Chapitre 1
Vérification de propriétés spécifiques

Serge HADDAD , François VERNADAT[1]

1. Introduction

Dans le premier volume de cette collection traitant des Réseaux de Petri [DIA 01], nous avons étudié la vérification de propriétés génériques de réseaux de Petri telles que le caractère borné ou la vivacité [HV 01]. Si ces propriétés renseignent le modélisateur sur le comportement général du réseau, celles-ci doivent être complétées par l'analyse des propriétés spécifiques du système modélisé. Aussi nous nous penchons dans ce chapitre sur l'expression et la vérification de propriétés spécifiques des réseaux de Petri.

Le plus souvent, le concepteur d'une application définit les fonctions et/ou les services de celle-ci à travers une spécification. Une fois son application modélisée, il désire vérifier que son modèle se conforme à la spécification. Afin de développer des algorithmes et des outils pour cette vérification, il est nécessaire de formaliser le concept de spécification. Deux possibilités ont été largement étudiées à cette fin : soit la spécification est définie par un ensemble de formules d'une logique adéquate, soit la spécification est définie à l'aide d'un modèle de comportement. Nous explorerons donc ces deux voies qui dans la pratique sont complémentaires : certaines propriétés s'exprimeront plus facilement à l'aide de formules et d'autres plus facilement à l'aide d'un comportement. Pour fixer les idées, considérons un exemple simplifié d'allocation en exclusion mutuelle de ressources : 2 clients sont en compétition pour accéder à une ressource. Le

[1] LAMSADE, Université Paris Dauphine, Place du Maréchal de Lattre de Tassigny, 75775 Paris cedex 16, France, (serge.haddad@lamsade.dauphine.fr)
LAAS-CNRS, 7, Avenue du Colonel Roche F-31077 Toulouse cedex, (francois.vernadat@laas.fr)

contrôle de l'accès en exclusion mutuelle de celle-ci est assuré par un mécanisme dont nous ferons abstraction.
Parmi les propriétés à vérifier, nous devons exprimer la propriété d'exclusion mutuelle : P1 "La ressource est utilisée au plus par un client", et par analogie au problèmes des philosophes, ce que l'on nomme généralement l'absence de famine : P2 "Tout client en attente de la ressource finira par l'obtenir". On veut aussi pouvoir spécifier le fonctionnement d'un client et exprimer P3 "Le client envoie d'abord une requête pour obtenir la ressource (a), qu'il reçoit ensuite un accord d'utilisation (b) et que finalement (c) il envoie un message de libération avant de retourner à son état initial". De façon générale, P1 et P2 s'exprimeront simplement à l'aide de formules de logiques temporelles tandis que P3 s'exprimera de façon compacte à l'aide d'un comportement tel que celui représenté sur la gauche de la figure 1. A contrario, P1 ne s'exprimera que de façon très indirecte par le comportement représenté sur la droite de la figure 1 : entre deux entrées consécutives en section critique (événement ? Ack), se trouve forcément la sortie du seul client présent en section critique (événement ! Lib).

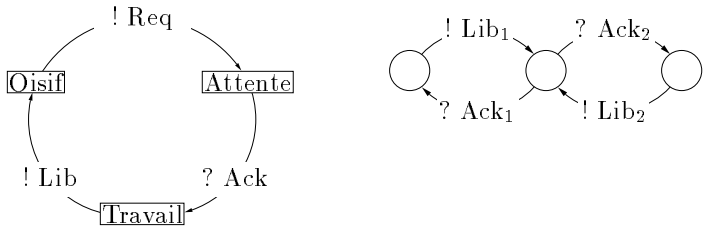


Figure 1: Exemples de spécifications comportementales

Une logique apte à raisonner sur le comportement de systèmes dynamiques à événements discrets doit nécessairement intégrer la notion de séquence d'états (finie ou infinie) correspondant à une exécution possible. De plus elle doit être en mesure d'exprimer des propriétés de sûreté comme "Au plus un processus en cours d'exécution d'une section critique, dans tout état de la séquence" (cf P1), des propriétés de vivacité comme "Si dans un état, un processus demande à exécuter une section critique alors dans un état futur ce processus exécutera cette section critique" (cf P2) et des propriétés d'équité comme "Tout processus en mesure de s'exécuter dans une infinité d'états sera choisi par l'ordonnanceur une infinité de fois". Le concept clef est ici le temps vu comme une succession discrète d'instants et les logiques qui intègrent cette notion sont appelées logiques temporelles.

Ces logiques se distinguent selon deux axes. Le parallélisme et/ou le non déterminisme impliquent l'existence de différentes exécutions d'un même système

et nécessitent leur prise en compte simultanée. Aussi :

- Soit on représente l'ensemble des exécutions comme un arbre où les différents successeurs d'un état sont obtenus par les instances d'événements possibles en cet état. On parle alors de *logique temporelle arborescente*.
- Soit on considère l'ensemble des exécutions comme des séquences distinctes. On parle alors de *logique temporelle linéaire*.

Le deuxième axe concerne les éléments de la séquence.

- Soit on considère une séquence d'états caractérisés par un ensemble de propositions atomiques. On parle alors de *logique temporelle propositionnelle*.
- Soit on considère une séquence de transitions élémentaires, chacune étiquetée par un événement. On parle alors de *logique temporelle événementielle*.

Aussi dans la première partie, nous introduirons la syntaxe et la sémantique d'une logique arborescente propositionnelle appelée $\mathcal{CTL}^*$. Nous en présenterons deux fragments très etudiés $\mathcal{CTL}$ et $\mathcal{LTL}$ (une logique temporelle linéaire). Nous montrerons ensuite comment effectuer la vérification de formules sur des modèles à états finis. Nous achèverons cette section en indiquant les adaptations à prendre en compte dans le cadre des réseaux de Petri à la fois dans un contexte propositionnel et événementiel. Le point principal est de considérer de manière appropriée les différents types de séquence de franchissement (finie, maximale finie ou infinie).

Après avoir vu l'approche logique pour la vérification, nous nous intéresserons à l'approche "comportementale". L'approche logique est parfois qualifiée de "double modèle" au sens où l'on dispose d'une logique pour spécifier les propriétés à vérifier et d'une structure qui représente le comportement du système (la "structure de Kripke" que l'on peut associer au graphe des marquages accessibles dans le cas d'un système décrit par un réseau de Petri). A contrario, l'approche comportementale est parfois qualifiée de "simple modèle" au sens où l'on ne dispose que d'une seule structure, un "système de transitions étiqueté" (une structure voisine du graphe des marquages accessibles d'un réseau de Petri). Celui-ci permet de représenter à la fois le comportement du système et sa spécification.

L'approche comportementale procède par comparaison. A l'aide de différentes relations d'équivalences ou de pré-ordres, on compare deux comportements : ceux-ci satisfont les mêmes propriétés si et seulement si ils sont équivalents. Différentes équivalences de comportement ont été introduites pour prendre en compte plusieurs classes de propriétés ou, de façon équivalente, plusieurs points de vue à considérer lorsque l'on analyse un système. Parmi ces différents points de vue, nous retrouverons la prise en compte du parallélisme et du non-déterminisme. Comme pour les logiques temporelles, nous serons amenés à distinguer deux grandes familles de relations d'équivalences de

comportements : la famille des "équivalences de traces" qui perçoit le fonctionnement d'un système à travers l'ensemble de ses séquences d'exécution (cf les logiques temporelles linéaires) et la famille des "bisimulations" qui perçoit le fonctionnement d'un système à travers son "arbre" d'exécution (cf les logiques temporelles arborescentes). Pour ces deux familles, on peut aussi être amené à privilégier les "états" d'une exécution (cf les logiques temporelles propositionnelles) ou les événements qui consituent l'exécution (cf les logiques temporelles évenementielles).

Une premiere section nous permettra de présenter de façon informelle différents points de vue possibles lorsque l'on compare le comportement de deux systèmes. La seconde section présentera la notion de bisimulation et de simulation. Les procédures de décision associées seront présentés. La troisième partie présentera les équivalences "faibles" qui permettent de comparer des systèmes décrits à différents niveaux d'abstraction. La dernière section s'attachera à montrer les liens entre l'approche comportementale et l'approche logique : nous présenterons en particulier la logique $\mathcal{HML}$ [HM 85] qui permet de caractériser "modalement" la relation de bisimulation. Dans l'autre sens, nous présenterons les résultats de [BCG 91] qui permettent de caractériser "comportementalement" la logique temporelle $\mathcal{CTL}^*$.

Dans la dernière partie, nous analyserons la décidabilité de l'évaluation de formules de logique temporelle sur un réseau de Petri et le test de bisimulation d'un réseau marqué avec un système de transitions étiqueté. Plus précisément, nous établirons que dans le cadre d'une logique temporelle propositionnelle, l'évaluation est indécidable aussi bien pour le fragment $\mathcal{CTL}$ que pour le fragment $\mathcal{LTL}$. Ce résultat s'étend à la logique arborescente événementielle. Dans ces trois cas, les formules utilisées ne requièrent qu'un nombre très limité d'opérateurs temporels ce qui indique la robustesse du résultat (voir par exemple [ESP 98]). En s'appuyant sur des arguments similaires, on montrera que le test de bisimulation de deux réseaux marqués est aussi indécidable [JAN 95].

A l'inverse pour une logique temporelle linéaire événementielle très expressive (le μ-calcul linéaire), l'évaluation de formules reste décidable [ESP 97]. Dans le cas de séquences maximales finies, la procédure s'appuie sur la décidabilité de l'accessibilité [MAY 84] tandis que pour les séquences maximales infinies, on se ramène à la technique des plus courtes séquences vues au chapitre 4 du premier volume traitant des Réseaux de Petri [HAD 01] ou aussi utilisées dans [RAC 78, YEN 92]. Enfin le test de bisimulation d'un réseau marqué et d'un système de transitions fini devient décidable (ici encore à l'aide du test d'accessibilité) [JAN 99]. Ce résultat est d'autant plus intéressant que très souvent la spécification d'un service est donnée par un tel système et la validation consiste à comparer cette spécification avec le réseau de Petri qui l'implémente.

2. Structures de Kripke et systèmes de transitions

Les structures de Kripke étiquetées décrivent de manière suffisamment générique le comportement des systèmes que l'on désire étudier. Celles-ci sont constituées d'un ensemble d'états pour lesquelles certaines propositions sont vérifiées et d'un ensemble de relations binaires de succession entre états, indicées par les événements du système.

Définition 1 *Une structure de Kripke étiquetée* $\mathcal{SKE} = \langle AP, \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}, \nu\rangle$ *est définie par :*

- AP *est un ensemble de propositions atomiques*
- Σ *est un alphabet fini d'evénements*
- S *est un ensemble d'états*
- $\xrightarrow{a}$ *est une relation binaire* $\subset S \times S$
- $\nu : S \to 2^{AP}$ *est un étiquetage qui associe à chaque état,* $\nu(s)$ *l'ensemble des propositions atomiques vérifiées en* s.

Lorsqu'on étudie une structure de Kripke étiquetée, on la considère le plus souvent munie d'un état initial s_0 ce que l'on note par (SKE, s_0). Lorsqu'on fait abstraction des événements, on a alors affaire à une structure de Kripke. A l'inverse, si l'on fait abstraction des propositions atomiques, on parle de système de transitions étiqueté. Les deux définitions suivantes formalisent ces concepts.

Définition 2 *Une structure de Kripke* $\mathcal{SK} = \langle AP, S, \to, \nu\rangle$ *est définie par :*

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

Définition 3 *Un système de transitions étiqueté* $\mathcal{STE} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *est défini par :*

- Σ *est un alphabet fini d'evénements*
- S *est un ensemble d'états*
- $\xrightarrow{a}$ *est une relation binaire* $\subset S \times S$

Par la suite, nous noterons $s \xrightarrow{a} s'$ *pour indiquer que* $(s, a, s') \in S \times \Sigma \times S$. *Par abus de notation, nous noterons aussi pour* $\sigma \in \Sigma^*$: $s \xrightarrow{\sigma} s'$ *pour indiquer que* s *est accessible à partir de* s *via la séquence d'actions (le mot)* σ.
Les systèmes que nous considérons pouvant être non-déterministes, nous noterons pour $s \in S, E \subset S$ *et* $a \in \Sigma$: $s \xrightarrow{a} E$ *ssi* $E = \{s' \in S : s \xrightarrow{a} s'\}$

Nous noterons enfin $s \stackrel{a}{\not\rightarrow}$, pour indiquer que s n'a pas de successeur par l'action a et $s \not\rightarrow$ pour indiquer que s n'a aucun successeur (i.e., constitue un état de blocage).

Ces structures peuvent être elles aussi initialisées.

3. Logique Temporelle

3.1. *Syntaxe et Sémantique*

Puisqu'on désire vérifier des systèmes dynamiques à événements discrets, exprimons ce qui est commun à tous ces systèmes : les états et la relation de succession entre états. A titre d'exemple, un état d'une application répartie se caractérise par l'état des processus (valeur des variables, compteur ordinal,...) et l'état de l'environnement (e.g. les messages des canaux). Devant la diversité des représentations possibles, on se satisfera d'une abstraction largement suffisante dans la plupart des cas à savoir un ensemble de propositions atomiques (notées $P, Q, \ldots$). A partir d'un état donné, la relation de succession induit un ensemble (le plus souvent infini) de séquences d'états débutant par cet état, encore appelées chemins dans la terminologie de la logique temporelle. Aussi la logique arborescente propositionnelle que nous étudierons $\mathcal{CTL}^*$se définit-elle inductivement par une syntaxe de formules d'état et de chemin [EME 96].

Définition 4 (Syntaxe de $\mathcal{CTL}^*$) *Soit AP un ensemble de propositions atomiques, alors les formules de $\mathcal{CTL}^*$sont définies par les règles suivantes :*

- *S1 Chaque proposition atomique P est une formule d'état.*
- *S2 Si f et g sont des formules d'état alors f **AND** g et **NOT** f sont des formules d'état.*
- *S3 Si f est une formule de chemin alors $\mathbf{E}f$ et $\mathbf{A}f$ sont des formules d'état.*
- *P1 Chaque formule d'état est une formule de chemin.*
- *P2 Si f et g sont des formules de chemin alors f **AND** g et **NOT** f sont des formules de chemin.*
- *P3 Si f et g sont des formules de chemin alors $\mathbf{X}f$ et $f\,\mathbf{U}g$ sont des formules de chemin.*

Seules les règles $S3$, $P1$ et $P3$ appellent des explications. On souhaite raisonner sur les séquences issues d'un état. Ainsi $\mathbf{E}f$ est vérifiée si à partir de cet état il existe une séquence qui vérifie f. $\mathbf{A}f$ est vérifiée si à partir de cet état toutes les séquences vérifient f. Si f est une formule d'état alors f s'interprète aussi comme une formule de chemin qui s'évalue sur le premier état de la séquence. $\mathbf{X}f$ ($\mathbf{X}$ pour "next") consiste à évaluer f sur la sous-séquence privée du

premier état. Enfin $f\mathbf{U}g$ (**U** pour "until") est vérifiée s'il existe un suffixe de la séquence pour laquelle g est vérifiée et telle que tous les suffixes précédents (y compris la séquence initiale) vérifient f. Autrement dit, f reste vérifiée jusqu'à ce que g soit vérifiée et g le sera. Nous formalisons maintenant la sémantique de $\mathcal{CTL}^*$ en introduisant la notion de modèle et de satisfaction de formule par un modèle.

Définition 5 (Modèle de $\mathcal{CTL}^*$) *Un modèle de $\mathcal{CTL}^*$ est une structure de Kripke $\mathcal{SK} = \langle AP, S, \rightarrow, \nu \rangle$ telle que $\rightarrow$ est une relation binaire totale :*
$\forall s \in S, \exists t \in S$ *tel que* $s \rightarrow t$

Traditionnellement la logique temporelle raisonne sur des séquences infinies. En effet, celle-ci s'intéresse particulièrement à des propriétés d'équité qui n'ont de signification que dans ce contexte. Ceci explique la contrainte sur la relation $\rightarrow$. Aussi une séquence $\sigma = (s_0, s_1, \ldots)$ est une suite infinie d'états telle que $\forall i \in \mathbb{N}, s_i \rightarrow s_{i+1}$. Nous reviendrons plus tard sur cette contrainte dans le contexte des réseaux de Petri. La séquence σ^i dénote le suffixe de σ, $(s_i, s_{i+1}, \ldots)$ d'où $\sigma^0 = \sigma$.

