identité).

Dans la section suivante, nous décrivons le principe d'une méthode qui exploite **uniquement les symétries des propositions atomiques de chaque état de l'automate indépendamment de la structure de graphe de cet automate** [HAD 00]. Puis dans la troisième section, nous l'illustrons sur un exemple de réseau de Petri bien formé. On y montre à la fois comment l'adaptation de la construction du graphe symbolique fournit une réalisation efficace de la méthode et comment l'asymétrie du modèle se traite de façon analogue. Ce point est d'ailleurs très important car dans la pratique les modèles sont très souvent partiellement symétriques.

Enfin dans la section finale, nous discutons de la possibilité de combiner les deux dernières méthodes ainsi que d'y associer des améliorations proposées par d'autres chercheurs [EME 99].

2. Principes de la méthode

2.1. *Vérification de structures de Kripke*

Afin de présenter la méthode dans toute sa généralité, nous nous plaçons au niveau sémantique c'est à dire au niveau de la structure de Kripke engendrée par le modèle syntaxique quelqu'il soit. Nous rappelons ici certaines définitions

du chapitre 1 en les particularisant pour rendre possible l'application de cette technique de vérification.

Définition 1 *Une structure de Kripke (finie)* $SK = \langle AP, S, \rightarrow, \nu \rangle$ *est définie par :*

- AP *est un ensemble de propositions atomiques*
- S *est un ensemble (fini) d'états*
- S_0*, le sous-ensemble des états initiaux*
- $\rightarrow$ *est une relation binaire* $\subset S \times S$
- $\nu : S \rightarrow 2^{AP}$ *est une fonction injective qui associe à chaque état* s*,* $\nu(s)$ *l'ensemble des propositions atomiques vérifiées en* s*.*

L'injectivité de la fonction ν signifie qu'un état est entièrement caractérisé par la valeur des propositions atomiques. Les structures issues des modèles les plus répandus vérifient toutes cette contrainte. D'autre part, AP est clos par négation : si P est une proposition atomique alors **NOT** P est une proposition atomique (avec la simplification **NOT NOT** $P \equiv P$). De plus, nous considérons systématiquement que ces structures sont initialisées. Nous nous focalisons sur la vérification de formules sur les états d'où l'absence d'information sur les arcs. La démarche s'adapterait sans difficulté aux formules sur les événements (i.e. les structures de Kripke étiquetées). Le résultat final de la méthode (proposition 9) est valable pour les structures infinies *à branchement fini* (i.e. tels que le nombre d'états initiaux soit fini et que chaque état ait un nombre fini de successeurs).

Définition 2 *Une structure de Kripke* $SK = \langle AP, S, \rightarrow, \nu \rangle$ *est dite à branchement fini ssi :*

- S_0*, le sous-ensemble des états initiaux est fini*
- $\forall s \in S$*,* $\{s' | s \rightarrow s'\}$ *est un ensemble fini. Autrement dit, chaque état a un nombre fini de successeurs.*

Une formule de logique temporelle linéaire peut être spécifiée à l'aide du langage LTL [PNU 77] ou du μ-calcul [KOZ 83]. Cependant lors de la phase de vérification, cette formule est le plus souvent traduite en automate de Büchi [VAR 96] (ou une des nombreuses variantes possibles).

Définition 3 *Un automate de Büchi* $B = \langle AP, Q, Q_0, \rightarrow, F, \nu \rangle$ *est défini par :*

- AP *un ensemble fini de propositions atomiques*
- Q *un ensemble fini d'états tels que pour* $q \in Q$*,* $\nu(q) \subset AP$ *désigne les propositions de l'état* q
- $Q_0 \subset Q$ *le sous-ensemble des états initiaux*
- $\rightarrow$ *est le relation de succession* $\subset Q \times Q$
- $F \subset Q$ *un sous-ensemble d'états de succès*

Rappelons succinctement la notion de satisfaction d'une formule (spécifiée par un automate) par une séquence infinie de la structure de Kripke. Les séquences infinies de la structure $\{s_i\}_{i=0..\infty}$ avec $s_0 \in S_0$ qui satisfont la formule caractérisée par l'automate de Büchi doivent être associées à un chemin infini dans l'automate $\{q_i\}_{i=0..\infty}$ avec $q_0 \in Q_0$. De plus les propriétés atomiques requises pour l'état q_i doivent être présentes dans l'état s_i : $\nu(q_i) \subset \nu(s_i)$. Enfin le chemin de l'automate doit rencontrer une infinité de fois le sous-ensemble F. Ceci nous conduit au concept clef de produit synchronisé.

Définition 4 *Soit SK une structure de Kripke et B un automate de Büchi, le produit synchronisé de SK et B est un graphe $Gr(SK, B) = (V, V_0, \rightarrow)$ défini par :*

- $V = \{(s, q) \mid \nu(q) \subset \nu(s)\}$, *l'ensemble des noeuds*
- $V_0 = \{(s, q) \in V \mid s \in S_0 \wedge q \in Q_0\}$, *l'ensemble des états initiaux*
- $\rightarrow \subset V \times V$, *la relation de succession entre noeuds est telle que* $(s, q) \rightarrow (s', q')$ *ssi* $s \rightarrow s'$ *et* $q \rightarrow q'$

Comme indiqué au chapitre 1, le principe de vérification consiste à traduire la négation de la formule en automate de Büchi et de rechercher une séquence de la structure acceptée par un chemin de l'automate de Büchi.

Définition 5 *Soit SK une structure de Kripke et B un automate de Büchi, B est satisfaisable par SK ssi il existe un chemin infini dans le produit synchronisé $Gr(SK, B)$ partant d'un noeud initial et rencontrant une infinité de fois le sous-ensemble de noeuds $\{(s, q) \mid q \in F\}$*

Si la structure est finie, cela revient à trouver un chemin élémentaire se terminant par un état déjà rencontré tel qu'entre ces deux occurences, on rencontre un état dont la deuxième composante est un état de succès. De nombreuses techniques ont été élaborées pour une recherche efficace de l'existence d'un tel chemin. Elles sont bien entendu applicables au graphe quotient que nous allons définir dans la section 2.3.

2.2. *Structures de Kripke symétriques*

Pour définir les symétries d'une structure, nous nous intéressons aux propositions atomiques qui caractérisent les états. Afin d'illustrer de manière concrète ce qui suit, nous nous référerons à un réseau de Petri bien formé [DIA 01] constitué d'une classe de couleurs $C = \{u, v, w, x\}$ qui n'est pas décomposée en sous-classes statiques. Nous commençons par rappeler quelques notions élémentaires sur les groupes [LAN 77].

Définition 6 *Soit G un groupe, d'élément neutre id et dont l'opération est notée $(\bullet)$.*

- *Soit E un ensemble, une action de G sur E est une fonction de $G \times E$ vers E telle que l'image de (g, e), notée par $g.e$, vérifie :*

$$\forall e \in E \; id.e = e \;\; \forall g, g' \in G \; (g \bullet g').e = g.(g'.e)$$

- *Le (sous-)groupe d'isotropie d'un élément e est défini par :*

$$G_e = \{g \mid g.e = e\}$$

- *Soit H un sous-groupe de G, l'orbite de e sous l'action de H, notée $H.e$, est définie par :*

$$\{g.e \mid g \in H\}$$

- *Cette action s'etend naturellement aux parties de E par :*

$$g.E' = \{g.e \mid e \in E'\}$$

Supposons que E soit l'ensemble des propositions atomiques du réseau, alors une proposition $[p(u) = 1]$ signifie que la place p contient exactement un jeton de couleur u et éventuellement des jetons de couleurs différentes. Si G est le groupe des permutations de C et si g est la permutation qui inverse u et v alors $g.[p(u) = 1] = [p(v) = 1]$. Soit $\{[p(u) = 1], [p(v) = 1]\}$ un sous-ensemble de propositions atomiques, alors le groupe d'isotropie de ce sous-ensemble est le sous-groupe des permutations qui laissent globalement invariant le sous-ensemble $\{u, v\}$.