Notons $\overline{AP} = \{\mathbf{NOT}\ P \mid P \in AP\}$. Par souci de simplicité, nous considérerons dans la suite que la fonction d'étiquetage ν est à valeurs dans $2^{AP \cup \overline{AP}}$ avec la contrainte évidente que :
$\forall P, s\ |\{P, \mathbf{NOT}\ P\} \cap \nu(s)| = 1$ (une proposition est soit vraie soit fausse)

Définition 6 (Sémantique de $\mathcal{CTL}^*$) *Soit $\mathcal{SK}$ un modèle, s un état de SK et $\sigma = (s_0, s_1, \ldots)$ une séquence de $\mathcal{SK}$, alors la satisfaction d'une formule de $\mathcal{CTL}^*$ sur ce modèle est définie par :*

S1 $\mathcal{SK}, s_0 \models P$ *si et seulement si* $P \in \nu(s_0)$.

S2 $\mathcal{SK}, s_0 \models f$ ***AND*** g *si et seulement si* $\mathcal{SK}, s_0 \models f$ *et* $\mathcal{SK}, s_0 \models g$.
$\mathcal{SK}, s_0 \models$ ***NOT*** f *si et seulement si on n'a pas* $\mathcal{SK}, s_0 \models f$.

S3 $\mathcal{SK}, s_0 \models \boldsymbol{E}f$ *si et seulement si* $\exists \sigma$ *issue de* s_0 *telle que* $\mathcal{SK}, \sigma \models f$.
$\mathcal{SK}, s_0 \models \boldsymbol{A}f$ *si et seulement si* $\forall \sigma$ *issue de* s_0, $\mathcal{SK}, \sigma \models f$.

P1 *Soit f une formule d'état,* $\mathcal{SK}, \sigma \models f$ *si et seulement si* $\mathcal{SK}, s_0 \models f$.

P2 $\mathcal{SK}, \sigma \models f$ ***AND*** g *si et seulement si* $\mathcal{SK}, \sigma \models f$ *et* $\mathcal{SK}, \sigma \models g$.
$\mathcal{SK}, \sigma \models$ ***NOT*** f *si et seulement si on n'a pas* $\mathcal{SK}, \sigma \models f$.

P3 $\mathcal{SK}, \sigma \models f\boldsymbol{U}g$ *si et seulement si* $\exists i$ *tel que* $\mathcal{SK}, \sigma^i \models g$ *et* $\forall j < i\ \mathcal{SK}, \sigma^j \models f$.
$\mathcal{SK}, \sigma \models \boldsymbol{X}f$ *si et seulement si* $\mathcal{SK}, \sigma^1 \models f$.

Dans la pratique, $\mathcal{CTL}^*$ est enrichi d'abréviations qui simplifient l'expression des propriétés :

(OR) f **OR** $g \equiv$ **NOT** (**NOT** f **AND NOT** g)

(true) $true \equiv \mathbf{NOT}\ P\ \mathbf{OR}\ P$

(false) $false \equiv \mathbf{NOT}\ true$

(F) $\mathbf{F}f \equiv true\mathbf{U}f$

(G) $\mathbf{G}f \equiv \mathbf{NOT}\ \ \mathbf{F}\ \ \mathbf{NOT}\ \ f$

(W) $f\mathbf{W}g \equiv f\mathbf{U}g\ \mathbf{OR}\ \mathbf{G}f$

$\mathbf{F}f$ signifie que f sera vérifiée pour un suffixe de la séquence considérée. $\mathbf{G}f$ signifie que f est vérifiée pour tous les suffixes de la séquence considérée. Contrairement à $f\mathbf{U}g$, $f\mathbf{W}g$ ($\mathbf{W}$ pour "weak until") n'implique pas que g soit vérifiée pour un suffixe. Dans ce cas, f reste vérifiée pour tous les suffixes. A titre d'exemple, $\mathbf{G}f \Leftrightarrow f\mathbf{W}false$.

$\mathcal{CTL}^*$ est un langage très expressif. Afin d'obtenir des algorithmes d'évaluation efficaces, on est conduit à restreindre ce langage. Les deux restrictions les plus significatives sont $\mathcal{CTL}$ et $\mathcal{LTL}$.

$\mathcal{CTL}$ est le langage formé des règles syntaxiques $S1$, $S2$, $S3$ et $P0$:

P0 Si f et g sont des formules d'état alors $\mathbf{X}f$ et $f\mathbf{U}g$ sont des formules de chemin.

$\mathcal{CTL}$ se focalise sur la notion d'état. En effet, on peut décrire entièrement la syntaxe sans définir les formules de chemin à l'aide des quatre opérateurs $\mathbf{AX}f$ (pour tout état successeur de l'état considéré, f est vérifiée), $\mathbf{EX}f$ (il existe un état successeur de l'état considéré pour lequel f est vérifiée), $\mathbf{A}f\mathbf{U}g$ (pour toute séquence issue de l'état considéré, f est vérifiée jusqu'à ce que g le soit et g le sera) et $\mathbf{E}f\mathbf{U}g$ (il existe une séquence issue de l'état considéré telle que f est vérifiée jusqu'à ce que g le soit et g le sera). L'intérêt de $\mathcal{CTL}$ réside dans le fait que, d'une part, il est suffisamment expressif pour la spécification de la plupart des propriétés usuelles et que, d'autre part, les méthodes de vérification de la satisfaction d'une formule par un modèle ont une complexité en temps de calcul proportionnelle à la taille du modèle et à la taille de la formule. Cependant certaines propriétés d'équité ne sont pas exprimables en $\mathcal{CTL}$. Cette lacune a conduit à différentes extensions du modèle par des opérateurs tels que $\mathbf{AGF}f$ (pour toute séquence issue de l'état considéré, f est vérifiée par un nombre infini d'états de la séquence) qui permettent d'exprimer des notions usuelles d'équité. Ces extensions disposent aussi de méthodes de vérification de complexité polynomiale. Nous renvoyons le lecteur aux références qui suivent pour plus de précisions sur ce langage [EC 81, EC 82, EH 85].

Exemple
La formule $\mathbf{AG}(\mathbf{A}req\mathbf{U}serv)$ exprime qu'à partir de tout état qui contient une requête, dans toute séquence la requête sera présente jusqu'à ce qu'elle soit servie.

$\mathcal{LTL}$, le langage formé des règles syntaxiques $S1$, $P1$, $P2$ et $P3$ se focalise sur la notion de séquence. En effet, on peut décrire entièrement la syntaxe sans

définir les formules d'état en considérant que les propositions atomiques sont des formules de chemin à vérifier sur le premier état de la séquence. Utiliser une telle logique se justifie en se plaçant du point de vue d'un observateur qui ne peut interagir avec le système. Dans ce cas, seules les séquences sont significatives (on se reportera à la section sur la bisimulation dans le cas d'une interaction possible). Un des intérêts de $\mathcal{LTL}$, illustré par les chapitres suivants est son applicabilité aux techniques d'ordre partiel pour la réduction de la complexité de la vérification. Généralement un modèle $\mathcal{SK}$ est initialisé par un état s_0 et la satisfaction des formules s'évalue sur $\mathcal{SK}, s_0$. Aussi étant donnée une formule $\mathcal{LTL}$ de chemin f, on notera par abus de langage $\mathcal{SK}, s_0 \models f$ pour indiquer que $\mathcal{SK}, s_0 \models \mathbf{A}f$.

Exemple
La formule **GF***p.exec* **OR FG***p.bloq* exprime qu'au cours de toute exécution soit le processus p est indéfiniment bloqué à partir d'un état donné, soit ce processus est choisi une infinité de fois par l'ordonnanceur.

3.2. *Méthodes d'évaluation*

L'objectif d'une méthode d'évaluation est de vérifier si une formule est satisfaite par un modèle particulier. Dans ce paragraphe, nous ne traitons que les modèles finis. L'étude de la vérification des modèles infinis tels que les graphes d'accessibilité de réseaux de Petri non bornés se fera à la fin du chapitre.

Dans la suite $\mathcal{SK}$ désignera le modèle, f_0 la formule à vérifier et s_0 l'état initial du modèle. Le problème à résoudre sera de déterminer si on a $\mathcal{SK}, s_0 \models f_0$.

3.2.1. *Vérification de formules $\mathcal{CTL}^*$*

Nous démontrons d'abord que si on dispose d'une méthode d'évaluation de $\mathcal{LTL}$ alors on peut construire une méthode d'évaluation de $\mathcal{CTL}^*$. Le principe de la construction est relativement simple. Tout d'abord on élimine l'opérateur **E** en le remplaçant par l'expression équivalente **NOT A NOT**. Considérons l'arbre syntaxique d'une formule d'état f_0 de $\mathcal{CTL}^*$:

- Un noeud étiqueté par **A** qui ne comporte pas dans son sous-arbre ce même opérateur **A** préfixe g une formule de $\mathcal{LTL}$.
- On évalue alors g pour tous les états du modèle et on crée une nouvelle proposition $[\mathbf{A}g]$. Cette proposition est affectée aux états du modèle selon le résultat de l'évaluation de g.

- On substitue dans f_0, $\mathbf{A}g$ par $[\mathbf{A}g]$ et on itère le procédé jusqu'à la disparition de l'opérateur $\mathbf{A}$.
- La formule obtenue est alors une formule de logique propositionnelle qui s'évalue localement sur chaque état.

Nous appelons "$\mathcal{CTL}^*$-vérifie" la méthode de vérification recherchée et "$\mathcal{LTL}$-vérifie" la méthode de vérification de formules de $\mathcal{LTL}$. Le texte de la méthode est donné ci-dessous.

$\mathcal{CTL}^*$-vérifie($\mathcal{SK}, s_0, f_0$)
Tant que $\exists f = \mathbf{A}g$ sous-formule de f_0 où $f \in \mathcal{LTL}$**faire**
  Introduire une nouvelle proposition atomique $[f]$
  Pour chaque état s **faire**
    Si $\mathcal{LTL}$-vérifie($\mathcal{SK}, s, g$) **Alors**
      ajouter $[f]$ à $\nu(s)$
    Sinon
      ajouter **NOT** $[f]$ à $\nu(s)$
    Fin si
  Fin pour
  Substituer $[f]$ à f dans f_0
Fin tant que
// f_0 est devenue une formule de logique propositionnelle
Si $s_0 \models f_0$ **Alors**
  renvoyer(VRAI) ;
Sinon
  renvoyer(FAUX) ;
Fin si

Nous appliquons cette méthode à la formule $\mathbf{A}(\mathbf{FAG}P$ **AND** $\mathbf{GAF}Q)$ **AND** R :

- $\mathbf{AG}P$ est une formule du type recherché.
- On évalue sur chaque état $\mathbf{G}P$ et on met à jour en conséquence $[\mathbf{AG}P]$.
- On transforme la formule initiale qui devient :
  $\mathbf{A}(\mathbf{F}[\mathbf{AG}P]$ **AND** $\mathbf{GAF}Q)$ **AND** R.
- Puis la formule est transformée en $\mathbf{A}(\mathbf{F}[\mathbf{AG}P]$ **AND** $\mathbf{G}[\mathbf{AF}Q])$ **AND** R.
- La formule finale est une formule propositionnelle
  $[\mathbf{A}(\mathbf{F}[\mathbf{AG}P]$ **AND** $\mathbf{G}[\mathbf{AF}Q])]$ **AND** R.

3.2.2. *Vérification de formules $\mathcal{LTL}$*

Nous examinons à présent la vérification de formules $\mathcal{LTL}$. Nous procèderons en trois étapes :

- Nous "normalisons" la formule de manière à repousser l'opérateur **NOT** devant les propositions atomiques.
- Nous définissons les automates à promesses qui acceptent des séquences infinies d'un modèle. Puis nous exhibons une construction d'un automate qui accepte exactement les séquences qui vérifient une formule donnée.
- Etant donné un modèle initialisé, nous montrons comment vérifier que ce modèle comporte au moins une séquence acceptée par un automate donné.

La méthode de vérification consiste alors à construire l'automate associé à **NOT** f_0 et vérifier que le modèle $(\mathcal{SK}, s_0)$ ne comporte pas de séquence acceptée par cet automate.

Normalisation d'une formule $\mathcal{LTL}$

Nous normalisons la formule à l'aide de l'opérateur **OR** et de l'opérateur **W** (le "weak until"). La normalisation d'une formule f notée $norm(f)$ repousse l'opérateur **NOT** devant les propositions atomiques. Les équivalences sous-jacentes à la transformation sont faciles à vérifier à partir des définitions (e.g. **NOT** $(f\mathbf{U}g) \Leftrightarrow$ (**NOT** g **AND** f)**W**(**NOT** f **AND NOT** g)).

- $norm(P) = P$, $norm(\mathbf{NOT}\ P) = \mathbf{NOT}\ P$, $norm(\mathbf{X}f) = \mathbf{X}norm(f)$
- $norm(f\ \mathbf{OR}\ g) = norm(f)\ \mathbf{OR}\ norm(g)$
- $norm(f\ \mathbf{AND}\ g) = norm(f)\ \mathbf{AND}\ norm(g)$
- $norm(f\mathbf{W}g) = norm(f)\mathbf{W}norm(g)$, $norm(f\mathbf{U}g) = norm(f)\mathbf{U}norm(g)$
- $norm(\mathbf{NOT\ NOT}\ f) = norm(f)$
- $norm(\mathbf{NOT}\ (f\ \mathbf{AND}\ g)) = norm(\mathbf{NOT}\ f)\ \mathbf{OR}\ norm(\mathbf{NOT}\ g)$
- $norm(\mathbf{NOT}\ (f\ \mathbf{OR}\ g)) = norm(\mathbf{NOT}\ f)\ \mathbf{AND}\ norm(\mathbf{NOT}\ g)$
- $norm(\mathbf{NOT}\ \mathbf{X}f) = \mathbf{X}norm(\mathbf{NOT}\ f)$
- $norm(\mathbf{NOT}\ (f\mathbf{U}g)) = (norm(\mathbf{NOT}\ g)\ \mathbf{AND}\ norm(f))$ $\mathbf{W}(norm((\mathbf{NOT}\ f)\ \mathbf{AND}\ norm(\mathbf{NOT}\ g))$
- $norm(\mathbf{NOT}\ (f\mathbf{W}g)) = (norm(\mathbf{NOT}\ g)\ \mathbf{AND}\ norm(f))$ $\mathbf{U}(norm(\mathbf{NOT}\ f)\ \mathbf{AND}\ norm(\mathbf{NOT}\ g))$

Automates et formules $\mathcal{LTL}$

Nous désirons construire un automate qui reconnaît exactement les séquences infinies qui vérifient une formule (normalisée) de $\mathcal{LTL}$. Cet automate essaie d'établir une preuve basée sur les propositions vérifiée par l'état initial de la séquence σ et sur une formule à vérifier par le suffixe σ^1. Chaque état correspond donc à une formule à vérifier.

Supposons que nous ayons à vérifier la formule $P\mathbf{W}Q$. D'après l'équivalence $f\mathbf{W}g \Leftrightarrow g\ \mathbf{OR}\ (f\ \mathbf{AND}\ \mathbf{X}(f\mathbf{W}g))$,

- soit dans l'état initial Q est vérifiée et σ^1 n'a pas de formule à vérifier,
- soit dans l'état initial P est vérifiée et σ^1 doit vérifier de nouveau $P\mathbf{W}Q$.

Nous obtenons ainsi l'automate représenté sur la figure 2.

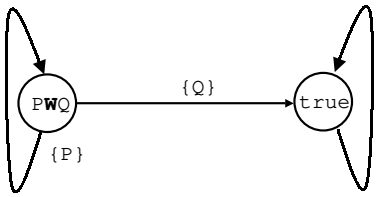


Figure 2: Un automate de reconnaissance de $P\mathbf{W}Q$

Une équivalence similaire s'applique à l'opérateur "until"
$f\mathbf{U}g \Leftrightarrow g$ **OR** (f **AND** $\mathbf{X}(f\mathbf{U}g)$).
Cependant cet automate accepte la séquence où P est indéfiniment vérifiée sans que Q ne le soit. Le point clef est que dans le cas de l'opérateur **U**, on ne peut indéfiniment choisir la deuxième alternative du **OR**. Notons $\mathbf{X}^p$ un opérateur qui est une promesse de vérifier plus tard une formule "until" par la première alternative du **OR**. L'automate de la formule $P\mathbf{U}Q$ est indiqué sur la figure 3. Le point-virgule présent sur l'arc le plus à gauche sépare les propositions à vérifier et les promesses à tenir.

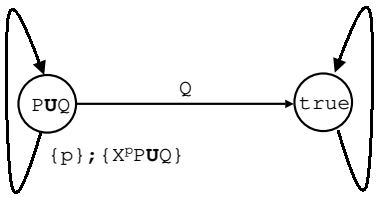


Figure 3: Un automate de reconnaissance de $P\mathbf{U}Q$

Voici maintenant la syntaxe et la sémantique des automates à promesses.

Définition 7 *Un automate à promesses* $A = \langle AP, Q, q_0, Prom, E\rangle$ *est défini par :*

- AP *un ensemble fini de propositions atomiques*
- Q *un ensemble fini d'états*

- $q_0 \in Q$ *l'état initial*
- $Prom$ *un ensemble fini de promesses*
- E *un ensemble fini d'arcs tels que pour* $e \in E$
  - $in(e) \in Q$ *désigne l'origine de l'arc*
  - $out(e) \in Q$ *désigne l'extrémité de l'arc*
  - $etiq(e) \subset AP \cup \overline{AP}$ *désigne les propositions de l'arc*
  - $prom(e) \subset Prom$ *désigne les promesses de l'arc*

Définition 8 *Soit* $\sigma = (s_0, s_1, \ldots, s_n, \ldots)$ *une séquence infinie d'un modèle* $\mathcal{SK}$. σ *est acceptée par* $A = \langle AP, Q, q_0, Prom, E \rangle$ *si et seulement si il existe un chemin* $(q_0, e_0, q_1, e_1, \ldots)$ *tel que :*

- $\forall n, in(e_n) = q_n$, $out(e_n) = q_{n+1}$, $etiq(e_n) \subset \nu(s_n)$
- $\forall n, \forall pr \in prom(e_n), \exists m > n$ *tel que* $pr \notin prom(e_m)$

Une séquence qui vérifie la première condition sera dite reconnue par le chemin.

Nous nous penchons sur la construction d'un automate équivalent à une formule f. Comme nous l'avons vu sur les exemples, on transforme une formule en une disjonction de clauses où chaque clause est une conjonction de propositions atomiques (et de négations de propositions) et de formules à vérifier sur la sous-séquence suivante. Nous notons $tr(f)$ la formule transformée où les formules à vérifier sur la sous-séquence suivante sont remplacées par des propositions (notées comme précédemment entre crochets). L'opérateur $\mathbf{X}^p$ est employé pour l'équivalence appliquée à l'opérateur "until" : il indique une promesse à tenir. Voici la construction de cette formule. On notera que la formule obtenue ne se présente pas syntaxiquement comme une disjonction de clauses conjonctives. Cependant cette transformation syntaxique s'obtient en appliquant itérativement l'équivalence f **AND** $(g$ **OR** $h) \Leftrightarrow (f$ **AND** $g)$ **OR** $(f$ **AND** $h)$. Cette transformation sera effectuée lors de la construction de l'automate.
Si $f = P$ **Alors** $tr(f) = f$
Si $f =$ **NOT** P **Alors** $tr(f) = f$
Si $f = h$ **OR** g **Alors** $tr(f) = tr(h)$ **OR** $tr(g)$
Si $f = h$ **AND** g **Alors** $tr(f) = tr(h)$ **AND** $tr(g)$
Si $f = \mathbf{X}g$ **Alors** $tr(f) = [\mathbf{X}g]$
Si $f = g\mathbf{U}h$ **Alors** $tr(f) = tr(h)$ **OR** $(tr(g)$ **AND** $[\mathbf{X}^p g\mathbf{U}h])$
Si $f = g\mathbf{W}h$ **Alors** $tr(f) = tr(h)$ **OR** $(tr(g)$ **AND** $[\mathbf{X}g\mathbf{W}h])$

La construction de l'automate procède ainsi :

- L'état initial de l'automate est créé et étiqueté avec la formule à vérifier.
- On applique la transformation à la formule décrite ci-dessus. Chaque clause correspond à un arc sortant de l'état. L'extrémité de l'arc est un

noeud étiqueté avec la conjonction de formules de la clause préfixées par un "next". L'arc est étiqueté par les propositions atomiques et les promesses de la clause.

- On itère le procédé jusqu'à ce qu'il n'y ait plus de nouvelle formule. Ce qui arrive nécessairement puisque chaque formule est une conjonction de sous-formules de la formule initiale.
- Pour des raisons de simplicité, on crée préalablement l'état étiqueté par $true$ qui boucle sur lui-même sans proposition ni promesse. Cet état n'est pas nécessairement accessible de l'état initial.

On trouvera ci-dessous une description plus formalisée de l'algorithme. Nous noterons $Aut(f)$, l'automate associé à f.

Créer l'état $(q_{true}, true)$
Créer l'arc e_{true} avec $in(e_{true}) = q_{true}$, $out(e_{true}) = q_{true}$,
$etiq(e_{true}) = \emptyset$, $prom(e_{true}) = \emptyset$
Créer l'état (q_0, f_0)
Insérer (q_0, f_0) dans $Atraiter$
Tant que $Atraiter \neq \emptyset$ **faire**
  Extraire(q, f) de $Atraiter$
  Calculer $tr(f)$
  Exprimer $tr(f)$ sous forme d'une disjonction de clauses conjonctives
  // $tr(f) =$ $\mathbf{OR}_{c \in Cl}$
  Pour chaque clause $c \in Cl$ **faire**
  // $c = \mathbf{AND}_{i \in I} P_i$ $\mathbf{AND}_{j \in J}$ **NOT** Q_j $\mathbf{AND}_{k \in K}[\mathbf{X} f_k]$ $\mathbf{AND}_{l \in L}[\mathbf{X}^p g_l]$
    Si $f' = \mathbf{AND}_{k \in K} f_k$ $\mathbf{AND}_{l \in L} g_l$ étiquette un état **Alors**
      On pose (q', f') cet état
    Sinon
      Créer (q', f')
      Insérer (q', f') dans $Atraiter$
    Fin si
    Créer un arc e avec $in(e) = q$, $out(e) = q'$
    $etiq(e) = \{P_i\}_{i \in I} \cup \{\mathbf{NOT}\ Q_j\}_{j \in J}$, $prom(e) = \{\mathbf{X}^p g_l\}_{l \in L}$
  Fin pour
Fin tant que

Soit $f = Q\mathbf{U}g$ avec $g = (P\ \mathbf{OR}\ \mathbf{X}P)\mathbf{W}R$. Alors :
$tr(f) = tr(g)\ \mathbf{OR}\ (Q\ \mathbf{AND}\ [\mathbf{X}^p f])$
$tr(g) = R\ \mathbf{OR}\ ((P\ \mathbf{OR}\ [\mathbf{X}P])\ \mathbf{AND}\ [Xg]) = R\ \mathbf{OR}\ (P\ \mathbf{AND}\ [Xg])\ \mathbf{OR}\ ([\mathbf{X}P]\ \mathbf{AND}\ [\mathbf{X}g])$

Par conséquent, $tr(f)$ est la disjonction de 4 clauses :

- R qui conduit à l'état étiqueté par $true$ (plus rien à vérifier)
- Q **AND** $[\mathbf{X}^p f]$ qui boucle sur l'état initial. On note que le chemin infini qui suit cet arc n'est pas accepté par l'automate car la promesse $\mathbf{X}^p f$ n'est jamais tenue.

– P **AND** $[\mathbf{X}g]$ qui conduit à l'état étiqueté par g.

– $[\mathbf{X}P]$ **AND** $[\mathbf{X}g]$ qui conduit à l'état étiqueté par P **AND** g

A l'aide de $tr(g)$, le lecteur vérifiera que l'automate construit correspond à la figure 4.

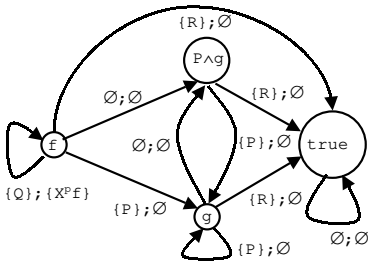


Figure 4: L'automate à promesses de $Q\mathbf{U}((P$ **OR** $\mathbf{X}P)\mathbf{W}R)$

Théorème 9 (Correction de l'automate) *Soit f une formule de $\mathcal{LTL}$, alors les séquences qui vérifient f sont exactement celles acceptées par $Aut(f)$.*

Preuve
Soit cl une clause de $tr(f)$. Par définition, $cl \Rightarrow tr(f)$. Nous définissons inductivement sur la taille de f un ensemble de sous-formules g de f telles que $cl \Rightarrow tr(g)$. Nous notons cet ensemble $dev(cl, f)$.
Si $f = P$ **OR** $f =$ **NOT** P **OR** $f = \mathbf{X}g$ **Alors**
  $dev(cl, f) = \{f\}$
Sinon si $f = h$ **AND** g **Alors**
  $dev(cl, f) = \{f\} \cup dev(cl, g) \cup dev(cl, h)$
Sinon si $f = g$ **OR** h **Alors**
  Si $cl \Rightarrow tr(g)$ **Alors**
    $dev(cl, f) = \{f\} \cup dev(cl, g)$
  Sinon $//cl \Rightarrow tr(h)$
    $dev(cl, f) = \{f\} \cup dev(cl, h)$
  Finsi
Sinon si $f = g\mathbf{U}h$ **Alors**
  Si $cl \Rightarrow tr(h)$ **Alors**
    $dev(cl, f) = \{f\} \cup dev(cl, h)$
  Sinon $//cl \Rightarrow tr(g)$ **AND** $[\mathbf{X}^p g\mathbf{U}h]$
    $dev(cl, f) = \{f\} \cup dev(cl, g)$
  Finsi
Sinon si $f = g\mathbf{W}h$ **Alors**
  Si $cl \Rightarrow tr(h)$ **Alors**
    $dev(cl, f) = \{f\} \cup dev(cl, h)$

Sinon $//cl \Rightarrow tr(g)$ **AND** $[\mathbf{X}g\mathbf{W}h]$
$dev(cl, f) = \{f\} \cup dev(cl, g)$
Finsi
Finsi

Soit $\sigma = (s_0, \ldots, s_i, \ldots)$ une séquence acceptée par un chemin de $Aut(f)$, $(q_0, e_0, \ldots, q_i, e_i, \ldots)$. Posons f_i la formule associée à q_i et cl_i la clause qui produit l'arc e_i. Nous démontrons par récurrence sur la taille de la formule g que $\forall g \in dev(cl_i, f_i)\ \sigma^i \models g$.

Si $g = P$ ou $g = \mathbf{NOT}\ P$ alors g est un terme de cl_i donc $g \in e_i$ ce qui implique que $g \in \nu(s_i)$ donc $\sigma^i \models g$.

Si $g = \mathbf{X}h$ alors $[\mathbf{X}h]$ est un terme de cl_i donc h est un terme de la conjonction constituant f_{i+1}. En appliquant l'hypothèse de récurrence, $\sigma^{i+1} \models h$ ce qui implique $\sigma^i \models \mathbf{X}h$.

Si $g = g_1$ **AND** g_2 alors $\forall k, g_k \in dev(cl_i, f_i)$. En appliquant l'hypothèse de récurrence, $\forall k\ \sigma^i \models g_k$ ce qui implique $\sigma^i \models g$.

Si $g = g_1$ **OR** g_2 alors $\exists g_k \in dev(cl_i, f_i)$. En appliquant l'hypothèse de récurrence, $\exists k\ \sigma^i \models g_k$ ce qui implique $\sigma^i \models g$.

Si $g = g_1\mathbf{U}g_2$ alors

1. Soit $cl_i \Rightarrow tr(g_2)$. D'où $g_2 \in dev(cl_i, f_i)$. En appliquant l'hypothèse de récurrence, $\sigma^i \models g_2$ ce qui implique $\sigma^i \models g$.
2. Soit $cl_i \Rightarrow tr(g_1)$ **AND** $[\mathbf{X}^p g_1 \mathbf{U} g_2]$. D'où $g_1 \in dev(cl_i, f_i), \mathbf{X}^p g \in prom(e_i)$ et g est un terme de la conjonction qui constitue f_{i+1}. En appliquant l'hypothèse de récurrence, $\sigma^i \models g_1$. Puisque g est un terme de la conjonction qui constitue f_{i+1}, nous pouvons appliquer le même raisonnement à σ^{i+1}, σ^{i+2},... jusqu'à ce que la première alternative du raisonnement s'applique à σ^j avec $j > i$. Ce qui arrive nécessairement car sinon $\forall j \geq i, \mathbf{X}^p g \in prom(e_j)$ contredisant l'acceptation de la séquence par le chemin. Nous avons donc $\forall i \leq k < j\ \sigma^k \models g_1$ et $\sigma^j \models g_2$. Par conséquent, $\sigma^i \models g$.

Si $g = g_1\mathbf{W}g_2$ alors

1. Soit $cl_i \Rightarrow tr(g_2)$. D'où $g_2 \in dev(cl_i, f_i)$. En appliquant l'hypothèse de récurrence, $\sigma^i \models g_2$ ce qui implique $\sigma^i \models g$.
2. Soit $cl_i \Rightarrow tr(g_1)$ **AND** $[\mathbf{X}g_1\mathbf{W}g_2]$. D'où $g_1 \in dev(cl_i, f_i)$ et g est un terme de la conjonction qui constitue f_{i+1}. En appliquant l'hypothèse de récurrence, $\sigma^i \models g_1$. Puisque g est un terme de la conjonction qui constitue f_{i+1}, nous pouvons appliquer le même raisonnement à σ^{i+1}, σ^{i+2},... et :
   - soit la première alternative du raisonnement s'applique à une séquence σ^j avec $j > i$. Nous avons donc $\forall i \leq k < j\ \sigma^k \models g_1$ et $\sigma^j \models g_2$. Par conséquent, $\sigma^i \models g$.

– soit $\forall j \geq i, \sigma^i \models g_1$ et par conséquent $\sigma^i \models g$.

Puisque $f = f_0$, $\sigma \models f$.

Supposons maintenant que $\sigma \models f$. Nous construisons un chemin dans $Aut(f)$ qui reconnaît σ. Pour commencer, nous définissons récursivement une clause de $tr(f)$ dépendant de σ :
$cl(f,\sigma) = \mathbf{AND}_{i\in I} P_i \ \mathbf{AND}_{j\in J}\ \mathbf{NOT}\ Q_j \ \mathbf{AND}_{k\in K}[\mathbf{X}f_k] \ \mathbf{AND}_{l\in L}[\mathbf{X}^p g_l]$
telle que :
$\sigma \models \mathbf{AND}_{i\in I} P_i \ \mathbf{AND}_{j\in J}\ \mathbf{NOT}\ Q_j \ \mathbf{AND}_{k\in K}\mathbf{X}f_k \ \mathbf{AND}_{l\in L}\mathbf{X}g_l$.
En voici la définition.
Si $f = P$ **Alors** $cl(f,\sigma) = f$
Sinon si $f = \mathbf{NOT}\ P$ **Alors** $cl(f,\sigma) = f$
Sinon si $f = \mathbf{X}g$ **Alors** $cl(f,\sigma) = [\mathbf{X}g]$
Sinon si $f = g$ **AND** h **Alors** $cl(f,\sigma) = cl(g,\sigma)$ **AND** $cl(h,\sigma)$
Sinon si $f = g$ **OR** h **Alors**
  Si $\sigma \models g$ **Alors**
    $cl(f,\sigma) = cl(g,\sigma)$
  Sinon // $\sigma \models h$
    $cl(f,\sigma) = cl(h,\sigma)$
  Fin si
Sinon si $f = g\mathbf{U}h$ **Alors**
  Si $\sigma \models h$ **Alors**
    $cl(f,\sigma) = cl(h,\sigma)$
  Sinon // $\sigma \models g$ **AND** $\mathbf{X}f$
    $cl(f,\sigma) = cl(g,\sigma)$ **AND** $[\mathbf{X}^p f]$
  Fin si
Sinon si $f = g\mathbf{W}h$ **Alors**
  Si $\sigma \models h$ **Alors**
    $cl(f,\sigma) = cl(h,\sigma))$
  Sinon // $\sigma \models g$ **AND** $\mathbf{X}f$
    $cl(f,\sigma) = cl(g,\sigma)$ **AND** $[\mathbf{X}f]$
  Fin si
Fin si

Soit e l'arc associé à $cl(f,\sigma)$, $q_1 = out(e)$ et f_1 la formule associée à q_1. Par construction on a $prop(e) \subset \nu(s_0)$ et $\sigma \models \mathbf{X}f_1$. Donc $\sigma^1 \models f_1$ et on peut itérer la construction obtenant ainsi un chemin qui reconnaÄt σ. Supposons qu'il existe une promesse $\mathbf{X}^p g\mathbf{U}h$ qui apparaisse sur le chemin en un certain rang i. Par construction de la clause, on a $\sigma^i \models g\mathbf{U}h$ mais dans ce cas il y a un rang $j \geq k$ tel que $\sigma^j \models h$ et par conséquent la clause associée à σ^j ne comporte pas la promesse. Donc ce chemin accepte σ. ♢

D'autres modèles d'automates peuvent représenter les formules de $\mathcal{LTL}$. Ici, nous avons, pour l'essentiel, suivi la démarche décrite dans [COU 99]. Le modèle

le plus répandu est certainement celui des automates de Büchi dont voici la syntaxe et la sémantique [BUC 62]. Pour une étude plus détaillée du lien entre logique temporelle linéaire et automates, on pourra se reporter à [VAR 96].