Nous sommes en mesure de caractériser une structure de Kripke symétrique.

Définition 7 *Soit SK une structure de Kripke et G un groupe qui agit sur AP, SK est symétrique (par rapport à G) ssi :*

- *Tout état a un "symétrique" par rapport à un élément quelconque de G :*

$$\forall s \in S, \forall g \in G, \; \exists s' \in S, \nu(s') = g.\nu(s)$$

  On étend l'action du groupe aux états en notant par $g.s$ l'unique s' de la formule précédente.
- *L'ensemble des états initiaux est invariant sous l'action de G : $G.S_0 = S_0$.*
- *La relation de succession est une congruence pour l'action de G :*

$$\forall s, s' \in S, \; \forall g \in G \; s \to s' \Leftrightarrow g.s \to g.s'$$

En se reportant à [DIA 01], le lecteur pourra vérifier que ces conditions sont vérifiées pour un réseau de Petri bien formé et son groupe de permutations admissibles.

2.3. *Vérification de structure de Kripke symétriques*

Soit q un état de l'automate de Büchi B, le sous-groupe d'isotropie de l'ensemble des propriétés atomiques de q, $G_{\nu(q)}$ sera plus simplement noté G_q bien que nous ne définissions pas d'action de G sur les états de l'automate. Remarquons que le cardinal du sous-groupe G_q est fortement lié au degré de symétrie de l'ensemble des propriétés atomiques de q indépendamment de la structure de l'automate de Büchi.

Plaçons-nous dans le contexte du réseau de Petri bien formé pour étudier les différentes situations. Si ce sous-groupe est égal au groupe G, l'état est entièrement symétrique, comme pour l'ensemble $\{[p(u) = 1], [p(v) = 1], [p(w) = 1], [p(x) = 1]\}$. Si par contre ce sous-groupe est réduit à $\{id\}$, l'état est complètement asymétrique comme pour l'ensemble $\{[p(u) = 1], [p(v) = 2], [p(w) = 3], [p(x) = 4]\}$. Dans la plupart des cas, ce sous-groupe est non trivial (i.e. différent de $\{id\}$ et de G). Plus généralement dans un réseau de Petri bien formé, le groupe d'isotropie d'un état de l'automate est donné de manière implicite par une partition de l'ensemble de couleurs telle que ce groupe est exactement le sous-groupe des permutations qui laisse globalement invariant chaque sous-ensemble de la partition. Pour $\{[p(u) = 1], [p(v) = 1]\}$, la partition est donnée par $\{\{u, v\}, \{w, x\}\}$. On appelle cette partition, *la partition locale de l'état.*

Notre objectif est la construction d'un produit synchronisé quotient noté $GRQ(SK, B)$ sur lequel on effectuera la recherche du chemin invalidant.

Dans cette structure quotient, chaque noeud est de la forme (H, O, q) où H est un sous-groupe de G, $O \subset S$ et $q \in Q$. De plus, les deux contraintes suivantes doivent être satisfaites :

(C1) $\forall s \in O, \nu(q) \subset \nu(s)$

(C2) $H.O = O$

L'idée intuitive est que ce noeud factorise l'ensemble des noeuds du produit synchronisé $\{(s, q)\}_{s \in O}$. On dira dans la suite que (s, q) est représenté par (H, O, q). Comme nous allons le voir, H est utilisé dans la définition de la relation de succession.

Illustrons cette notion de noeud symbolique sur un réseau de Petri bien formé. Par exemple, H sera implicitement défini par la partition $\{\{u\}, \{v\}, \{w, x\}\}$ et O sera simplement un marquage symbolique (noté dans la suite du paragraphe $\widehat{m}$) associé à la classe C temporairement décomposée en sous-classes statiques correspondant à la partition. Dans notre exemple, les trois sous-classes statiques sont $\{u\}$, $\{v\}$ et $\{w, x\}$. On remarquera que cette partition locale d'une classe de couleurs est analogue au découpage en *sous-classes statiques* à ceci près qu'elle est locale à l'état du produit synchronisé quotient. Aussi

nous appellerons les éléments d'une partition locale de couleurs : *sous classes-statiques locales*.

La construction démarre avec l'ensemble des noeuds initiaux de la forme : $(G_{q_0}, G_{q_0}.s_0, q_0)$ avec $s_0 \in S_0$, $q_0 \in Q_0$ et $\nu(q_0) \subset \nu(s_0)$. Remarquons que la condition $C1$ est satisfaite en raison de la définition de G_{q_0} et que la condition $C2$ découle immédiatement du fait que G_{q_0} est un groupe.

Il nous suffit maintenant de définir la relation successeur. (H_1, O_1, q_1) admet (H_2, O_2, q_2) pour successeur, ce qu'on note $(H_1, O_1, q_1) \rightarrow (H_2, O_2, q_2)$ ssi :

$$q_1 \rightarrow q_2 \text{ et } \exists s_1 \in O_1, \exists s_2 \in S \text{ avec } s_1 \rightarrow s_2 \text{ et } \nu(q_2) \subset \nu(s_2)$$

Dans ces conditions, O_2 et H_2 sont définis par :

$$O_2 = (H_1 \cap G_{q_2}).s_2 \text{ et } H_2 \subset G_{O_2}$$

Cette règle laisse une certaine liberté dans le choix du sous-groupe H_2. Ce choix dépendra du modèle syntaxique et de la représentation d'un noeud symbolique. Idéalement on cherche à maximiser H_2, i.e. $H_2 = G_{O_2}$. Dans le pire des cas, on prendra $H_2 = H_1 \cap G_{q_2}$.

En appliquant ceci au réseau de Petri bien formé, en présence d'un arc $q_1 \rightarrow q_2$ de l'automate on tentera de générer un successeur à (H_1, O_1, q_1) en plusieurs étapes :

- On raffine la partition statique du marquage symbolique $\widehat{m}_1$ en prenant celle induite par $H_1 \cap G_{q_2}$ ce qui revient à considérer que la nouvelle partition est constituée d'intersections d'un ensemble de la partition associé au marquage symbolique et d'un ensemble de la partition locale associée à q_2.
- On effectue un franchissement symbolique (pour une transition arbitraire) et on vérifie que le marquage symbolique obtenu $\widehat{m}_2$ satisfait les propositions atomiques $\nu(q_2)$. Ceci peut se faire au niveau symbolique puisque la partition a été suffisamment raffinée.
- Toute paire de sous-classes statiques réduites à une unique sous-classe dynamique $D_1 = \{Z_1\}$, $D_2 = \{Z_2\}$ telles que $\widehat{m}_2(Z_1) = \widehat{m}_2(Z_2)$ est transformée en un unique sous-classe statique. Ceci revient à étendre le sous-groupe des permutations à partir de $H_1 \cap G_{q_2}$ en laissant invariant l'ensemble O_2 des marquages ordinaires associé au marquage symbolique.

Il est relativement immédiat que cette construction correspond à la relation successeur définie dans le cas général.

Le lemme qui suit et la proposition 9 démontrent la validité de cette construction.