Définition 10 *Un automate de Büchi* $\mathcal{B} = \langle AP, Q, Q_0, \rightarrow, F \rangle$ *est défini par :*

- AP *un ensemble fini de propositions atomiques*
- Q *un ensemble fini d'états tels que pour* $q \in Q$, $etiq(q) \subset AP \cup \overline{AP}$ *désigne les propositions de l'état*
- $Q_0 \subset Q$ *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*

Définition 11 *Soit* $\sigma = (s_0, s_1, \ldots, s_n, \ldots)$ *une séquence infinie d'un modèle* $\mathcal{SK}$. σ *est acceptée par* $\mathcal{B} = \langle AP, Q, Q_0, \rightarrow, F \rangle$ *si et seulement si il existe un chemin* $(q_0, q_1, \ldots)$ *avec* $q_0 \in Q_0$ *tel que :*

- $\forall n, q_n \rightarrow q_{n+1}$ *et* $etiq(q_n) \subset \nu(s_n)$
- $\exists f \in F$ *t.q.* $\forall n \; \exists m > n \; q_m = f$

On notera que les propositions portent sur les états et non plus sur les transitions, que l'on a un ensemble d'états initiaux et que la condition d'acceptation est définie par un ensemble d'états de succès dont au moins un état doit être atteint une infinité de fois par le chemin. Le pouvoir d'expression des automates de Büchi et des automates à promesses est identique. Il est important de noter que $\mathcal{LTL}$ a un pouvoir d'expression plus restreint que ces modèles d'automates [WOL 83]. Un autre langage de formules (bien moins intuitif) le μ-calcul linéaire a quant à lui un pouvoir d'expression équivalent à ces modèles [DAM 92]. Nous expliquons informellement la traduction d'un automate à promesses en automate de Büchi :

- Supposons que l'on ait n promesses. Pour chaque arc e de l'automate à promesses, on construit $n+1$ états de l'automate de Büchi $\{(q_e, i)\}_{i \in 1 \ldots n+1}$ avec $etiq((q_e, i)) = etiq(e)$.
- Les états initiaux de l'automate sont les $(q_e, 1)$ tels que $in(e) = q_0$.
- Pour $i \leq n$, il y a un arc de (q_e, i) vers $(q_{e'}, i)$ si $out(e) = in(e')$ et si Pr_i appartient à $prom(e')$.
- Pour $i \leq n$, il y a un arc de (q_e, i) vers $(q_{e'}, i+1)$ si $out(e) = in(e')$ et si Pr_i n'appartient pas à $prom(e')$.
- Il y a un arc de $(q_e, n+1)$ vers $(q_{e'}, 1)$ si $out(e) = in(e')$.
- Les états de succès sont les états $\{(q_e, n+1)\}$.

La transformation des arcs en états est usuelle et ne nécessite pas de commentaires particuliers. Lorsqu'au cours de la reconnaissance d'une séquence, on se

trouve dans un état (q_e, i) avec $i \leq n$, on attend que la promesse Pr_i soit tenue. Si elle ne l'est pas dans le prochain état, on passe dans un état de même indice i sinon on passe dans un état indicé par $i+1$. Arrivé dans un état d'indice $n+1$, toutes les promesses ont été tenues au moins une fois et on recommence alors l'examen des promesses. Ainsi si les promesses sont indéfiniment tenues, on passera une infinité de fois par les états d'indice $n+1$ alors que dans le cas contraire on "stagnera" dans un sous-ensemble d'états d'indice $i \leq n$. Nous laissons au lecteur le soin de trouver une transformation d'un automate de Büchi en automate à promesses. La figure 5 illustre la construction. Par souci de lisibilité, on a supprimé les états non accessibles. Les états "blanc" correspondent à l'indice 1, les états "gris foncé" correspondent à l'indice 2 et les états "gris clair" correspondent à l'indice 3. Les états initiaux sont signalés par un arc entrant. Les arcs gras indiquent des transitions entre états de même indice, alors que les arcs fins sont associés à des changements d'indice.

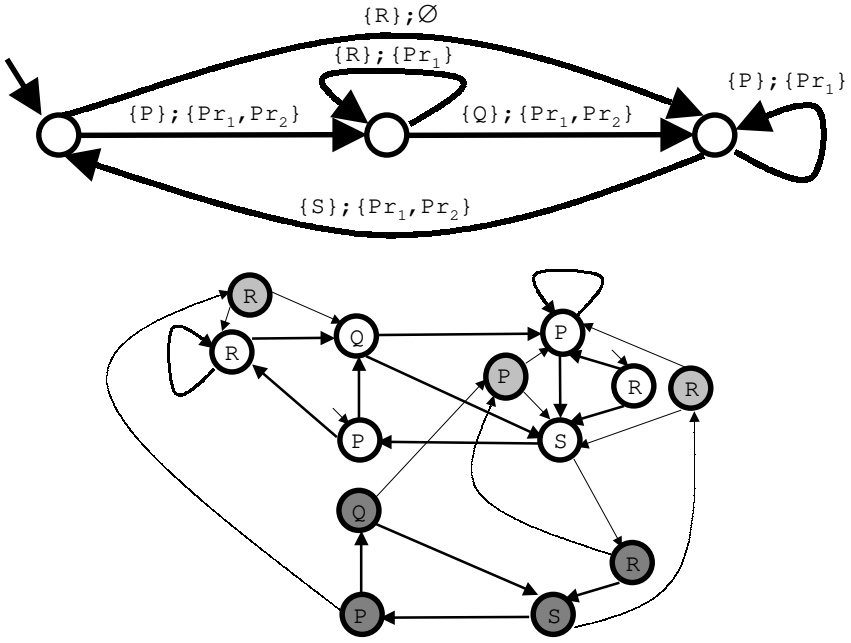


Figure 5: Transformation d'un automate à promesses en automate de Büchi

Existence d'une séquence acceptée par un automate de Büchi

L'existence d'une séquence infinie σ d'un modèle $\mathcal{SK}$ acceptée par un automate de Büchi se vérifie à l'aide d'une construction standard appelée produit synchronisé dont nous donnons ci-dessous la définition suivie de quelques explications.

Définition 12 *Soit $\mathcal{SK}$ un modèle et $\mathcal{B}$ un automate de Büchi, alors $\mathcal{SK} \times \mathcal{B} = (AP', S', \rightarrow', \nu')$ est défini par :*

- $AP' = AP$ *un ensemble fini de propositions atomiques*
- $S' = \{(s,q) | s \in S, q \in Q, etiq(q) \subset \nu(s)\}$
- $(s,q) \rightarrow' (s',q') \Leftrightarrow s \rightarrow s'$ *et* $q \rightarrow q'$
- $\nu'(s,q) = \nu(s)$

De manière évidente, le produit synchronisé engendre les séquences infinies dont la première composante (une séquence infinie de $\mathcal{SK}$) est reconnue par la deuxième composante (un chemin infini de Q). Il nous reste à vérifier si le produit synchronisé contient une séquence infinie débutant par (s_0, q_0) avec $q_0 \in Q_0$ et dont la deuxième composante contient une infinité d'occurrences d'états de F. C'est l'objet du théorème suivant. Une composante fortement connexe (c.f.c.) d'un graphe est élementaire si le sous-graphe associé à cette c.f.c est un unique sommet sans boucle (autrement dit, il n'est pas possible de construire un chemin infini dans cette c.f.c).

Théorème 13 *Soit $\mathcal{SK}$ un modèle fini, s_0 un état de $\mathcal{SK}$ et $\mathcal{B}$ un automate de Büchi alors :*
$\exists\, \sigma = (s_0, s_1, \ldots)$ séquence $\mathcal{SK}$ acceptée par $B \Leftrightarrow \exists\, \mathcal{C}$ une c.f.c. non élémentaire de $\mathcal{SK} \times B$ accessible d'un $(s_0, q_0) \in S'$ avec $q_0 \in Q_0$ contenant un état (s, f) avec $f \in F$

Preuve
Soit $\sigma = (s_0, s_1, \ldots)$ une séquence de $\mathcal{SK}$ acceptée par $(q_0, q_1, \ldots)$ un chemin de $\mathcal{B}$ alors par construction $((s_0, q_0), (s_1, q_1), \ldots)$ est une séquence de $\mathcal{SK} \times \mathcal{B}$ qui rencontre une infinité de fois des états (s, f) avec $f \in F$. Puisque $\mathcal{SK} \times \mathcal{B}$ est fini, l'un de ces états (noté (s^*, f^*)) est atteint une infinité de fois par la séquence. Puisque de (s^*, f^*) on atteint de nouveau cet état par une séquence non nulle, la c.f.c. contenant (s^*, f^*) est non élementaire. Puisque le premier état de la séquence est (s_0, q_0) cette c.f.c. est accessible de (s_0, q_0).

Si le membre gauche de l'équivalence est vérifié, alors il existe une séquence finie σ_1 de (s_0, q_0) vers (s, f) et une séquence finie non nulle σ_2 de (s, f) vers lui-même. Par conséquent, $\sigma = \sigma_1.\sigma_2^\infty$ est une séquence infinie dont la deuxième composante (un chemin dans $\mathcal{B}$) accepte la première (une séquence de $\mathcal{SK}$). $\diamondsuit$

Ce résultat nous fournit un moyen efficace de vérification : une fois construit le produit synchronisé, on calcule les c.f.c. au moyen de l'algorithme de Tarjan [AHO 74] et on les examine. La taille du produit synchronisé est proportionnelle aux tailles du modèle et de la formule. L'algorithme, quant à lui, opère en un temps polynomial en fonction de la taille du produit synchronisé.

Cependant cette efficacité n'est qu'apparente. D'une part, la taille du modèle d'exécution est très grande relativement à la taille du modèle de spécification (e.g. taille du graphe d'accessibilité versus taille du réseau de Petri). D'autre part, la taille de l'automate peut être une fonction exponentielle de la taille de la formule. Ce dernier point n'est pas aussi critique car les formules sont généralement de taille très réduite. Aussi pour remédier à ces problèmes de complexité, différentes techniques ont été proposées. En amont, on cherche à vérifier la formule sur un modèle d'exécution plus petit mais équivalent (voir les chapitres suivants). En aval, on cherche à vérifier une formule sans développer complètement le produit synchronisé par des méthodes "dites à la volée" [GOD 93, GER 95] ou encore à réduire la taille de la représentation par des structures de données de type BDD (diagrammes de décision binaire) [BRY 86].

3.3. *Logique temporelle et réseaux de Petri*

La spécification de formules de logique temporelle propositionnelle de réseau de Petri implique la définition de propriétés atomiques. Puisque nous évaluons les formules de logique temporelle sur le graphe d'accessibilité, un état du modèle est un marquage accessible. Aussi, à toute expression booléenne dont le domaine est l'ensemble des marquages du réseau convient. Dans la pratique, les expressions utilisées s'évaluent facilement. On notera p pour le marquage de p dans l'état courant. Voici quelques exemples de formules fréquentes.

- Deux places p_1, p_2 sont mutuellement exclusives : $\mathbf{AG}(p_1 \cdot p_2 = 0)$
- Quelque soit le marquage atteint, une place p sera inévitablement marquée : $\mathbf{AGAF}(p > 0)$
- Quelque soit le marquage atteint, on peut toujours marquer une place p : $\mathbf{AGEF}(p > 0)$
- Au cours de toute séquence d'exécution, une place p est indéfiniment marquée et démarquée $\mathbf{AG}(\mathbf{F}(p > 0)\ \mathbf{AND}\ \mathbf{F}(p = 0))$
- Une transition t est vivante (toujours franchissable dans le futur de n'importe quel état) : $\mathbf{AGEF}\ \mathbf{AND}_{p \in P}\ (p \geq Pré(p, t))$

Comme l'illustre le dernier exemple, il est possible de raisonner sur la franchissabilité d'une transition. Cependant ce n'est pas le cas du franchissement proprement dit car il nécessiterait d'évaluer l'évolution du marquage des places entre deux états successifs. Aussi on étend le langage $\mathcal{CTL}^*$ en considérant l'opérateur $\mathbf{X}_{\{e\}}$ dont la sémantique est définie par :
$\sigma \models \mathbf{X}_{\{e\}} f$ si et seulement si $\mathcal{SK}, \sigma^1 \models f$ et la première transition de σ est étiquetée par e

Dans ce paragraphe, on considère qu'une transition de réseau de Petri n'est jamais étiquetée par le mot vide. Les méthodes de vérification décrites plus

haut s'étendent de manière immédiate à cette nouvelle logique qui est à la fois propositionnelle et événementielle. Supposons que l'étiquetage d'un réseau soit l'identité. Nous pouvons maintenant exprimer le fait qu'une transition t soit indéfiniment franchie dans toute séquence : $\mathbf{AGFX}_{\{t\}}true$.

Lorsque nous étudierons la décidabilité de la vérification de formules de logique temporelle sur un réseau de Petri quelconque, nous distinguerons les cas suivants :

- La logique propositionnelle $\mathcal{CTL}^*$(et ses fragments) en interdisant ces nouveaux opérateurs.
- La logique événementielle $\mathcal{CTL}^*$(et ses fragments) si les seules propositions atomiques sont $true$ et $false$.

La sémantique de la logique temporelle s'appuie sur les séquences infinies mais le modélisateur désire aussi raisonner sur les séquences finies. Par exemple, on souhaite savoir si une place p est toujours marquée dans un marquage mort. La formule $\mathbf{AG}(\mathbf{AND}_{t\in T}\mathbf{NOT}\ \mathbf{X}_{\{t\}}true \Rightarrow p > 0)$ qui semble convenir est incorrecte car c'est en réalité une tautologie. En effet, une séquence infinie ne satisfait jamais $\mathbf{AND}_{t\in T}\mathbf{NOT}\ \mathbf{X}_{\{t\}}true$.

Pour prendre en compte ces besoins de vérification, il est nécessaire de distinguer le type de séquence étudiée et d'introduire une sémantique de satisfaction adéquate. Puisque nous traitons les séquences, nous considérons que les formules de chemin sont représentées par un automate tel que les arcs de cet automate sont étiquetés par des étiquettes de transition de réseau de Petri. Afin de ne pas alourdir la présentation par une énumération de tous les cas possibles, nous nous limitons à une logique linéaire événementielle définie au moyen d'automates de Büchi étiquetés.

Définition 14 *Un automate de Büchi étiqueté* $\mathcal{BE} = \langle \Sigma, Q, Q_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F\rangle$ *est défini par :*

- Σ *un alphabet fini*
- Q *un ensemble fini d'états*
- $Q_0 \subset Q$ *le sous-ensemble des états initiaux*
- $\xrightarrow{a}$ *est une relation binaire* $\subset S \times S$
- $F \subset Q$ *un sous-ensemble d'états de succès*

Définition 15 *Soit* $\sigma = (t_1, t_2, \ldots, t_n, \ldots)$ *une séquence infinie d'un réseau de Petri* R, σ *est acceptée par* $\mathcal{BE} = \langle \Sigma, Q, Q_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F\rangle$ *si et seulement si il existe un chemin* $(q_0, q_1, \ldots)$ *avec* $q_0 \in Q_0$ *tel que :*

- $\forall n, q_n \xrightarrow{l(t_{n+1})} q_{n+1}$
- $\exists f \in F$ *t.q.* $\forall n\ \exists m > n\ q_m = f$

Les autres types de séquence qui intéressent le modélisateur sont les séquences finies et les séquences finies maximales (i.e. qui se terminent par un marquage mort). On recherche alors un chemin dans l'automate qui s'achève par un état de succès. Une deuxième manière d'aborder le problème des séquences maximales finies dans le cas des réseaux de Petri (bornés) consiste à compléter le graphe d'accessibilité (fini) en ajoutant une boucle à tous les marquages morts, étiquetée par une action spéciale. Ainsi toute séquence maximale de ce nouveau graphe est infinie et celles qui sont prolongées artificiellement se reconnaissent par l'occurence de l'action spéciale.

Définition 16 *Soit $\sigma = (t_1, t_2, \ldots, t_f)$ une séquence finie (éventuellement maximale) d'un réseau de Petri R, σ est acceptée par $\mathcal{BE} = \langle \Sigma, Q, Q_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F\rangle$ si et seulement si il existe un chemin $(q_0, q_1, \ldots, q_f)$ avec $q_0 \in Q_0$ et $q_f \in F$ tel que : $\forall n < f, q_n \xrightarrow{l(t_{n+1})} q_{n+1}$*

Autoriser l'étiquetage d'une transition par le mot vide (i.e. une transition non observable) complique grandement la sémantique de la satisfaction et introduit le problème de la divergence. Une séquence divergente est une séquence infinie dont un suffixe est composé exclusivement de transitions non observables. Il importe de traiter spécifiquement ce type de séquence à la fois du point de vue de la sémantique et de celui de la vérification. Ce problème sera évoqué dans le contexte de l'approche comportementale.

4. Approche Comportementale

De nombreuses relations d'équivalence ont été utilisées ou spécifiquement proposées pour la comparaison et l'analyse de systèmes concurrents depuis l'équivalence de traces ou de langages [AHO 74] à l'équivalence observationnelle [MIL 89] en passant par les modèles de refus et les équivalences de test [LED 90, BRI 88]. Voir [DE 87, ARN 92, OH 86, GLA 90] pour un panorama des relations d'équivalences existantes.

Cette explosion s'explique d'une part, par la difficulté de définir formellement une sémantique universelle des systèmes de processus [ARN 92] et par la variété des propriétés spécifiques des systèmes étudiés ou des points de vue que l'on peut adopter pour conduire leur analyse : vérification ou test. Dans cette section, nous nous limiterons à l'aspect vérification.

Contrairement à l'approche logique, l'approche comportementale privilégie les informations associées aux labels d'actions et oublie le plus souvent les informations associées aux états. La structure prise en compte par l'analyse comportementale est un système de transitions étiqueté (cf définition 3).

Avant d'aller plus avant dans la formalisation, nous allons tenter d'illustrer différents points de vue susceptibles d'être pris en compte. L'exemple choisi est le fonctionnement simplifié d'une machine à café : le consommateur introduit une pièce dans le monnayeur, il choisit ensuite sa boisson en appuyant sur le bouton associé.

Les systèmes de transitions ci-dessous, $\langle D, 0\rangle, \langle D', 0'\rangle$ et $\langle D'', 0''\rangle$, représentent chacun un fonctionnement plus ou moins "semblable" au distributeur de boissons que nous venons de décrire. L'approche comportementale à travers les différentes équivalences de comportements qui ont été proposées dans la littérature va nous permettre de formaliser différentes notions de "similarité".

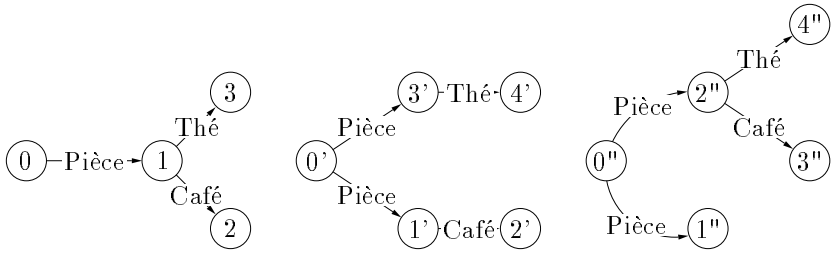


Figure 6: Trois Distributeurs de Boissons : $\langle D, 0\rangle, \langle D', 0'\rangle$ et $\langle D'', 0''\rangle$

A la manière de ce qui a été présenté dans le chapitre 3 du volume sur les réseaux de Petri [HV 01], on peut associer à tout STEI un langage.

Définition 17 *Langage associé à un STEI*

Soit $\langle \mathcal{STE}, s_0\rangle$ *un STEI avec* $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$,

$L(\langle \mathcal{STE}, s_0\rangle) =_{Def} \{\sigma \in \Sigma^* \ : \exists s' \in S \text{ tel que } \ s_o \stackrel{\sigma}{\rightarrow} s'\}$

Contrairement aux automates à états finis, les STEI peuvent comporter une infinité d'états, ils comportent un seul état initial et n'introduisent pas la notion d'état terminal [AHO 74]. Tout état du STEI est donc considéré comme un état terminal et le langage reconnu par le STEI est fermé par préfixe : tout préfixe d'un mot reconnu est lui-même reconnu.

Définition 18 *Equivalence langage*

La notion de langage précédemment introduite nous permet de définir une première notion d'équivalence entre deux systèmes de transitions basée sur l'égalité de leurs langages respectifs.

Soient $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *et* $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ *deux systèmes de transitions,* s_0 *et* s'_0 *leurs états initiaux respectifs :*

$\langle \mathcal{STE}, s_0 \rangle \equiv \langle \mathcal{STE}', s'_0 \rangle$ *ssi* $L(\langle \mathcal{STE}, s_0 \rangle) = L(\langle \mathcal{STE}', s'_0 \rangle)$

Langage : Un premier critère de comparaison de ces distributeurs nous est fourni par l'étude de leur langage.

Ici $L(\langle D, 0 \rangle) = L(\langle D', 0' \rangle) = L(\langle D'', 0'' \rangle) = \{\epsilon, \text{Pièce, Pièce.Café, Pièce.Thé }\}$ et pour ce critère ces trois STE sont équivalents. En particulier, du point de vue de l'exploitant du distributeur, chacun de ceux-ci n'offre une boisson que si celle-ci a été réglée.

Traces maximales : Le critère précédent fait abstraction de la possibilité de blocage, il confond ces 3 STE alors que ceux-ci présentent des blocages différents. La notion de trace maximale permet de prendre en compte cet aspect. On considère toujours les STE comme des accepteurs de langages, mais maintenant seules sont "reconnues" les séquences maximales : les séquences infinies ou les séquences aboutissant sur des états sans successeur.

Définition 19 *Traces Maximales*

Pour $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$ *et un STEI* $\langle \mathcal{STE}, s_0 \rangle$*, on associe* $L_{Max}(\langle \mathcal{STE}, s_0 \rangle)$ *l'ensemble de ses traces maximales définies par :*

$$\begin{aligned} L_{Max}(\langle \mathcal{STE}, s_0 \rangle) \quad &=_{Def} \ (L(\langle \mathcal{STE}, s_0 \rangle) \cap \Sigma^{\infty}) \\ &\cup \ \{\sigma \in L(\langle \mathcal{STE}, s_0 \rangle) \ : \exists s' \in S \text{ tel que } \ s \stackrel{\sigma}{\rightarrow} s' \text{ et } s' \not\rightarrow\} \end{aligned}$$

Définition 20 *Equivalence des traces maximales*

Muni de cette notion de trace maximale, nous définissons la relation d'équivalence associée ainsi :

Soient $\langle \mathcal{STE}, s_0 \rangle$ *et* $\langle \mathcal{STE}', s'_0 \rangle$ *deux systèmes de transitions,* s_0 *et* s'_0 *leurs états initiaux respectifs :*

$\langle \mathcal{STE}, s_0 \rangle \equiv_{Max} \langle \mathcal{STE}', s'_0 \rangle$ *ssi* $L_{Max}(\langle \mathcal{STE}, s_0 \rangle) = L_{Max}(\langle \mathcal{STE}', s'_0 \rangle)$

Remarque Langage et Traces Maximales

La définition 19 est purement dénotationnelle, d'un point de vue opérationnel la notion de "traces maximales" peut être exprimée à partir de la notion de langage en "complétant" le STEI (ou l'automate) en ajoutant : un état $\perp$ à l'ensemble des sommets ($\perp \notin S$), un label *echec* à l'alphabet Σ ($echec \notin \Sigma$) et, enfin, en reliant tout état de blocage à l'état $\perp$ par une transition étiquetée par *echec*.

Propriété 21 *Langage et Traces Maximales*

Pour $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$*, on définit* $Max(\mathcal{STE})$ *ainsi :*

$Max(\mathcal{STE}) =_{Def} \mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma'} \rangle$

$$où : \begin{cases} \Sigma' = \Sigma \cup \{echec\}, \\ S' = S \cup \{\bot\}, \\ \overset{a'}{\{\rightarrow'_{a\in\Sigma'}\}} =_{Def} \overset{a}{\{\rightarrow_{a\in\Sigma}\}} \cup \{ \ s \overset{echec}{\rightarrow} \bot \ : s \in S \ tel\ que \ s \not\rightarrow\} \end{cases}$$

$L_{Max}(\langle \mathcal{STE}, s_0\rangle) = L_{Max}(\langle \mathcal{STE}', s'_0\rangle)$ *ssi*
$L(\langle Max(\mathcal{STE}), s_0\rangle) = L(\langle Max(\mathcal{STE}'), s'_0\rangle)$

On a maintenant, $L_{Max}(\langle D, 0\rangle) = L_{Max}(\langle D', 0'\rangle) = \{$Pièce.Café, Pièce.Thé$\}$ par contre $L_{Max}(\langle D'', 0''\rangle) = \{$Pièce, Pièce.Café, Pièce.Thé$\}$. Suivant ce nouveau critère, seuls D et D' restent équivalents. Si l'on prend en compte maintenant le point de vue du client, il est effectivement important d'isoler le distributeur D'' qui peut accepter une pièce sans délivrer de boisson. Toujours du point de vue du client, ne pas pouvoir distinguer D et D' n'est pas acceptable : D laisse le choix de la boisson à l'utilisateur tandis que D' choisit la boisson à sa place.

Refus & Acceptation : L'équivalence de traces maximales prend en en compte les blocages "globaux", les équivalences basées sur la notion de refus, ou dualement d'acceptation, permettent de prendre en compte la notion de blocages partiels et en particulier la possibilité de "refuser" d'exécuter une action. Ainsi, on peut considérer, en plus des séquences admises, la possibilité de refuser ou d'accepter une action.

Définition 22 *Eléments de base des sémantiques de refus [GLA 90, LED 90]*

Soit $\langle \mathcal{STE}, s_0\rangle$ un STEI, pour $s \in S, \sigma \in \Sigma^*$ et $A \subset \Sigma$, on note :

1. $s \ ref\ A \Leftrightarrow_{Def} \forall a \in A, s \overset{a}{\not\rightarrow}$
2. $s \models after\ \sigma \Leftrightarrow_{Def} \{s' \in S \ : \ s \overset{\sigma}{\rightarrow} s'\}$
3. $s \models after\ \sigma\ ref\ A \Leftrightarrow_{Def} \exists s' \in$ "$s\ after\ \sigma''$ *tel que* $s'\ ref\ A$
4. $\mathcal{STE} \models after\ \sigma\ ref\ A \Leftrightarrow_{Def} s_0 \models after\ \sigma\ ref\ A$

(1) permet de définir des blocages partiels à travers l'ensemble de refus que l'on peut associer à un sommet. (2) désigne le sous-ensemble des sommets accessibles à partir du sommet s via la séquence σ. (3) stipule que "à partir du sommet s, il est possible, via la séquence σ, d'atteindre un sommet qui refusera toutes les actions de A.

Définition 23 *Relation de Conformité [BRI 88, LED 90] Considérons deux STEI $\langle \mathcal{STE}, s_0\rangle$ et $\langle \mathcal{STE}', s'_0\rangle$ et L la réunion de leurs alphabets respectifs ($L = \Sigma \cup \Sigma'$)*

$$\mathcal{STE} \ \underline{conf} \ \mathcal{STE}' \Leftrightarrow_{Def} \begin{cases} \forall \sigma \in L(\langle \mathcal{STE}, s_0\rangle), \forall A \subset L : \\ \quad Si\ \mathcal{STE}\ after\ \sigma\ ref\ A\ alors\ \mathcal{STE}'\ after\ \sigma\ ref\ A \end{cases}$$

Informellement, une implémentation $\mathcal{STE}$ "est conforme" à la spécification $\mathcal{STE}'$ lorsque : pour toute séquence σ du comportement spécifié, si l'implémentation peut évoluer par σ alors les ensembles d'action A qu'elle peut refuser d'exécuter après cette évolution sont au plus ceux que la spécification peut refuser après σ [DRI 92].

Définition 24 *Equivalence de test [BRI 88]*

$$\mathcal{STE} \underline{te}\ \mathcal{STE}' \Leftrightarrow_{Def} \begin{cases} L(\langle \mathcal{STE}, s_0 \rangle) = L(\langle \mathcal{STE}', s'_0 \rangle) \\ \mathcal{STE}\ \underline{conf}\ \mathcal{STE}'\ et\ \mathcal{STE}'\ \underline{conf}\ \mathcal{STE} \end{cases}$$

Pour ce dernier point de vue, qui fonde les sémantiques basées sur le refus, les STE D et D' ne sont pas "testing-equivalent" (non D $\underline{\text{te}}$ D'). En effet D' peut refuser l'action Café ou l'action Thé après avoir exécuté l'action Pièce tandis que D après Pièce offrira toujours la possibilité d'obtenir du Thé ou du Café. En reprenant les éléments de terminologie de la définition 22, on obtient par exemple :

$0'\ after$ Pièce ref {Thé, Pièce} et $0'\ after$ Pièce ref {Café, Pièce} tandis que la seule action refusée à partir de 0 est Pièce, i.e., $0\ after$ Pièce ref {Pièce}.

Nous ne développerons pas plus avant ces sémantiques dites de refus (failure semantics), mais nous invitons le lecteur intéressé à se reporter à [ARN 92] où le chapitre 8 est consacré aux différentes équivalences de traces.

4.1. *Relations de Bisimulation*

Les trois équivalences que nous venons d'évoquer adoptent un point de vue "linéaire" au sens où elles privilégient les séquences d'exécutions du STE et font abstraction de la structure arborescente de celui-ci. Pour illustrer notre propos, considérons maintenant une machine à café où la seule boisson disponible est le café $\underline{\text{sucré}}$: le client introduit une pièce et doit obtenir le café puis le sucre.

Les machines représentées figure 7 sont indiscernables pour les sémantiques de refus et les équivalences de test qui leur sont associées [BRI 88]. Les deux machines peuvent refuser le sucre après avoir délivré la boisson. Pour $s \in \{0, 0'\}$ on a :
$s\ after$ Pièce ref {Pièce, Sucre} et $s\ after$ Pièce.Café ref {Pièce, Café, Sucre}

Du point de vue du client ces deux machines sont donc tout autant imparfaites. Un client pouvant expérimenter, aussi longtemps qu'il le veut ces deux machines, est incapable de les distinguer. Pour chacune de celles-ci, certaines expérimentations conduiront à obtenir un café sucré et d'autres à un café sans sucre.

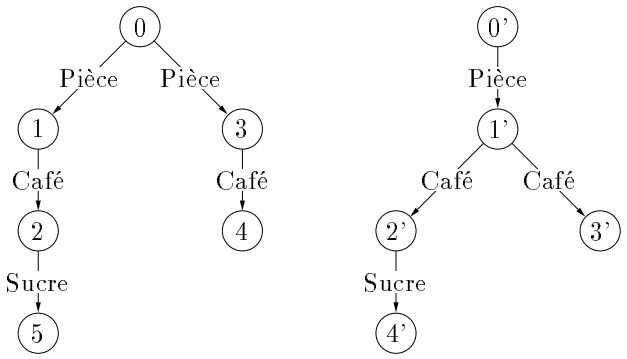


Figure 7: Deux machines à Café : $\langle M, 0\rangle$ et $\langle M', 0'\rangle$

Du point de vue de l'analyse et en particulier si l'on cherche à comprendre pourquoi on ne peut garantir au client qu'il obtiendra une boisson sucrée il est pourtant intéressant de les distinguer : dans le premier cas, l'absence de sucre sera consécutive du non-déterminisme associé à l'action Pièce tandis que dans le second il résultera du non-déterminisme associé à l'action Café. La notion de bisimulation qui prend en compte la structure arborescente du STE, et non plus uniquement sa structure linéaire, va nous permettre de distinguer ces deux machines.