Lemme 8 *Soient SK une structure de Kripke symétrique et B un automate de Büchi, soient $Gr(SK, B)$ le produit synchronisé et $GRQ(SK, B)$ le produit synchronisé quotient alors :*

- *Tout les états de $Gr(SK,B)$ représentés par un état de $GRQ(SK,B)$ atteint par un prédécesseur, sont atteignables par un état de $Gr(SK,B)$ représenté par ce prédécesseur :*
$$(H_1,O_1,q_1) \rightarrow (H_2,O_2,q_2) \Rightarrow \forall s_2 \in O_2, \exists s_1 \in O_1,\ (s_1,q_1) \rightarrow (s_2,q_2)$$
- *Un chemin (éventuellement infini) de $Gr(SK,B)$ fournit un chemin (éventuellement infini) dans $GRQ(SK,B)$:*
$$(s_0,q_0) \rightarrow (s_1,q_1) \rightarrow \ldots \rightarrow (s_n,q_n) \rightarrow \ldots$$
$$\Rightarrow \exists (H_0,O_0,q_0) \rightarrow (H_1,O_1,q_1) \rightarrow \ldots \rightarrow (H_n,O_n,q_n) \rightarrow \ldots$$
avec $\forall i$, $s_i \in O_i$.

Preuve
Démontrons la première affirmation. Appelons $s_1^* \in O_1$ et $s_2^* \in O_2$ les deux états qui justifient l'existence de l'arc dans le produit synchronisé symbolique. On a $s_1^* \rightarrow s_2^*$ et $\nu(q_2) \subset \nu(s_2^*)$. Soit maintenant s_2 un état quelconque de O_2. Par définition de O_2, il existe un $g \in H_1 \cap G_{q_2}$ tel que $s_2 = g.s_2^*$. Puisque $g \in G_{q_2}$, $\nu(q_2) = g.\nu(q_2) \subset g.\nu(s_2^*) = \nu(s_2)$. Posons maintenant $s_1 = g.s_1^*$. Puisque $g \in H_1$, on a $s_1 \in O_1$. Puisque $\rightarrow$ est une congruence, $s_1^* \rightarrow s_2^* \Rightarrow s_1 = g.s_1^* \rightarrow g.s_2^* = s_2$. Ce qui achève la démonstration.

Démontrons la deuxième affirmation par récurrence sur la longueur de la séquence (ceci s'applique évidemment à une séquence infinie). Soit $(s_0,q_0) \in GR(SK,B)$, alors $\nu(q_0) \subset \nu(s_0)$, $s_0 \in S_0$ et $q_0 \in Q_0$. Donc $(G_{q_0}, G_{q_0}.s_0, q_0) \in GRQ(SK,B)$. Supposons le chemin construit pour les n premiers noeuds. On a $s_n \in O_n$, $s_n \rightarrow s_{n+1}$, $\nu(q_{n+1}) \subset \nu(s_{n+1})$ et $q_n \rightarrow q_{n+1}$. En vertu de la définition de la relation de succession dans la structure quotient, $\exists H_{n+1}$ tel que $(H_n,O_n,q_n) \rightarrow (H_{n+1}, (H_n \cap G_{q_{n+1}}).s_{n+1}, q_{n+1})$. Ce qui achève la démonstration. ♢

On remarquera que le premier point de ce lemme n'est plus vrai si on remplace prédécesseur par successeur. D'où la restriction qui suit (toujours vérifiée pour les sytèmes finis) dans la proposition finale.

Proposition 9 *Pour une structure de Kripke symétrique à branchement fini, il existe un chemin invalidant dans le produit synchronisé ssi il existe un chemin invalidant dans le produit synchronisé quotient.*

Preuve
Supposons qu'il existe un chemin invalidant dans le produit synchronisé, alors la deuxième affirmation du lemme précédent produit un chemin dans le produit synchronisé quotient dont la deuxième composante (un chemin dans l'automate) rencontre une infinité d'occurences d'états de succès. C'est donc un chemin invalidant.

Supposons qu'il existe un chemin invalidant dans le produit synchronisé quotient, $(H_0, O_0, q_0) \rightarrow (H_1, O_1, q_1) \rightarrow \ldots \rightarrow (H_n, O_n, q_n) \rightarrow \ldots$. On construit alors un arbre dont les noeuds (excepté la racine) sont des noeuds du produit synchronisé ordinaire. Appelons $\perp$ la racine (i.e. l'unique noeud de hauteur 0). Les noeuds de hauteur $n > 0$ sont exactement les noeuds $\{(s_{n-1}, q_{n-1})\}$ avec $s_{n-1} \in O_{n-1}$. Si $n > 0$, on choisit pour père à un noeud (s_n, q_n), l'un quelconque des noeuds (s_{n-1}, q_{n-1}) associés à la première affirmation du lemme précédent lorsqu'on l'applique à $(H_{n-1}, O_{n-1}, q_{n-1}) \rightarrow (H_n, O_n, q_n)$ et à s_n. Le père des noeuds $\{(s_0, q_0)\}$ est évidemment $\perp$. Puisque la structure est à branchement fini et que l'automate est fini, le produit synchronisé est à branchement fini. Donc cet arbre infini est de degré fini. En vertu de lemme de Koening (voir le chapitre 4 de [DIA 01]), on peut en extraire un chemin infini. Le suffixe, obtenu en supprimant le premier noeud, est un chemin invalidant dans le produit synchronisé. ◊

3. Déroulement de la méthode

Nous allons maintenant détailler la méthode sur un réseau de Petri bien formé issu de la modélisation d'un algorithme réparti.

3.1. *Présentation du modèle*

3.1.1. *Description informelle de l'algorithme*

Cet algorithme a pour objectif la réalisation d'un service de rendez-vous bipoint symétrique utilisable par une application répartie sur plusieurs sites [BAG 89]. L'interface entre le service et l'application est une primitive comprenant comme unique paramètre un ensemble de sites. Un appel à cette primitive signifie que l'application désire un rendez-vous avec l'un quelconque des sites choisi dans cet ensemble. Pour qu'un rendez-vous puisse avoir lieu entre deux sites, il est nécessaire que chacun d'entre eux ait appelé cette primitive et que le site partenaire soit inclus dans l'ensemble spécifié. Cet algorithme s'exécute dans un environnement réseau asynchrone et fiable. Cependant les canaux de communication ne sont pas supposés respecter la discipline FIFO.

A titre d'exemple, on pourra imaginer une application client-serveur comprenant plusieurs serveurs redondants assurant le même service qui n'acceptent les requêtes des clients que si celles-ci proviennent de sites prédéterminés. Dans ce cas, un client désirant effectuer une requête demandera un rendez-vous avec l'un quelconque des serveurs et un serveur exécutera une boucle indéfinie où

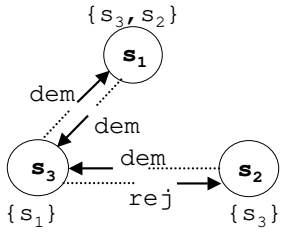

Figure 1. *Un premier exemple de rendez-vous*

chaque tour de boucle débute par un appel à cette primitive en spécifiant les sites autorisés et se poursuit par le traitement de la requête du site dont l'indentité est renvoyée par le service.

Ici nous ne nous intéressons qu'à la prise de rendez-vous et non à l'échange de données qui suit le rendez-vous. En effet, cette deuxième phase ne présente pas de difficultés algorithmiques.

Nous présentons le fonctionnement de l'algorithme à l'aide d'exemples. Lorsqu'une application désire un rendez-vous, la couche service envoie successivement une demande (par un message *dem*) à chaque site jusqu'à ce qu'il ait reçu une réponse favorable (par un message *ack*). Une réponse défavorable se traduit par l'envoi d'un message *rej*.

Dans le premier exemple représenté en figure 1, les trois sites ont à peu près simultanément appelé la primitive de rendez-vous ; s_1 désire un rendez-vous avec s_2 ou s_3, s_3 désire un rendez-vous avec s_1 et s_2 désire un rendez-vous avec s_3. Chaque site envoie alors une demande. Dans le cas de s_1, il y a un choix indéterministe de s_3 comme premier site à contacter.