La notion de Bisimulation, introduite par Park [PARK 81] est à la base de nombreuses relations d'équivalences utilisées pour la vérification de systèmes communicants.

Définition 25 *Relation de Bisimulation*

Soit un STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$*, une relation binaire* B*,* $(B \subset S \times S)$ *est une relation de bisimulation ssi elle vérifie :*

$\forall (p,p') \in B$ *et* $\forall t \in \Sigma$:

$$[\forall q \in S \ Si \quad p \stackrel{t}{\rightarrow} q \ \ alors \ \exists q' \in S \ : \quad p' \stackrel{t}{\rightarrow} q' \ \ et \ (q,q') \in B]$$
$$et \quad [\forall q' \in S \ Si \quad p' \stackrel{t}{\rightarrow} q' \ \ alors \ \exists q \in S \ : \quad p \stackrel{t}{\rightarrow} q \quad et \ (q',q) \in B]$$

Deux sommets $s_1, s_2 \in S$ *sont en bisimulation ssi il existe une relation de bisimulation* B *telle que* $(s_1, s_2) \in B$*.*

Définition 26 *Bisimulation entre systèmes de transitions*

Soit deux STE, $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *et* $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ *tels que* $S \cap S' = \emptyset$ *et pour lesquels on pose* $\mathcal{S} = S \cup S'$

Une relation binaire B, ($B \subset \mathcal{S} \times \mathcal{S}$) est une relation de bisimulation entre $\mathcal{STE}$ et $\mathcal{STE}'$ ssi elle vérifie :

$\forall (p,p') \in B$ *et* $\forall t \in \Sigma \cup \Sigma'$:

$$[\forall q \in \mathcal{S}\ Si \quad p \xrightarrow{t} q \ alors\ \exists q' \in \mathcal{S} \ : \quad p' \xrightarrow{t} q' \ et\ (q,q') \in B]$$
$$et \quad [\forall q' \in \mathcal{S}\ Si \quad p' \xrightarrow{t} q' \ alors\ \exists q \in \mathcal{S} \ : \quad p \xrightarrow{t} q \quad et\ (q',q) \in B]$$

Définition 27 *Systèmes de transitions bisimilaires*

La définition précédente s'étend de façon canonique aux systèmes de transitions initialisés en posant que deux STEI $\langle \mathcal{STE}, s_0 \rangle$ et $\langle \mathcal{STE}', s'_0 \rangle$ sont en bisimulation (ou sont bisimilaires) ssi il existe une relation de bisimulation reliant leurs états respectifs initiaux. i.e.

$\langle \mathcal{STE}, s_0 \rangle$ et $\langle \mathcal{STE}', s'_0 \rangle$ sont en bisimulation ssi $\exists$ une bisimulation $B \subset \mathcal{S} \times \mathcal{S}$ entre $\mathcal{STE}$ et $\mathcal{STE}'$ telle que $(s_0, s'_0) \in B$

Exemple 1

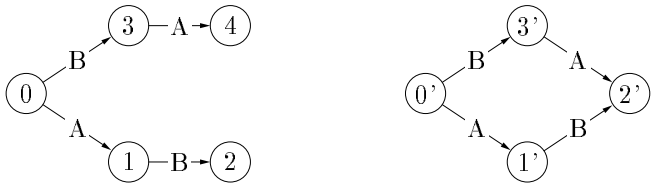


Figure 8: Deux STE Bisimilaires : $\langle E, 0 \rangle, \langle E', 0' \rangle$

Les STEI $\langle E, 0 \rangle, \langle E', 0' \rangle$, représentés Fig 8, sont en bisimulation par la relation $B = \{(0,0')(1,1'),(2,2')(3,3')(4,2')\}$.

Les STEI $\langle D, 0 \rangle, \langle D', 0' \rangle$ et $\langle D'', 0'' \rangle$, associés aux distributeurs de boissons représentés Fig 6 ne sont pas bisimilaires. Montrons par exemple que $\langle D, 0 \rangle$ et $\langle D', 0' \rangle$ ne sont pas en bisimulation. Procédons par l'absurde et supposons l'existence d'une bisimulation B entre D et D' avec $(0,0') \in B$. Comme $0 \xrightarrow{\text{Pièce}} 1$, on doit avoir $(1,1') \in B$ ou $(1,3') \in B$. $(1,1') \in B$ est impossible car $1 \xrightarrow{\text{Thé}}$ et $1' \not\xrightarrow{\text{Thé}}$. De même $(1,3') \in B$ entraîne une contradiction car $1 \xrightarrow{\text{Café}}$ et $3' \not\xrightarrow{\text{Café}}$.

Les STEI $\langle M, 0 \rangle, \langle M', 0' \rangle$, associés aux machines à café représentées Fig 7, ne sont pas bisimilaires. Même si ces STEI sont petits, il devient vite difficile de montrer "à la main" à partir de la définition dénotationnelle de la bisimulation

(cf def 25) l'existence ou la non existence d'une bisimulation.
La section 4.1.1 donne un algorithme permettant de décider de la bisimulation qui nous permettra de statuer sur ces deux systèmes.
La section 4.3.4, en introduisant la logique d'Hennesy-Milner, nous donnera les éléments de langage permettant de distinguer sans ambiguïté ces deux systèmes et de montrer qu'ils ne sont pas bisimilaires.

Propriété 28 *Propriétés des bisimulations : [ARN 92]*

- *La relation inverse d'une bisimulation est aussi une bisimulation.*
- *La composée de 2 bisimulations est une bisimulation.*
- *La réunion de 2 bisimulations est une bisimulation.*

Les propriétés ci-dessus permettent de définir une relation de bisimulation spécifique, l'équivalence forte qui est la bisimulation la plus large.

Définition 29 *Equivalence Forte*

L'équivalence forte, notée $\sim$*, est définie par*

$p \sim q \Leftrightarrow_{Def}$ *il existe une bisimulation* B *telle que* $(p,q) \in B$

$\sim$ *est réflexive car l'identité est une bisimulation. La symétrie et la transitivité proviennent respectivement du fait que l'ensemble des bisimulations est stable respectivement par inversion et par composition.*

4.1.1. *Algorithme de Décision de la Bisimulation*

Cette section montre comment construire, si elle existe, une relation de bisimulation à partir de tout système de transition finitaire. La propriété 32 fonde l'algorithme de décision. Nous présentons aussi quelques notions et propriétés élémentaires qui permettent de "compacter" les calculs.

Définition 30 *STE finitaire*

Un STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *est dit finitaire, ou de façon équivalente, possède une relation d'accessibilité à image finie ssi :*

$\forall s \in S$ *et* $\forall a \in \Sigma$*, l'ensemble* $\{q \in S$ *tel que* $s \stackrel{a}{\rightarrow} q\}$ *est fini.*

Définition 31 $\sim_n$ *équivalences*

Pour un STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$*, on considère la suite de relations indexées pas i, notées* $\sim_i$*, suivantes :* $\forall p, q \in S$

- $p \sim_0 q$

– $p \sim_{n+1} q$ *ssi* $\forall a \in \Sigma$

$\forall p' \in S \quad p \xrightarrow{a} p' \Rightarrow \exists q' \in Q \ : \ q \xrightarrow{a} q'$ *tel que* $p \sim_n p'$

$\forall q' \in S \quad q \xrightarrow{a} q' \Rightarrow \exists p' \in Q \ : \ p \xrightarrow{a} p'$ *tel que* $q \sim_n q'$

Intuitivement, on teste la $\sim_n$-équivalence entre deux systèmes comme suit. Pour chaque système, on construit l'arbre des séquences de longueur inférieure ou égale à n (obtenu en considérant différentes occurrences d'un même état comme des états différents). Puis on vérifie que ces deux arbres se bisimulent. La propriété suivante précise les relations entre $\sim_n$-équivalence et bisimulation.

Propriété 32 *$\sim_n$-équivalences et bisimulation*

1. *Pour tout STE,* $\mathcal{STE} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$*, la propriété suivante est vérifiée :* $\forall n \in \mathbb{N}, \sim \subset \sim_{n+1} \subset \sim_n$
2. *Si de plus,* $\mathcal{STE} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *est finitaire alors :*

$$\sim = \bigcap_{n\geq 0} \sim_n$$

3. *Si de plus,* $\mathcal{STE} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *est fini alors :* $\sim = \sim_{n_s}$ *où* $n_s = |S|$

1. Par récurrence sur n
2. Appelons $R = \cap_{n\geq 0} \sim_n$. D'après le premier point de la proposition, on a $\sim \subset R$. Pour démontrer que $R \subset \sim$, sachant que $\sim$ est l'union des bisimulations, il suffit de prouver que R est une bisimulation. Soit $s \ R \ s'$ et $s' \xrightarrow{a} t'$. Alors par définition de $R, \forall n, \exists t_n$ tel que $s \xrightarrow{a} t_n$ et $t' \sim_n t_n$ Puisque le système de transitions est finitaire, $\exists t$ tel que $t = t_n$ pour une infinité de n. Ce qui signifie que $\forall n, \exists n' > n$ tel que $t' \sim_{n'} t$. Ce qui implique d'après le premier point que $t' \sim_n t$ donc $t' \ R \ t$. La deuxième partie de la preuve est analogue à la première.
3. D'après le premier point de la proposition, $\sim_{n+1} \subset \sim_n$. De plus si pour un système de transitions (fini ou infini) $\sim_{n+1} = \sim_n$, alors $\sim_{n+1} = \sim$ puisqu'il vient en remplaçant dans la définition $\sim_n$ par $\sim_{n+1}$, que $\sim_{n+1}$ est une relation de bisimulation. Enfin supposons que pour S, toutes les relations $\sim_i$ pour $0 \leq i \leq n_s$ soient différentes, alors le nombre de classes d'équivalence croit strictement en fonction de i, ce qui est absurde puisque ce nombre doit être inférieur ou égal à n_s. Il existe donc $n \leq n_s$ tel que $\sim_n = \sim$ et par conséquent $\sim_{n_s} = \sim$.

La dernière caractérisation (cf prop 32.3) est particulièrement importante puisqu'elle nous fournit un algorithme de calcul de la bisimulation dans le cas de STE finis.

Propriété 33 *Application, Equivalence et Ensemble Quotient*

Soit f une application de $A \mapsto B$:

1. *Soit $\equiv_f$, la relation binaire $\subset A \times A$, définie par $a_1 \equiv_f a_2$ ssi $f(a) = f(b)$.*
   $\equiv_f$ est une relation d'équivalence.
2. *Soit $\pi^f(A) =_{Def} \bigcup_{b \in f(A)} f^{-1}(b)$.*
   $\pi^f(A)$ est une partition de A.
3. *$\pi^f(A) = A/\equiv_f$*

1) est immédiat car "=" est elle-même une relation d'équivalence. 2) Π_A est un recouvrement de A dont les "blocs" sont disjoints car f^{-1} est injective. 3) est assuré car $\forall \pi \in \pi^f(A)$: a_1 et $a_2 \in \pi \Rightarrow a_1 \equiv_f a_2$

Définition 34 *Output d'un sommet*

Pour un STE, $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$, on considère l'application $Output_{\mathcal{STE}}$: $S \mapsto \mathcal{P}(\Sigma)$ qui associe à chaque sommet $q \in S$, le sous-ensemble de Σ défini de la façon suivante : $Output_S(s) =_{Def} \{a \in \Sigma$ tel que $s \stackrel{a}{\rightarrow}\}$

Lorsqu'il n'y aura pas d'ambiguïté sur le STE de référence, on notera simplement $Output(s)$ à la place de $Output_{\mathcal{STE}}(s)$

Propriété 35 *Caractérisation équivalente de la $\sim_1$-équivalence*

En reprenant les notations introduites dans la propriété 33, on considère aussi la relation d'équivalence $\equiv_{Output}$.

Pour tout STE, $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$, on a :$\sim_1 = \equiv_{Output}$

Deux sommets sont équivalents à l'ordre 1 ssi ils permettent d'effectuer les mêmes actions. Il suffit de remarquer que $\sim_1$ est définie à partir de l'équivalence d'ordre $\sim_0$, équivalence pour qui deux sommets quelconques sont toujours équivalents (i.e $\sim_0 = S \times S$)

La propriété 35 permet donc de calculer directement l'ensemble des classes d'équivalence de S pour $\sim_1$ (en d'autres termes, le quotient de S par $\sim_1$), en utilisant la partition de S définie par l'application $Output^{-1}$. Une seconde propriété évidente qui peut être mise à profit pour limiter les calculs consiste à remarquer que la relation de bisimulation (et plus généralement toute équivalence de comportement) ne peut distinguer deux états de blocage entre-eux.

Propriété 36 *Bisimulation et Blocage*

Pour tout STE, $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle$, et tout couple de sommets $p, q \in S$

$[\ Output(p) = \ Output(q) = \emptyset] \Rightarrow p \sim_n q, \forall n \in \mathbb{N}$

Exemple 2 Application aux machines à café $\langle D, 0\rangle$ et $\langle D', 0'\rangle$ représentées Fig 7 :

Nous voulons savoir si $\langle D, 0\rangle$ et $\langle D', 0'\rangle$ sont bisimilaires. Comme les ensembles de sommets respectifs de ces machines (S et S') sont disjoints, nous posons $\mathcal{S} = S \cup S'$ et nous allons chercher la plus grande bisimulation (i.e $\sim$) contenue dans $\mathcal{S}$. $\langle D, 0\rangle$ et $\langle D', 0'\rangle$ seront bisimilaires ssi $0 \sim 0'$. Nous calculons $\sim$ en considérant la suite des $\sim_k$-équivalences (cf def 31 et prop 32).

Calcul de $\sim_1$

La table ci-dessous représente le graphe de l'application $Output^{-1}$ (def 34), en utilisant la propriété 35, on obtient
$\mathcal{S}/\sim_1 = \{\{0, 0'\}, \{\{1'\}, \{3'\}, \{2, 2', 3, 4'\}\}$

$\mathcal{P}(\Sigma)$	$\mathcal{P}(\Sigma) \mapsto S$	$\mathcal{P}(\Sigma) \mapsto S'$	$\mathcal{P}(\Sigma) \mapsto S \cup S'$
Σ	$\emptyset$	$\emptyset$	$\emptyset$
{Thé, Café}	$\{1\}$	$\emptyset$	$\{1\}$
{Thé, Pièce}	$\emptyset$	$\emptyset$	$\emptyset$
{Café, Pièce}	$\emptyset$	$\emptyset$	$\emptyset$
{Pièce}	$\{0\}$	$\{0'\}$	$\{0, 0'\}$
{Café}	$\emptyset$	$\{1'\}$	$\{1'\}$
{Thé}	$\emptyset$	$\{3'\}$	$\{3'\}$
$\emptyset$	$\{2, 3\}$	$\{2', 4'\}$	$\{2, 2', 3, 4'\}$

Calcul de $\sim$

$0 \not\sim_2 0'$: En effet on a $0 \stackrel{\text{Pièce}}{\rightarrow} 1$ et aucun des successeurs de 0' par Pièce (i.e 1'et 3') n'est équivalent à l'ordre 1 à 1 (i.e $1' \not\sim_1 1$ et $3' \not\sim_1 1$). On peut donc directement déduire que $\langle D, 0\rangle$ et $\langle D', 0'\rangle$ ne sont pas bisimilaires.

La propriété 36, assure que $\{2, 2', 3, 4'\} \in \mathcal{S}/\sim$ Mis à part cette classe $\{2, 2', 3, 4'\}$, les autres classes sont réduites à un singleton et donc minimales. On peut donc en déduire que $\mathcal{S}/\sim = \mathcal{S}/\sim_2 = \{\{0\}, \{0'\}, \{\{1'\}, \{3'\}, \{2, 2', 3, 4'\}\}$

Caractérisation Opérationnelle

Le procédé décrit dans la définition 4.1.1 permet d'obtenir la plus grande bisimulation incluse dans une relation binaire donnée R comme la limite de la suite décroissante de relations $<\sim_n>_{n\geq 0}$. Chacun des termes de la suite peut être décrit de façon équivalente sous la forme d'une partition de l'ensemble des états. Le calcul des termes de cette suite décroissante revient au problème de raffinement d'une partition (Multi Relational Coarset Partition Problem) [PT 87] utilisé initialement dans le cadre de la minimalisation des automates [AHO 74]. On obtient ainsi les algorithmes les plus performants en $O(\Delta.log(S))$ où Δ dénote le nombre de transitions et S le nombre de sommets du graphe [FER 89].

Π-Bisimulation

Nous avons vu dans cette section, la relation de bisimulation standard telle que celle présentée par [PARK 81, MIL 89]. Celle-ci ne prend en compte que les labels d'événements et ne permet pas de prendre en compte les états du système. La notion de Π-Bisimulation [CLE 89] généralise la notion de bisimulation en imposant que la relation de bisimulation soit contenue dans une relation d'équivalence donnée a priori. On peut ainsi, par le biais de cette relation, prendre en compte certaines caractéristiques des états dans le calcul de la bisimulation. Nous utiliserons cette notion de bisimulation en section 4.3.1 pour pouvoir considérer une logique de Hennessy-Milner prenant en compte des propositions d'état et aussi en section 4.2.5 rendre l'équivalence observationnelle sensible à la divergence.

Définition 37 Π-*Bisimulation*

Soit un STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *et* Π *une relation d'équivalence sur* S *(i.e* $\Pi \subset S\times S$*), une relation binaire* B *sur* S *est une relation de* Π*-bisimulation ssi elle vérifie :* $B \subset \Pi$ *et*

$\forall (p,p') \in B$ *et* $\forall t \in \Sigma$:

$$[\forall q \in S \ Si \quad p \stackrel{t}{\rightarrow} q \ \ alors \ \exists q' \in S \ : \quad p' \stackrel{t}{\rightarrow} q' \ \ et \ (q,q') \in B]$$
$$et \quad [\forall q' \in S \ Si \quad p' \stackrel{t}{\rightarrow} q' \ \ alors \ \exists q \in S \ : \quad p \stackrel{t}{\rightarrow} q \quad et \ (q',q) \in B]$$

Notons que dans le cas où $\Pi = S \times S$, on retrouve la notion standard de bisimulation. La procédure de décision générale donnée en section 4.1.1 s'adapte aux Π-bisimulations. Il suffit d'initialiser la suite de relations d'équivalences en prenant $\sim_0 = \Pi$.

4.1.2. *Simulation et co-simulation*

Autant la notion de bisimulation induit une relation d'équivalence sur les STE, autant la notion de simulation permet de définir un pré-ordre (une relation binaire réflexive et transitive) sur les STE. La notion de simulation est définie en rompant la symétrie de la définition de la bisimulation

Définition 38 *Simulation*

Soit deux STE, $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *et* $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$*, une relation binaire* R*,* $(R \subset S \times S')$ *est une simulation entre* $\mathcal{STE}$ *et* $\mathcal{STE}'$ *ssi elle vérifie :* $\forall (p,p') \in R$ *et* $\forall t \in \Sigma$:

$$\forall q \in S \ Si \quad p \stackrel{t}{\rightarrow} q \ \ alors \ \exists q' \in S' \ : \quad p' \stackrel{t}{\rightarrow} q' \ \ et \ (q,q') \in R$$

Extension aux STEI : *Comme pour la relation de bisimulation, la relation de simulation peut être étendue aux systèmes de transitions initialisés en convenant que* $\langle \mathcal{STE}, s_0 \rangle$ *simule* $\langle \mathcal{STE}', s'_0 \rangle$ *ssi il existe une relation de simulation reliant leurs états respectifs initiaux : i.e., il existe une relation de simulation* R *contenant* (s_0, s'_0)

La **co-simulation** permet d'obtenir une relation d'équivalence à partir du pré-ordre de simulation.

Définition 39 *Co-simulation*

La co-simulation se définit en considérant la relation d'équivalence associée au pré-ordre de simulation.

$\mathcal{STE}$ *co-simule* $\mathcal{STE}' \Leftrightarrow_{Def}$ $\mathcal{STE}$ *simule* $\mathcal{STE}'$ *et* $\mathcal{STE}'$ *simule* $\mathcal{STE}$.

Exemple 3

Figure 9: Deux STE co-similaires et non bisimilaires

Considérons $R = \{(0,0'), (1,1'), (2,2'), (3,1')\}$ et vérifions qu'il s'agit bien d'une simulation entre $\langle S, 0 \rangle$ et $\langle S', 0' \rangle$.

On a $0 \xrightarrow{A} \{1,3\}$ et $0' \xrightarrow{A} 1'$ avec $(1,1') \in R_1$ et $(3,1') \in R_1$

On a $1 \xrightarrow{B} 2$ et $1' \xrightarrow{B} 2'$ avec $(2,2') \in R_1$

Comme 2 et 3 sont des états de blocage, il n'y a rien de plus à vérifier.
R_1 est donc une simulation contenant $(0,0')$, donc $\langle S, 0 \rangle$ simule $\langle S', 0' \rangle$.

Dans l'autre sens, on montre de même que la relation$R_2 = \{(0',0), (1',1), (2',2)\}$ est une relation de simulation entre $\langle S', 0' \rangle$ et $\langle S, 0 \rangle$. En effet, on a : $0' \xrightarrow{A} 1'$ et $0 \xrightarrow{A} 1$ avec $(1',1) \in R_2$ & $1' \xrightarrow{B} 2'$ et $1 \xrightarrow{B} 2$ avec $(2',2) \in R_2$

On a donc $\langle S, 0 \rangle$ simule $\langle S', 0' \rangle$ et $\langle S', 0' \rangle$ simule $\langle S, 0 \rangle$. $\langle S, 0 \rangle$ et $\langle S', 0' \rangle$ sont donc Co-similaires.

Par contre, $\langle S, 0 \rangle$ et $\langle S', 0' \rangle$ ne sont pas bisimilaires.

Il suffit de raisonner par l'absurde et de considérer B une relation de bisimulation. Celle-ci a minima contiendrait $(0',0)$. Par suite, comme $0 \xrightarrow{A} \{1,3\}$

et $0' \stackrel{A}{\rightarrow} 1'$, B devrait aussi contenir aussi les couples $(1',1),(3',1)$. Comme $1' \stackrel{B}{\rightarrow} 2$ et $3 \stackrel{B}{\not\rightarrow}$ on ne peut avoir $(1',3) \in B$ d'où la contradiction.
nb : La relation $B' = \{(3,2'),(2,2')\}$ est la plus grande bisimulation entre S et S'.

Remarque L'exemple précédent montre que la co-simulation est plus faible que la bisimulation. La bisimulation peut pourtant se définir en termes de simulation de la façon suivante : une relation de simulation R dont la relation symétrique R^{-1} est elle même une relation de simulation est une bisimulation. La section 4.3.4 présentant la caractérisation modale des équivalences de comportement nous permettra de préciser les relations entre les notions de simulation, co-simulation et bisimulation.

4.1.3. *Procédure de décision pour la simulation*

La construction présentée est très proche de celle présentée en 4.1.1 pour décider de la bisimulation. Au lieu d'utiliser une suite de relations (les $\sim_k$-équivalences de la définition 31), on introduit une fonction E qui par itérations successives va permettre d'obtenir son plus petit point fixe qui correspond, en l'occurrence, à la relation de simulation cherchée. Le lecteur intéressé trouvera un exposé complet de cette construction et des preuves associées dans [ARN 92].

Nous présentons ici la forme plus générale permettant de calculer une Π-Simulation analogue à la notion de Π-bisimulation introduite définition 37). Ce calcul construit à partir d'une relation binaire quelconque R, la plus grande simulation (si elle existe) contenue dans R.

Définition 40 *Simulation engendrée par une relation*

Soient deux STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *et* $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$

On considère l'application $E : \mathcal{P}(S\times S') \mapsto \mathcal{P}(S\times S')$ *qui associe à toute relation binaire* $R \subset S\times S'$, *la relation* $E(R)$ *sur* $S\times S'$ *définie par :*
$(s,s') \in E(R) \Leftrightarrow_{Def} (1)\wedge(2)$ *où*
(1) $(s,s') \in R$
(2) $\forall t\in\Sigma, \forall q\in S \;:\; s \stackrel{t}{\rightarrow} q \Rightarrow \exists q'\in S' \;:\; s' \stackrel{t}{\rightarrow} q'$ *tel que* $(q,q') \in R$

Propriété 41 *Caractérisation d'une simulation*

Soient deux STE, $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *et* $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$

1. *La propriété suivante est vérifiée :*
   $\forall n \in \mathbb{N}, E^{n+1}(R) \subset E^n(R) \subset R$

2. *Si de plus, S et S' sont finitaires alors :*
   La suite de relations $< E^n(R) >_{n \geq 0}$ *admet comme limite* R^ω *avec* $R^\omega = \bigcap_{n \geq 0} E^n(R)$.
   R^ω *est le plus grand point fixe de la fonction* E
   i.e $E(R^\omega) = R^\omega$ *et* $E(A) = A \Rightarrow A \subseteq R^\omega$
3. *Si de plus, S et S' sont finis alors :*
   $R^\omega = E^k(R)$ *où* $k = max(|S|, |S'|)$

La propriété précédente est analogue à la caractérisation des bisimulations (cf prop 32). Le point 2) montre que R^ω est la plus grande simulation entre Σ et Σ' incluse dans R. Comme E est une application décroissante d'un ensemble de parties dans lui-même, la convergence de la suite est assurée [ARN 92]. Par construction, R^ω est incluse dans R et constitue la plus grande solution de l'équation $E(R) = E$, R^ω est donc la simulation la plus large incluse dans R. Le point 3) de la propriété précédente fournit un moyen de décider de la simulation entre deux STE finis.

4.2. *Equivalences Faibles*

Les relations d'équivalence que nous avons considérées jusqu'à présent supposaient que les systèmes de transitions que nous comparions admettaient les mêmes ensembles de labels de transitions. En l'état, cette contrainte d'identité des alphabets d'action limite fortement les possibilités de mise en oeuvre des équivalences ou des pré-ordres de comportements : il n'est pas possible, par exemple de comparer des systèmes décrits à différents niveaux d'abstraction. Ainsi les différents distributeurs de boisson que nous avons représentés dans cette section n'ont de sens que dans la mesure où ils représentent de façon abstraite le "service" rendu par un distributeur de boisson ; de toute évidence un véritable distributeur serait autrement plus complexe. Dans la pratique, l'approche comportementale nécessite donc de pouvoir comparer des systèmes décrits à différents niveaux d'abstraction. Concrètement, on veut comparer des systèmes en faisant "abstraction" de certains événements (actions) que l'on ne désire pas observer où que l'on ne suppose pas pertinents vis à vis de l'analyse que l'on veut conduire. La première étape revient donc à définir le critère d'observation du système : les événements que l'on veut observer et ceux dont on fait abstraction.

Une solution simple consiste à considérer un sous-ensemble $\mathcal{O}$, d'actions observables, de l'ensemble des labels Σ. Une fois ce critère défini, il faut expliciter ce que l'on entend par faire abstraction d'un événement inobservable. Une première alternative nous est fournie en réutilisant la notion de projection déjà définie dans les langages.

Définition 42 *Projection d'un langage*

Soit $\mathcal{O}$ un sous-ensemble de Σ, σ un mot de Σ^*, la projection de σ sur $\mathcal{O}$, notée $\sigma_{\lfloor\mathcal{O}}$ est définie récursivement par :

$$\lambda_{\lfloor\mathcal{O}} =_{Def} \lambda \text{ et } (\sigma.a)_{\lfloor\mathcal{O}} =_{Def} \begin{cases} \sigma_{\lfloor\mathcal{O}}.a & si\ a \in \mathcal{O} \\ \sigma_{\lfloor\mathcal{O}} & sinon \end{cases}$$

La projection opère comme une "gomme" qui efface toutes les lettres du mot n'appartenant pas à $\mathcal{O}$. Cet opérateur de projection s'étend de façon canonique à un ensemble de mots en posant pour $L \subset \Sigma^*$: $L_{\lfloor\mathcal{O}} =_{Def} \{\sigma_{\lfloor\mathcal{O}} : \sigma \in L\}$

Définition 43 *Equivalence langage faible*

La comparaison entre les systèmes est définie vis à vis d'un critère d'observation commun : ces systèmes seront équivalents si les projections de leurs langages par rapport à ce critère d'observation sont égales.

Soient $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ et $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ deux systèmes de transitions, s_0 et s'_0 leurs états initiaux respectifs et $\mathcal{O}$ un critère d'observation commun i.e., $\mathcal{O} \subset \Sigma \cap \Sigma'$

$\langle \mathcal{STE}, s_0\rangle \equiv_{\mathcal{O}} \langle \mathcal{STE}', s'_0\rangle$ *ssi* $L(\langle \mathcal{STE}, s_0\rangle)_{\lfloor\mathcal{O}} = L(\langle \mathcal{STE}', s'_0\rangle)_{\lfloor\mathcal{O}}$

nb : Cette équivalence généralise l'équivalence langage présentée définition 18. En effet, en prenant $\mathcal{O} = \Sigma \cup \Sigma'$, on a $\langle \mathcal{STE}, s_0\rangle \equiv_{\mathcal{O}} \langle \mathcal{STE}', s'_0\rangle$ ssi $\langle \mathcal{STE}, s_0\rangle \equiv \langle \mathcal{STE}', s'_0\rangle$

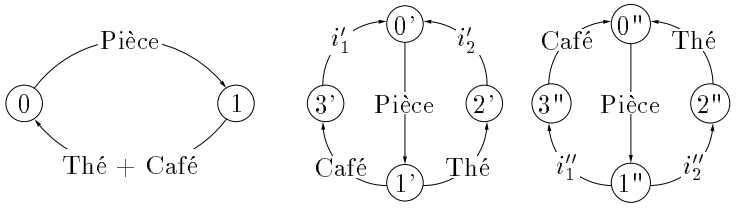


Figure 10: Trois autres machines à Café

Exemple 4 Application de l'équivalence langage faible

Considérons les trois machines à café $\langle M, 0\rangle, \langle M', 0'\rangle$ et $\langle M'', 0''\rangle$ représentées Figure [2] 10 et comparons les en observant uniquement l'alphabet $\mathcal{O}$ constitué des actions Pièce, Thé et Café. Ces trois systèmes admettent après projection le même langage décrit par l'expression rationnelle suivante : (Pièce.(Thé+ Café))*, ils sont donc équivalent langage pour $\mathcal{O}$: $\langle M, 0\rangle \equiv_{\mathcal{O}} \langle M', 0'\rangle \equiv_{\mathcal{O}} \langle M'', 0''\rangle$.

[2] l'étiquette Thé + Café reliant les états 1 à 0 signifie simplement que l'on peut aller de l'état 0 a'l'état 1 soit par l'action Thé soit par l'action Café

Pour autant du point de vue d'un client, $\langle M'', 0'' \rangle$ se distingue des deux autres puisque cette machine "choisit de facon autonome" la boisson délivrée au client. Dans l'état 1", M" est apte à offrir du Thé ou de Café mais les actions i''_1 et i''_2 dont on avait a priori voulu faire abstraction ont une influence sur le service offert par ces machines puisque suivant leurs occurrences, celle ci delivrera du Thé ou du Café.

Les actions dont on avait décidé a priori de faire abstraction ($\{i'_1, i'_2, i''_1, i''_2\}$) disparaissent sous l'effet de la "gomme" qui a opéré lors de la projection. Pour certaines d'entre-elles, i'_1 et i'_2, l'abstraction réalisée est légitime au sens où celles-ci ne modifient pas le comportement "observable" du système, pour d'autres i''_1 et i''_2, l'abstraction réalisée n'est pas fondée puisque ces actions ont une influence observable sur le comportement du système.

Toute la difficulté est de savoir a priori s'il est "raisonnable" ou non de ne pas observer (faire abstraction d') une action. L'équivalence observationnelle, introduite par R. Milner dans son calcul pour les systèmes communicants CCS [MIL 89], introduit une notion d' "expérimentation abstraite" qui permet de pallier ce problème.

4.2.1. *Expérimentation, Saturation*

La notion d'expérimentation abstraite permet une abstraction moins radicale que la notion de projection qui efface purement et simplement toute action inobservable. Ici, les actions inobservables sont dans un premier temps banalisées et renommées par un symbole commun τ. De plus, une nouvelle relation de transition, dite d'"expérimentation abstraite" et notée $\Rightarrow$, est définie en prenant en compte la relation de transition originelle ($\rightarrow$) et les séquences d'actions inobservables. Nous en donnons maintenant la définition et donnerons l'intuition de celle-ci en s'appuyant sur l'exemple de saturation présenté Figure 5.