A la réception de la demande de s_2, s_3 répond défavorablement puisque s_2 ne fait pas partie des sites qui l'intéressent actuellement. Quand s_1 et s_3 reçoivent leur demande respective, ils n'ont ici pas besoin d'acquitter la demande et s'engagent immédiatement dans l'échange de données. Le site s_2 reste pour l'instant bloqué car il a épuisé l'ensemble des candidats possibles. Il ne se débloquera que lors d'une réception d'une demande ultérieure du site s_3.

Le deuxième exemple représenté sur la figure 2 décrit une situation symétrique où chacun des trois sites désire communiquer avec les deux autres. Dans cette exécution, le choix indéterministe d'un candidat conduit le service de s_1 à envoyer une demande de rendez-vous à s_2 tandis que celui-ci envoie sa demande à s_3, ce dernier envoyant sa demande à s_1. A la réception d'une demande, chacun des trois sites se trouve devant un dilemne. Le rendez-vous proposé convient au site mais il a envoyé lui-même une demande à un troisième site. Deux at-

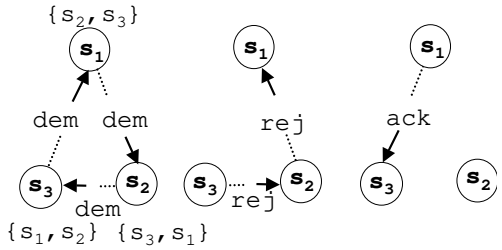

Figure 2. *Exemple de comportement asymétrique*

titudes sont possibles (nous les décrivons pour le site s_2 à la réception de la demande de s_1) :

Comportement 1 Comme s_2 est "engagé" par sa demande à s_3, il rejette la demande du site s_1. Si tous les sites adoptent cette attitude, s_1 et s_3 rejettent aussi les demandes reçues. La situation peut se reproduire avec le second candidat possible de chaque site. Dans ce cas, aucun rendez-vous n'est obtenu alors que plusieurs sont possibles.

Comportement 2 De manière opportuniste, s_2 attend une réponse de s_3 pour rendre sa réponse à s_1 : si s_3 rejette sa requête il accepte celle de s_1 sinon il la rejette. Dans tous les cas de figure, il semble assuré d'un rendez-vous. Malheureusement si les autres sites font de même, tous restent bloqués en attente d'une réponse. On a affaire ici à un *interblocage* de communication.

On se trouve confronté à l'un des paradoxes de l'algorithmique répartie. Pour simplifier la conception de tels algorithme, on recherche la symétrie mais les solutions complètement symétriques ne garantissent pas la progression de l'algorithme. Il faut introduire à faible dose l'asymétrie.

La manière la plus usuelle de le faire consiste à développer un code identique qui conduit à un comportement asymétrique en raison ce qui distingue chaque site, c'est à dire leur identité. Ici si l'identité de l'émetteur d'une demande reçue est supérieure à celle du récepteur alors que celui-ci attend une réponse à sa propre demande, le site choisira de retarder sa réponse. Dans le cas contraire, il rejettera la demande. Cependant, il s'autorisera à réémettre une demande vers le site rejeté, si par hasard la réponse attendue était un rejet.

En appelant At_i la proposition qui indique qu'un site i attend une réponse et $Ret_{i,j}$ la proposition qui indique qu'un site i retarde sa réponse à une demande d'un site j, Le comportement que nous venons de décrire garantit la propriété suivante :

$$\bigwedge_i (At_i \Rightarrow \bigwedge_{j<i} \overline{Ret_{i,j}})$$

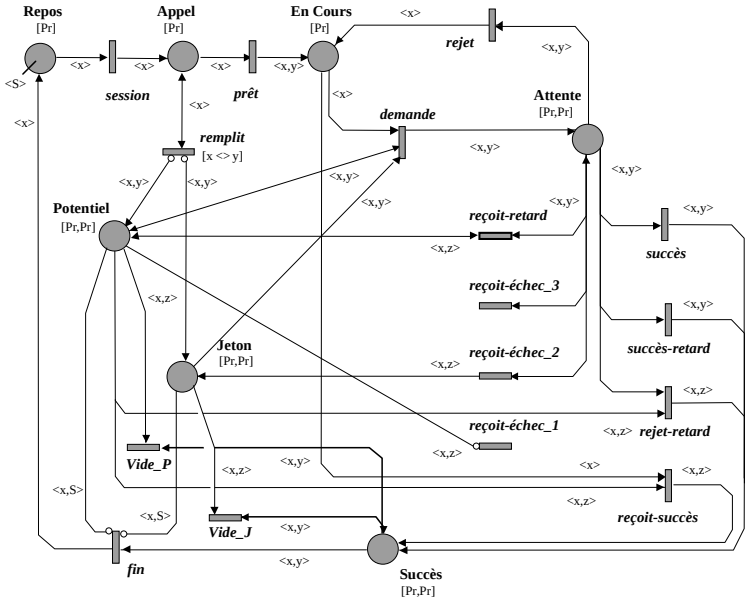

Figure 3. *Modèle du protocole des sites (excepté la gestion des retards)*

Ainsi sur le deuxième exemple, s_1 retarde sa réponse tandis que s_3 (respectivement s_2) rejette la demande de s_2 (respectivement s_1). Une fois reçu le rejet de s_2, s_1 peut répondre favorablement à s_3 et un rendez-vous est pris.

Dans le paragraphe qui suit, nous allons spécifier formellement cet algorithme à l'aide d'un réseau de Petri bien formé.

3.1.2. *Modélisation du comportement symétrique de l'algorithme*

Notre modélisation se déroule en deux étapes : nous spécifions un comportement symétrique de l'algorithme par un réseau bien-formé puis nous interdisons les séquences qui ne respectent pas la propriété introduite au paragraphe précédent à l'aide d'un automate de contrôle que nous présenterons dans la section suivante.

L'ensemble des sites de l'application répartie est décrit par une classe de couleurs $I = 1..N$ où N est le nombre de sites. Cette classe ne comporte pas de sous-classes statiques. La deuxième classe de couleurs représentant les types de messages est constituée de trois sous-classes statiques réduites à un élément : a pour un acquittement, r pour un rejet et d pour une demande. Les domaines

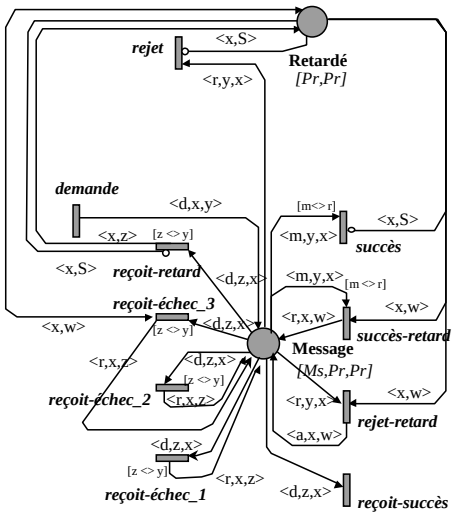

Figure 4. *Modèle de gestion des messages et des retards de réponse*

des places et des transitions sont des produits cartésiens d'occurences de ces deux classes.

Afin de rendre le réseau bien-formé compréhensible, nous le considérons comme ontenu par une fusion sur les transitions des sous-réseaux des figures 3 et 4.

Les places *Repos*, *Appel*, *En Cours*, *Attente* et *Succès* représentent l'évolution du protocole du point de vue de chaque site.

- Une marque de couleur $\langle x \rangle$ dans la place *Repos* signifie que le site s_x est au repos.
- Une marque de couleur $\langle x \rangle$ dans la place *Appel* signifie que le service de rendez-vous du site s_x est requis.
- Le "déplacement" de cette marque vers la place *En Cours* indique le début d'une session de recherche de rendez-vous pour ce site.
- La place *Attente* est marquée par un jeton de couleur $\langle x, y \rangle$, pour indiquer que s_x a émis une demande de rendez-vous vers un site s_y.
- Enfin, un jeton de couleur $\langle x, y \rangle$ marque la place *Succès* lorsque le site s_x a obtenu un rendez-vous avec le site s_y.

Les trois places suivantes correspondent aux variables de contrôle de chaque site. Les deux premières apparaissent sur la figure 3 et la troisième sur la figure 4.

- La place *Jeton* modélise les autorisations d'envoi de demande d'un site vers un autre. Un jeton de couleur $\langle x, y \rangle$ dans cette place indique que le

site s_x a le droit d'envoyer une demande vers le site s_y.
- La place *Potentiel* modélise pour chaque site s_x, l'ensemble des sites parmi lesquels s_x désire choisir un partenaire pour un rendez-vous. Un jeton de couleur $\langle x, y\rangle$ signifie que s_y appartient à l'ensemble des sites spécifiés par l'application du site s_x pour la demande courante de rendez-vous.
- La place *Retardé* est marquée par un jeton de couleur $\langle x, w\rangle$ pour indiquer que le site s_x retarde sa réponse au site s_w.

Enfin la place $Message$ (présente sur la figure 4) représente les messages en transit suivant le format $\langle type\ de\ message, source, destinataire\rangle$.

Initialement, tous les sites sont aux repos. Aussi le marquage initial se réduit à la fonction constante $\langle S_I\rangle$ (i.e. une somme de jetons, un par couleur de I) marquant la place *Repos*. Par souci de simplification, on a omis sur la figure l'indice de S_I.

Nous décrivons maintenant le rôle des transitions de ce réseau.

Le mécanisme de spécification de l'ensemble des sites partenaires possibles d'un rendez-vous se fait au moyen de la transition *remplit*. Lorsqu'un site s_x se trouve dans l'état *Appel*, cette transition permet d'ajouter une marque $\langle x, y\rangle$ simultanément aux places *Potentiel* et *Jeton*. Les arcs de "remplissage" vers ces places sont en fait des doubles arcs de même valuation (arc sortant standard et arc entrant de type inhibiteur), de façon à éviter la production de doublons. Une fois la transition *prêt* franchie, l'ensemble des sites avec lesquels le site s_x désire avoir un rendez-vous est "figé" dans la seconde composante des marques $\langle x, y\rangle$ contenues dans *Potentiel*. La transition *demande* correspond à l'émission d'une demande de rendez-vous vers l'un des sites possibles à condition de disposer d'une autorisation de demande (qui sera consommée) dans la place *Jeton*. Cette transition produit une marque de la place $Message$ correspondant au message de demande (figure 4). L'état du site demandeur devient *Attente* (dans la deuxième composante du jeton produit se trouve le site dont on attend la réponse).

A la réception d'une demande, quatre comportements sont possibles : un retard de réponse, l'absence de réponse si les demandes se sont croisées, l'envoi d'un acquittement et l'envoi d'un rejet. Nous décrivons les différents cas ci-dessous.

La transition *recoit-retard* modélise le fait que la réponse à une demande d'un site s_z est retardée si le site s_x est dans l'état *Attente* de réponse d'un autre site s_y, que s_z est un candidat possible pour s_x et que s_x n'a pas retardé d'autre site. On remarquera que la condition asymétrique $[z > x]$ imposée dans l'algorithme n'est pas représentée sur le réseau. Notons que la garde $[z <> y]$ exclut de traiter par cette transition les demandes croisées. Ceci est aussi valable pour les transitions *recoit-échec_i* pour i de 1 à 3.

Les trois transitions suivantes correspondent à la réception d'une demande qui fait évoluer le récepteur vers l'état *Succès*.

- Si l'état du récepteur est *En Cours* et que la demande provient d'un candidat possible alors la transition *recoit-succès* est franchie.
- Si le récepteur s_x attend une réponse de s_y et reçoit une demande de celui-ci, il passe à l'état *Succès*. Autrement dit, les demandes se sont croisées et la demande vaut pour un acquittement. Deux transitions *succès* et *succès-retard* correspondent à ce cas. La seconde se distingue de la première par l'existence d'un site demandeur dont la réponse avait été retardée, ce qui produit un message de rejet vers ce site. Comme ces deux transitions sont activables aussi bien par la réception d'une demande que par celle d'un acquittement, le type du message est une variable u qui est contrainte par la garde $[u \neq r]$ (i.e. le type de message accepté est $u \in \{d, a\}$).

Les trois transitions suivantes modélisent différents cas d'émission d'un message de rejet, lors de la réception d'une demande de rendez-vous par un site en *Attente*.

- La transition *recoit-échec_1* indique que le site demandeur ne fait pas partie des sites désirés par le récepteur (voir l'arc inhibiteur de cette transition issu de la place *Potentiel*).
- La transition *recoit-échec_2* modélise le fait qu'une demande d'un site s_z est rejetée par s_x si le site s_x attend une réponse d'un troisième site bien que s_z soit un candidat possible pour s_x. Cette transition ne doit être franchie que si $[z < x]$ autrement dit si la transition *recoit-retard* est infranchissable. Nous garantissons ceci en introduisant une priorité de franchissement entre ces deux transitions. De plus cette transition est la seule qui fournit au récepteur après la phase d'appel une autorisation d'émettre une demande vers le site dont il rejette la demande. Sans cette nouvelle autorisation deux sites pourraient désirer un rendez-vous mutuel alors que leur autorisation respective aurait déjà été utilisée.
- La transition *recoit-échec_3* est franchie dans le cas où le récepteur, en attente d'une réponse, a déjà retardé un autre candidat.

Les réceptions d'acquittement sont traitées par les mêmes transitions que celles associées aux réceptions d'une demande croisée. Il nous reste donc à considérer les réceptions d'un rejet.

- La transition *rejet-retard* correspond à la réception d'un message de rejet en provenance du candidat potentiel alors que la réponse à un autre site avait été retardée. Ce dernier obtient alors le RDV et est prévenu par l'émission d'un message d'acquittement.
- La transition *rejet* modélise le fait que la réception d'un rejet en provenance du site candidat conduit à un retour à l'état *En Cours*, au cas où il n'y aurait pas eu de réponse retardée.

Avant toute nouvelle session de RDV pour un site, il convient pour ce site de ré-initialiser les autorisations non utilisées et l'ensemble des candidats possibles de la session courante. Le "nettoyage" des places *Potentiel* et *Jeton* est opéré pour toute couleur $\langle x \rangle$ qui marque la place *Succès*, par des franchissements de transitions $vide_P$ et $vide_J$ respectivement. Une fois ces deux places vidées de jetons dont la première composante est x, la transition fin devient franchissable pour la couleur $\langle x \rangle$, ce qui déplace cette couleur vers la place *Repos*. Rappelons que de nouvelles marques seront produites pour ces places au début de la prochaine session (voir la place *Appel*).

Nous pouvons facilement donner une borne inférieure de la complexité de ce modèle mesurée en nombre d'états. On remarque que dans un état donné, les places *Jeton* et *Potentiel* peuvent (ou non) contenir une marque $\langle x, y \rangle$ pour deux couleurs distinctes de I. Autrement dit, le nombre de marquages distincts est d'au moins $2^{N \cdot (N-1)}$. Une analyse du contenu possible des autres places montre que ce facteur est prépondérant dans la complexité de l'espace d'accessibilité.

Aussi on pourrait envisager de réduire la complexité du réseau, en limitant arbitrairement les différentes possibilités de marquage de la place *Potentiel*. Par exemple, chaque site désirerait communiquer avec tous les autres à chaque session de rendez-vous. Cependant il faudrait encore démontrer l'équivalence entre la validité d'une propriété pour le modèle réduit et cette même validité pour le modèle initial. De plus, il n'est pas évident que la réduction soit significative, car la place *Jeton* aurait encore un nombre de marquages possibles du même ordre de grandeur.

Comme nous l'avons indiqué, la modélisation de l'algorithme induit un sur-ensemble du comportement de l'algorithme car nous n'avons pas spécifié la condition $[z > x]$ pour le franchissement de la transition *recoit-retard*. Aussi dans la section qui suit nous introduisons un automate de contrôle asymétrique qui sera synchronisé avec le réseau bien formé. Rappelons qu'alors la condition manquante $[z < x]$ sur le franchissement *recoit-échec_2* sera respectée grace au mécanisme des priorités entre transitions.

3.1.3. *Restriction des comportements de l'algorithme*

Nous allons éliminer les séquences incorrectes du réseau de Petri bien formé en interdisant la situation où un site en attente retarde un site d'identité inférieure.

Il y a correspondance bi-univoque entre un état de l'automate représenté sur la figure 5 et un sous-ensemble A de sites en attente. Autrement dit, le nombre d'états de cet automate est de 2^N ce qui reste relativement petit en

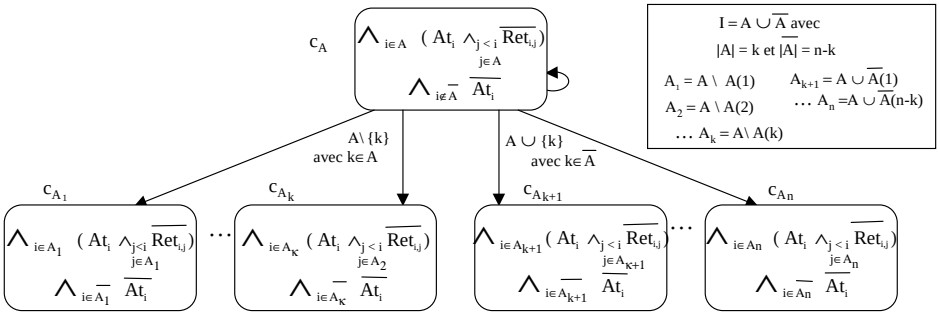

Figure 5. *Automate de contrôle des asymétries*

regard de la taille du graphe d'accessibilité (voir le paragraphe précédent). Les propositions atomiques de cet automate sont At_i qui est vérifiée si une marque $\langle i \rangle$ est présente dans la place *Attente* et $Ret_{i,j}$ qui est vérifiée si une marque $\langle i, j \rangle$ est présente dans la place *Retardé*. Les transitions de cet automate correspondent à l'entrée d'un site en attente ou sa sortie.

Dans l'état correspondant au sous-ensemble A, la partition locale est définie comme suit : chaque couleur de A correspond à une sous-classe statique locale tandis que $\overline{A}$ tout entier est une sous-classe statique locale. Ainsi, moins il y a de sites en attente, plus l'état de l'automate est symétrique.

3.2. *Spécification de la propriété à vérifier*

Nous illustrons la méthode sur la formule d'équité suivante : *aucun site s_i ne peut rester indéfiniment bloqué en attente d'un rendez-vous alors qu'infiniment souvent des appels au service de rendez-vous par des partenaires possibles de s_i incluent s_i comme partenaire possible.*

Comme décrit au premier chapitre, nous devons utiliser l'automate de Büchi correspondant à la négation de cette formule. Cependant nous ne désirons pas prendre en compte toutes les séquences infinies de notre modélisation. En effet dans le réseau bien formé, rien n'empêche deux sites de prendre une infinité de rendez-vous alors que deux autres sites pourraient avoir un rendez-vous de manière indépendante. Autrement dit, le réseau de Petri autorise des séquences où un site ne progresse pas dans son protocole alors qu'il n'est pas en attente d'un message.

L'automate de Büchi doit donc reconnaitre les séquences d'exécution équitables où un site s_i reste indéfiniment bloqué en attente d'un rendez-vous alors

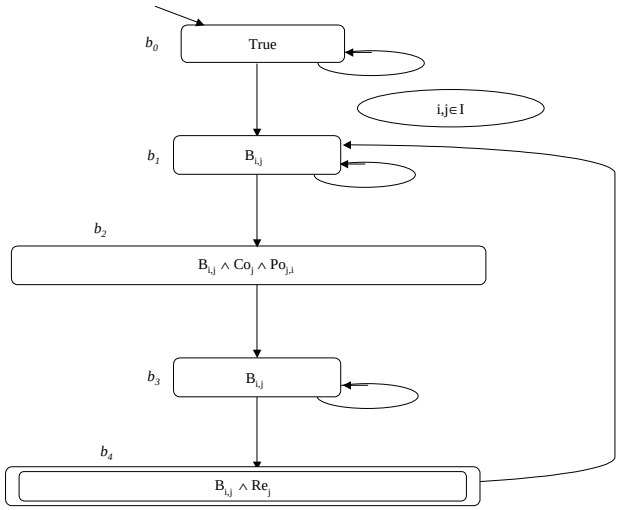

Figure 6. *Automate de Büchi de* $\overline{f}$

qu'infiniment souvent des appels au service de rendez-vous par des partenaires possibles de s_i incluent s_i comme partenaire possible. Dans ce cas, puisque le nombre de sites est fini, on peut trouver une formulation équivalente qui est la suivante : l'automate de Büchi doit reconnaitre les séquences d'exécution équitables où un site s_i reste indéfiniment bloqué en attente d'un rendez-vous alors qu'infiniment souvent des appels au service de rendez-vous par un autre site s_j partenaire possible de s_i incluent s_i comme partenaire possible.

La spécification de la formule $\overline{f}$ s'exprime comme suit :

$$\overline{f} = \vee_{i \in I} \vee_{j \neq i} [\mathbf{F}\ \mathbf{G}\ B_{i,j} \wedge \mathbf{G}\ \mathbf{F}\ (Co_j \wedge Po_{j,i}) \wedge \mathbf{G}\ \mathbf{F}\ Re_j]$$

La proposition atomique $B_{i,j}$ est vérifiée si une marque $\langle i \rangle$ est présente dans la place *En cours* et une marque $\langle i, j \rangle$ est présente dans la place *Potentiel*. La proposition atomique Co_j est vérifiée si une marque $\langle j \rangle$ est présente dans la place *En cours*. La proposition atomique $Po_{j,i}$ est vérifiée si une marque $\langle j, i \rangle$ est présente dans la place *Potentiel*. La proposition atomique Re_j est vérifiée si une marque $\langle j \rangle$ est présente dans la place *Repos*.

Aussi infiniment souvent le site s_j désirera des rendez-vous incluant s_i comme partenaire possible et obtiendra un rendez-vous mais avec un autre site que s_i (puisque celui-ci est indéfiniment dans *En cours*).

L'automate de Büchi de cette formule est partiellement représenté par la figure 6. Par mesure de clarté, une seule des branches (i, j) a été explicitée. La taille de l'automate est donc d'ordre quadratique. Ici, la structure de l'automate

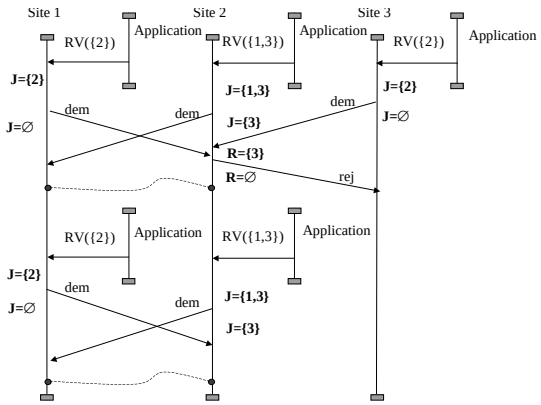

Figure 7. *Une exécution fautive*

est déduite immédiatement de la formule à invalider. Dans le premier état, on attend le blocage d'un site qui intervient dans le deuxième état. Une fois cet état atteint, on alterne l'occurence des états où l'on a soit $Co_j \wedge Po_{j,i}$ soit Re_j. Le choix de l'état final garantit cette infinité d'occurences. L'automate de la figure 6 est certainement plus réduit que celui qui aurait été obtenu par une traduction automatique telle qu'elle est décrite au premier chapitre. Par exemple, on a tenu compte du fait qu'un site ne passait jamais directement de l'état *Repos* à l'état *En cours* et vice versa.

L'algorithme de rendez-vous admet des séquences qui ne vérifient pas la formule d'équité. La figure 7 illustre l'une de ces séquences. Le scénario est représenté sous forme d'un chronogramme où le temps est représenté verticalement et chaque axe correspond à l'activité d'un des trois sites. Les messages sont des flèches obliques dont l'origine est associée à l'émission du message et l'extrémité est associée à la réception du message.

Cette séquence débute par trois appels à la primitive de rendez-vous. Les sites s_1 et s_3 désirent un rendez-vous avec s_2 tandis que celui-ci désire un rendez-vous avec l'un des autres sites. s_2 choisit d'émettre sa première demande vers s_1. A la réception de la demande de s_3, il retarde sa réponse. Puis à la réception de la demande de s_1, il conclut le rendez-vous et rejette s_3. Le site s_3 peut alors rester indéfiniment bloqué tandis que les deux autres sites concluent une infinité de rendez-vous. Le site s_2 inclut toujours s_3 dans sa demande mais choisit systématiquement s_1 pour envoyer sa première demande.

3.3. *Construction du produit synchronisé symbolique*

3.3.1. *Calcul des partitions locales de couleurs*

La recherche d'une séquence invalidante doit s'effectuer dans un double produit synchronisé, mettant en jeu le réseau bien formé symétrique, l'automate de contrôle des asymétries et l'automate de la formule. Or le produit synchronisé est une opération commutative et associative. On peut donc synchroniser d'abord les deux automates puis appliquer la construction d'un produit synchronisé symbolique sur le réseau de Petri bien formé et l'automate produit. De cette façon la méthode présentée dans le cadre de systèmes symétriques est applicable aussi aux systèmes asymétriques à condition de pouvoir les exprimer comme un produit synchronisé d'un système symétrique et d'un automate asymétrique.

Avant de construire le produit synchronisé symbolique, il nous faut calculer et représenter l'orbite de chaque état de l'automate. Nous nous limiterons ici à un réseau bien formé constitué d'une seule classe de 4 sites $I = 1..4$. La manière la plus simple de procéder consiste à *repérer* les couleurs qui marquent les mêmes propositions dans les états des automates. Les couleurs sont donc regroupés en sous-ensembles, qui constituent la partition locale de l'état. L'orbite est implicitement définie comme le groupe des permutations laissant globalement invariant chaque ensemble de la partition. Par exemple :

- Pour un état dont les propositions atomiques sont $\langle \overline{At_1}, \overline{At_2}, \overline{At_3}, \overline{At_4} \rangle$, la partition est composée de l'unique singleton $\{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle \langle 4\rangle\}$ (i.e. toutes les permutations de couleurs sont autorisées)
- Pour un état dont les propositions atomiques sont $\langle \overline{At_2} \rangle$, la partition est composée d'un singleton $\{\langle 2\rangle\}$ et d'un ensemble composé du reste des couleurs, ici $\{\langle 1\rangle, \langle 3\rangle, \langle 4\rangle\}$ (les permutations doivent laisser invariant $\langle 2\rangle$)
- Pour un état dont les propositions atomiques sont $\langle Co_1, At_2, \overline{At_3} \rangle$, la partition est composée des quatre éléments, $\{\langle 1\rangle\}$, $\{\langle 2\rangle\}$, $\{\langle 3\rangle\}$ et $\{\langle 4\rangle\}$. L'orbite est réduite à l'identité.

3.3.2. *Les états du produit synchronisé symbolique*

Rappelons que nous devons représenter des triplets (sous-groupe de permutations, sous-ensemble de marquages, état de l'automate) où le sous-ensemble de marquages est l'image d'un quelconque état du sous-ensemble par le sous-groupe. Le sous-groupe sera implicitement représenté par une partition locale non nécessairement égale à la partition de l'état de l'automate. Le sous-ensemble des marquages sera représenté par un marquage symbolique dont les sous-classes statiques (ici locales) sont celles de la partition. Rappelons qu'un

marquage symbolique construit en rapport à une partition en sous-classes statiques des couleurs est en fait construit sur une partition *anonyme* plus fine, dite partition en *sous-classes dynamiques*. Ainsi, les couleurs qui appartiennent à la même sous-classe statique locale et qui ont la même distribution seront représentées symboliquement et de façon indifférenciée, par la même sous-classe dynamique (dénotée communément au moyen d'une variable Z indicée). L'appartenance à une sous-classe statique et le nombre d'éléments représentés sont les caractéristiques essentielles de toute sous classe dynamique.

La configuration initiale pour laquelle tous les sites sont au *Repos*, correspond à un unique état dans le produit synchronisé symbolique : $\langle \widehat{m}_0, b_0, c \rangle$ où $\widehat{m}_0$ est défini par :

- une unique sous classe statique locale $\{\langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle 4 \rangle\}$ composée d'une unique sous classe dynamique (notée Z_1),
- $\widehat{m}_0 = \langle Z_1 \rangle.Re$ avec $\mid Z_1 \mid = 4$ (Re représente ici la place *Repos*)

Les propriétés atomiques de b_0 (*true*) et $c_\emptyset$ sont satisfaites trivialement par $\widehat{m}_0$.

3.3.3. *Franchissement symbolique dans le produit synchronisé symbolique*

Une rapide évaluation du fonctionnement du réseau de Petri bien formé montre que plus de 50 transitions sont nécessaires pour construire le contre-exemple. Nous nous limiterons donc à représenter une suite de deux franchissements symboliques qui illustre le regroupement des sous-classes statiques.

Rappelons qu'un état est constitué d'un marquage symbolique, d'un état de l'automate de contrôle et d'un état de l'automate de Büchi.

Le marquage symbolique $\widehat{m}_1$ décrit la situation suivante. Le site s_1 est au repos ; les sites s_2 et s_3 ont conclu un rendez-vous et le site s_4 est en attente d'un rendez-vous avec s_1 comme seul candidat possible. Le site s_4 a utilisé son autorisation et se trouve bloqué dans l'état en cours. Comme tous les sites sont précisés, on en conclut que la partition locale est constitué de quatre sous-classes statiques, chacune comprenant une unique sous-classe dynamique.

$$\widehat{m}_1 =$$
$$\langle Z_1 \rangle.Repos + \langle Z_4 \rangle.En\ cours + \langle Z_4, Z_1 \rangle.Potentiel + (\langle Z_2, Z_3 \rangle + \langle Z_3, Z_2 \rangle).Succès$$

où chaque sous-classe dynamique Z_i est associée à la sous-classe statique $\{i\}$.

L'état de l'automate de contrôle est nécessairement $c_\emptyset$. Nous supposons de plus que l'état de l'automate est b_0.

La transition fin est franchissable soit pour la couleur $\langle 2 \rangle$, soit pour la couleur $\langle 3 \rangle$. Nous allons effectuer successivement ces deux franchissements sym-

boliques (qui dans ce cas particulier sont assimilables à des franchissements ordinaires). Remarquons que les états associés dans les deux automates seront inchangés. Soit $\widehat{m}_2$ le marquage obtenu après franchissement de $fin(\langle 2 \rangle)$.

$$\widehat{m}_2 = (\langle Z_1 \rangle + \langle Z_2 \rangle).Repos + \langle Z_4 \rangle.En\ cours + \langle Z_4, Z_1 \rangle.Potentiel + \langle Z_3, Z_2 \rangle.Succès$$

Aucune paire de sous-classes statiques de la partition locale ne peut être fusionnée car les sous-classes dynamiques ont des marquages différents.

Soit maintenant $\widehat{m}_3$ le marquage obtenu après franchissement de $fin(\langle 3 \rangle)$.

$$\widehat{m}_3 = (\langle Z_1 \rangle + \langle Z_2 \rangle + \langle Z_3 \rangle).Repos + \langle Z_4 \rangle.En\ cours + \langle Z_4, Z_1 \rangle.Potentiel$$

Les deux sous-classes statiques $\{2\}$ et $\{3\}$ de la partition locale peuvent être fusionnées car elles contiennent chacune une sous-classe dynamique de marquage identique et ceci nous conduit à une nouvelle version de $\widehat{m}_3$.

$$\widehat{m}_3 = (\langle Z_1 \rangle + \langle Z_2 \rangle).Repos + \langle Z_3 \rangle.En\ cours + \langle Z_3, Z_1 \rangle.Potentiel$$

avec trois sous-classes statiques : $\{1\}$ dont l'unique sous-classe dynamique est Z_1, $\{2, 3\}$ dont l'unique sous-classe dynamique est Z_2 et $\{4\}$ dont l'unique sous-classe dynamique est Z_3.

4. Discussion

L'approche que nous proposons ici peut être améliorée en exploitant les symétries pour réduire l'automate, avant la phase de vérification proprement dite [AJA 98]. Lors de l'analyse de l'automate si deux états sont bisimulables à une permutation près, seul un des états est conservé. Pour des formules presques symétriques cette approche s'avère très fructueuse et la réduction de l'automate peut être alors exponentielle. Malgré tout, la réduction essentielle se situe au niveau du produit synchronisé symbolique.

Assez souvent, comme dans notre exemple on est conduit à vérifier des formules symétriques sur des systèmes asymétriques. L'approche que nous avons choisi de présenter a le mérite de la simplicité et de l'efficacité. Cependant cette approche est impraticable si l'asymétrie ne peut être spécifiée par restriction de séquences. On peut alors par exemple combiner notre méthode avec celle développée dans [EME 99]. Cette approche autorise l'asymétrie des transitions partant d'un état complètement symétrique (invariant sous l'action du groupe de symétries). Une approche plus sytématique est possible en définissant un système asymétrique comme un système dont chaque état a son propre groupe de symétries. On étendr alors la méthode de la section 2 par une gestion plus fine de la règle de franchissement symbolique.

La taille des modèles spécifiés est aussi une composante de la complexité de notre approche, puisque la vérification est fondamentalement basée sur le produit synchronisé des états de la formule, de l'automate de contrôle des asymétries et du RDP symétrique. On remarquera que l'automate de contrôle des asymétries du protocole de Bagrodia étudié était relativement petit, comparé à la taille de l'espace d'états du réseau bien formé. De ce point de vue le problème de la réduction de modèle semble concentré au niveau du réseau de Petri bien formé.

Il reste que l'automate de contrôle des asymétries doit pour l'instant être spécifié manuellement au même titre que le réseau. L'un des problèmes ouverts est donc d'aider à la construction d'un modèle des asymétries le plus concis possible, tout en cernant au maximum les conditions où les asymétries doivent être prises en compte. La notation indexée que nous avons exploitée sur la classe de couleurs principale du modèle participe à cet effort.

L'implémentation des travaux présentés est en cours. Le fait que nous réutilisions l'approche symbolique nous permet d'exploiter certaines fonctionnalités de l'outil GreatSPN [CHI 97].

5. Conclusion

La construction du graphe symbolique d'un réseau de Petri bien formé permet la représentation succincte de l'espace d'accessibilité. Cependant ce graphe ne peut être exploité directement pour vérifier la validité de formules de logique temporelle. Nous avons présenté ici une adaptation de la technique de construction du graphe symbolique qui permet la vérification d'une formule de logique temporelle même asymétrique. Cette technique est déduite d'une méthode applicable à un système symétrique quelqu'en soit le formalisme de représentation du moment que sa sémantique soit définie en terme de structure de Kripke. Nous avons illustré cette méthode sur un algorithme réparti de rendez-vous multiple. L'intérêt de cet exemple est double. D'une part, il montre comment appliquer la méthode à un système partiellement symétrique en le spécifiant comme le produit synchronisé d'un réseau bien formé et d'un automate asymétrique. D'autre part, cette modélisation souligne l'intérêt des mécanismes additionnels tels que les arcs inhibiteurs et les priorités pour obtenir un réseau concis et lisible.

Références

[AJA 98] K. Ajami, S. Haddad et J.-M. Ilié. Exploiting Symmetry in Linear

Temporal Model Checking : One Step Beyond. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems TACAS'98, part of Theory and practice of Software ETAPS'98*, volume 1384 of *LNCS*, pages 52–67, Lisbon - Portugal, April 1998. Springer Verlag.

[BAG 89] R.L. Bagrodia. Synchronization of asynchronous processes in CSP. *ACM Toplas*, 11,4 :585–597, 1989.

[CHI 97] G. Chiola, G. Franceshinis, R. Gaeta et M. Ribaudo. GreatSPN1.7 : GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation, North Holland Journal*, 24, 1997.

[CLA 96] E.M. Clarke, R. Enders, T. Filkorn et S. Jha. Exploiting Symmetry in Temporal Logic Model Chacking. *Formal Methods and System Design*, 9 :77–104, 1996.

[DIA 01] M. Diaz, ed. *Les réseaux de Petri. Modèles fondamentaux.* Hermes, 2001.

[EME 90] E.A. Emerson. Temporal and Modal Logic. *Handbook of Theoretical Computer Science*, B :995–1072, 1990.

[EME 96] E.A. Emerson et A. Prasad Sistla. Symmetry and Model Checking. *Formal Methods and System Design*, 9 :307–309, 1996.

[EME 99] E. A. Emerson et R. J. Trefler. From Asymmetry to Full Symmetry : New Techniques For Symmetry Reduction in Model Checking. In *Proc of CHARMEÕ99*, Lecture Notes in Computer Science, pages 142–156, Bad Herrenalb - Germany, September 1999. Springer Verlag.

[GER 93] R. Gerth, D. Peled, M. Vardi et P. Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. In *Proc. Int Conf. on Protocol Specification Testing and Verification*, 1993.

[HAD 00] S. Haddad, J-M. Ilié et K. Ajami. A model checking method for partially symmetric systems. In *FORTE 2000,* 13^{th} *Formal Description Techniques for Distributed Systems and Communication Protocols*, Pise, Italie, Octobre 2000.

[KOZ 83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27 :333–354, 1983.

[LAN 77] S. Lang. *Algebra.* seventh printing. Addison-Wesley, 1977.

[PNU 77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[VAR 96] M. Vardi. An Automata-theoretic Approach to Linear Temporal Logic. *Lecture Notes in Computer Science*, 1043 :238–266, 1996.