Définition 44 *Expérimentation abstraite :* $\Rightarrow$

Soit un STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a \in \Sigma} \rangle, \mathcal{O}$ *un sous-ensemble de* Σ *et* τ *un symbole n'appartenant pas à* Σ

$\Rightarrow \subset S \times \Sigma \cup \{\tau\} \times S$ *peut se définir ainsi :* $\Rightarrow =_{Def} \stackrel{\tau}{\Rightarrow} \bigcup \cup_{o \in \mathcal{O}} [\stackrel{o}{\Rightarrow}]$ *où*

- $\stackrel{\tau}{\Rightarrow}$ *la relation de transition relative aux expérimentations inobservables, est obtenue en prenant la fermeture réflexive et transitive de la réunion des relations de transition inobservable :*

$\stackrel{\tau}{\Rightarrow} =_{Def} [\bigcup_{i \in \Sigma \setminus \mathcal{O}} \stackrel{i}{\rightarrow}]^*$

nb : $\stackrel{\tau}{\Rightarrow}$ permet d'une part de banaliser toutes les actions inobservables : tous les labels d'actions inobservables sont renommés en un label commun : τ. La transitivité de $\stackrel{\tau}{\Rightarrow}$ permet de considérer une séquence d'actions inobservables comme une action "atomique" inobservable. Enfin la réflexivité de $\stackrel{\tau}{\Rightarrow}$ assure que de tout état du STE il est possible de faire une action inobservable : il s'agit ici d'ajouter une transition inobservable "neutre" bouclant sur tout sommet du STE.

- *$\stackrel{a}{\Rightarrow}$, la relation relative aux expérimentations observables se définit comme la double composition (à droite et à gauche) de la relation d'expérimentation inobservable $\stackrel{\tau}{\Rightarrow}$ avec la relation de transition observable originelle $\stackrel{a}{\rightarrow}$.*

$$\stackrel{a}{\Rightarrow} =_{Def} \stackrel{\tau}{\Rightarrow} o \stackrel{a}{\rightarrow} o \stackrel{\tau}{\Rightarrow} \text{ pour } a \in \mathcal{O}$$

nb : $\stackrel{a}{\Rightarrow}$ permet d'étendre la notion de transition observable en intégrant les séquences de transitions inobservables. Comme $\stackrel{\tau}{\Rightarrow}$ est réflexive, $\stackrel{a}{\Rightarrow}$ contient la relation de transition observable originelle : $\stackrel{a}{\rightarrow}$. La double composition à droite et à gauche permet de considérer comme une action observable "atomique" toute action observable précédée et suivie par une séquence d'actions inobservables.

Définition 45 *Système de transition étiqueté saturé*

Le STE obtenu en substituant à la relation de transitions originelle ($\rightarrow$) la relation d'expérimentation ($\Rightarrow$) est appelé STE saturé.

Soit un STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ et $\mathcal{O}$ un sous-ensemble de Σ et τ un symbole n'appartenant pas à Σ

$$Sat_{\lfloor\mathcal{O}}(\mathcal{STE}) =_{Def} \langle \Sigma \cup \{\tau\}, S, \{\stackrel{a}{\Rightarrow}\}_{a\in\Sigma\cup\{\tau\}}\rangle$$

nb : Dans le cas où $\Sigma = \mathcal{O}$, la saturation consiste simplement à ajouter au système initial une "boucle" étiquetée par τ sur chacun des sommets.

$$Sat_{\lfloor\Sigma}(\mathcal{STE}) = \langle \Sigma \cup \{\tau\}, S, [\{\stackrel{a}{\rightarrow}\}_{a\in\Sigma} \cup \{p \stackrel{\tau}{\rightarrow} p \ : p \in S\}\rangle$$

Exemple 5 Exemple de Saturation

La figure 11 présente les saturés des systèmes M' et M" représentés Figure 10 lorsque $\mathcal{O} = \{\text{Pièce}, \text{Thé}, \text{Café}\}$.

Pour ne pas surcharger la figure, les boucles de τ résultant de la fermeture réflexive de la relation $\stackrel{\tau}{\Rightarrow}$, normalement associées à chacun des états, ne sont pas représentées.

Les labels d'actions inobservables $(i'_1, i'_2, i''_1, i''_2)$ ont été banalisés dans les systèmes saturés. Cette banalisation jointe à la double composition qui permet de prendre en compte les séquences d'actions inobservables, conduit à une relation d'expérimentation observable $\stackrel{o}{\Rightarrow}$ non déterministe.

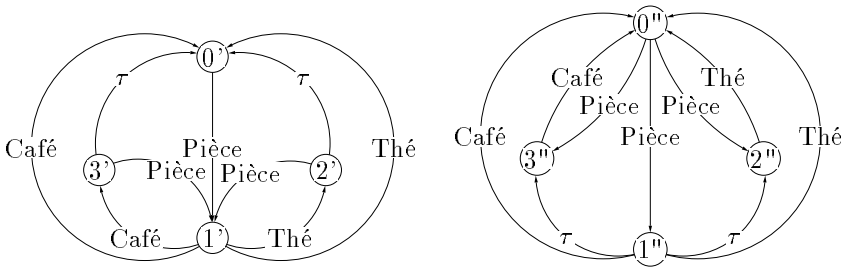


Figure 11: $Sat_{\lfloor\mathcal{O}}(M')$ et $Sat_{\lfloor\mathcal{O}}(M'')$

Partant de STE déterministes, on peut aboutir à des systèmes saturés non-déterministes. Ainsi partant de 1' dans $Sat_{\lfloor\mathcal{O}}(M')$, l'expérimentation Café conduira en 3' ou en 1' suivant que seule l'action Café s'est produite ou que celle-ci a été suivie par l'action inobservable i'_1. De même partant de 0" dans $Sat_{\lfloor\mathcal{O}}(M'')$, l'expérimentation Pièce peut conduire dans l'état 1" où les actions Thé et Café sont possibles, dans l'état 2" où seule l'action Thé est possible soit dans l'état 3" où seule l'action Café est possible.

L'abstraction réalisée en choisissant de ne pas observer les actions i''_1, i''_2 n'occulte pas le fait que la machine M'' peut choisir la boisson à la place du client.

4.2.2. *Bisimulation Faible, Equivalence Observationnelle*

L'équivalence observationnelle de deux systèmes [MIL 89] peut se définir directement à partir de la bisimulation (def 25) en considérant la relation de bisimulation entre les systèmes saturés.

Définition 46 *Bisimulation Faible*

Soit deux STE $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ *et* $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ *et* $\mathcal{O} \subset \Sigma \cap \Sigma'$*, un ensemble de labels d'actions communs.*

Une relation binaire $B \subset S \times S'$ *est une* $\lfloor_{\mathcal{O}}$*Bisimulation entre* $\mathcal{STE}$ *et* $\mathcal{STE}'$ *ssi* B *est une bisimulation entre* $Sat_{\lfloor\mathcal{O}}(\mathcal{STE})$ *et* $Sat_{\lfloor\mathcal{O}}(\mathcal{STE}')$

Cette bisimulation paramétrée par un ensemble d'actions observables est souvent qualifiée de bisimulation "faible" par opposition à la bisimulation standard, dite "forte", qui prend en compte tous les labels d'actions. L'équivalence observationnelle introduite dans CCS [MIL 89] est la plus large bisimulation faible.

Les notions de bisimulation forte et faible coïncident lorsque tous les labels d'actions sont observés (i.e $\mathcal{O} = \Sigma \cup \Sigma'$) : B est une bisimulation (forte) entre $\mathcal{STE}$ et $\mathcal{STE}'$ ssi B est une $_{\lfloor(\Sigma\cup\Sigma')}$B*isimulation* entre $Sat_{\lfloor\Sigma}(\mathcal{STE})$ et $Sat_{\lfloor\Sigma'}(\mathcal{STE}')$

4.2.3. *Décision de la bisimulation faible*

Comme la bisimulation faible est finalement définie comme une bisimulation forte sur des systèmes saturés, la procédure de décision que nous avons donnée pour la bisimulation forte s'applique pour décider de la bisimulation faible.

Lorsque l'on considère le système saturé, la propriété 35 reste valable (elle peut être simplifiée en remarquant que tout sommet du système saturé possède τ dans son $Output$). Il en est de même pour la propriété 36 mais celle-ci devient inopérante car dans le système saturé, tout sommet possède au moins un τ-successeur (lui-même). Dans le cas de la bisimulation faible, la propriété 36 se reformule en termes de blocage faible (def 47).

Propriété 47 *Bisimulation faible et blocage faible*

Pour tout STE, $\mathcal{STE} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$*, tout sous-ensemble* $\mathcal{O}$ *de* Σ *et tout couple de sommets* $p, q \in S$

$$[\ Output_{Sat_{\lfloor\mathcal{O}}(\mathcal{STE})}(p) = \ Output_{Sat_{\lfloor\mathcal{O}}(\mathcal{STE})}(q) = \{\tau\}] \Rightarrow p \sim_{\mathcal{O}} q$$

Exemple 6 Suite de l'exemple 5

On considère de nouveau les STEI $\langle M', 0'\rangle$ et $\langle M'', 0''\rangle$ de l'exemple 5. On cherche une bisimulation entre $\langle M', 0'\rangle$ et $\langle M'', 0''\rangle$. On pose $\mathcal{S} = S \cup S'$

La propriété 36 nous permet d'obtenir l'équivalence à l'ordre 1 :

$$\mathcal{S}/\sim_1 = \{\{0', 2', 3', 0''\}, \{1', 1''\}, \{2''\}, \{3''\}\}$$

.

A l'ordre 2, il est facile de voir que $1' \not\sim_2 1''$. En effet, on a $1'' \stackrel{\tau}{\Rightarrow} 2''$ or 1' admet un seul τ- successeur, lui même, qui n'est pas équivalent à l'ordre 1 à 2". En poursuivant ainsi, on montrerait, à l'ordre 3, que 1'et 1" ne sont pas équivalents. Ainsi $\langle M', 0'\rangle$ et $\langle M'', 0''\rangle$ ne sont pas équivalent observationnellement.

4.2.4. *Modèles Quotients/Abstraits*

Les équivalences ou les pré-ordres que nous avons vus jusqu'ici permettent de comparer un système et sa spécification, tous deux exprimés sous forme de

graphe. La vérification comportementale, en procédant ainsi par comparaison, permet de s'assurer que le système et sa spécification ont les mêmes propriétés (le même comportement) modulo le critère d'abstraction retenu pour exécuter la comparaison.

Pour les relations d'équivalences, on peut aussi procéder par projection ou "introspection". Au lieu de comparer deux STE, on peut construire pour un STE donné, le plus petit [3] STE qui lui est équivalent. On parle ainsi de projection ou de modèle abstrait.

Définition 48 *Projection observationnelle*

Pour un STE, $\mathcal{STE} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$*, un sous-ensemble* $\mathcal{O} \subset \Sigma$ *de labels observables et* $\sim_{\mathcal{O}}$*, la relation d'équivalence observationnelle associée, on note* $\mathcal{STE}/\sim_{\mathcal{O}}$ *le STE quotient de* $\mathcal{STE}$ *par* $\sim_{\mathcal{O}}$

$\mathcal{STE}/\sim_{\mathcal{O}} =_{Def} \langle S/\sim_{\mathcal{O}}, \mathcal{O}\cup\{\tau\}, \{\overset{a}{\sim\!\!\gg}\}_{a\in\mathcal{O}\cup\{\tau\}}\rangle$ *où :*

1. $S/\sim_{\mathcal{O}}$ *est quotient de* S *par* $\sim_{\mathcal{O}}$
2. $\overset{a}{\sim\!\!\gg}$ *la plus petite relation vérifiant :*

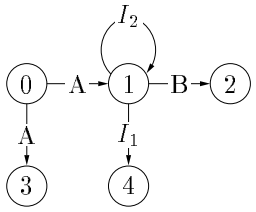
   (a) $(a \in \mathcal{O}$ *et* $q \overset{o}{\Rightarrow} q') \Rightarrow q/\sim_{\mathcal{O}} \overset{o}{\sim\!\!\gg} q'/\sim_{\mathcal{O}}$

   (b) $(a \notin \mathcal{O}$ *et* $q \overset{o}{\Rightarrow} q'$ *et* $q \not\sim_{\mathcal{O}} q') \Rightarrow q/\sim_{\mathcal{O}} \overset{\tau}{\sim\!\!\gg} q'/\sim_{\mathcal{O}}$

(2.a) signifie que toute transition observable subsiste dans la projection. (2.b) signifie que seules les transitions inobservables reliant deux états non équivalents sont conservées dans la projection.

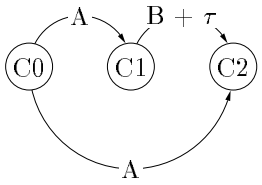
Exemple 7 Exemple de projection

On considère $\langle X, 0\rangle$ le STEI représenté ci-contre. On choisit d'observer $\mathcal{O} = \{A, B\}$. Dans ce cas, on a $X/\sim_{\mathcal{O}} = \{\{0\}, \{1\}, \{2,3,4\}\}$

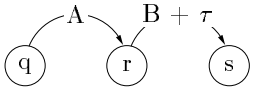


[3] vis à vis du nombre d'états

Le STEI $\langle X/\sim_{\mathcal{O}}, C0\rangle$ obtenu par projection est représenté ci-contre. Il comporte 3 sommets :
$C0 = \{0\}, C1 = \{1\}, C2 = \{2,3,4\}$ On peut noter que certaines transitions inobservables subsistent ($1 \stackrel{I_1}{\rightarrow} 4$ qui matérialisait la possibilité de blocage après A apparaît sous la forme $C1 \stackrel{\tau}{\rightsquigarrow} C2$) tandis que d'autres disparaissent. La transition $1 \stackrel{I_3}{\rightarrow} 1$ montre que le système peut avoir une exécution infinie inobservable, on parlera par la suite (cf sections 4.2.5) de divergence. Cette possibilité de divergence du système est absorbée dans la classe $C1$).



Par construction, le STEI obtenu par projection ($\langle X/\sim_{\mathcal{O}}, C0\rangle$) est équivalent au STEI projeté $\langle X, 0\rangle$. La projection est minimale vis à vis du nombre d'états. Par contre elle n'est pas minimale vis à vis du nombre de transitions. Soit $\langle Y, q\rangle$, le STEI représenté ci-contre. On a de nouveau $\langle Y, q\rangle \sim_{\mathcal{O}} \langle X, 0\rangle$ mais $\langle Y, q\rangle$ comporte moins de transitions que $\langle X/\sim_{\mathcal{O}}, C0\rangle$.



Exemple 8 Suite de l'exemple 5

En reprenant les résultats obtenus dans l'exemple 5, on a :

$$\mathcal{S}/\sim = \{\{0', 2', 3'\}, \{0''\}, \{1'\}, \{2''\}, \{3''\}\}$$

On peut par simple projection[4] en déduire la relation $\sim$ sur les ensembles de sommets respectifs (S et S') des systèmes STEI $\langle M', 0'\rangle$ et $\langle M'', 0''\rangle$. Ici on obtient :

$$S/\sim = \{\{0', 2', 3'\}, \{1'\}\} \text{ et } S'/\sim = \{\{0''\}, \{1''\}, \{2''\}, \{3''\}\}$$

La figure ci-dessous reprend les différentes étapes du calcul : à gauche le système initial, au milieu le système sature et, à droite, le système projeté.

[4] projection ici au sens usuel : projection de la partition sur les ensembles de sommets associés à chacun des STE

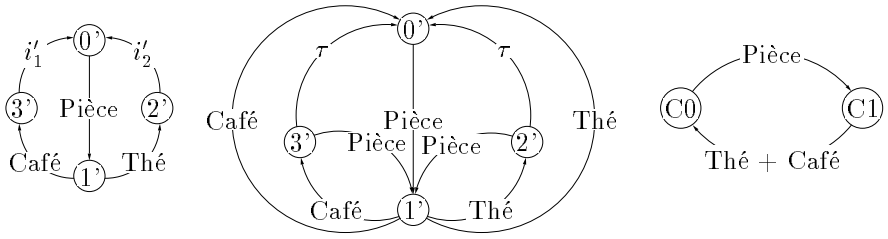


Figure 12: $\langle M', 0' \rangle, Sat_{\lfloor \mathcal{O}}(M')$ et $\langle M'/\sim_{\mathcal{O}}, 0'/\sim_{\mathcal{O}} \rangle$

Pour $\langle M'', 0'' \rangle$, les classes d'équivalence, $S''/\sim_{\mathcal{O}}$, sont réduites à des singletons, et le STE projeté est identique (isomorphe au renommage près, des actions inobservables i''_1 et i''_2 par τ) au STE initial.

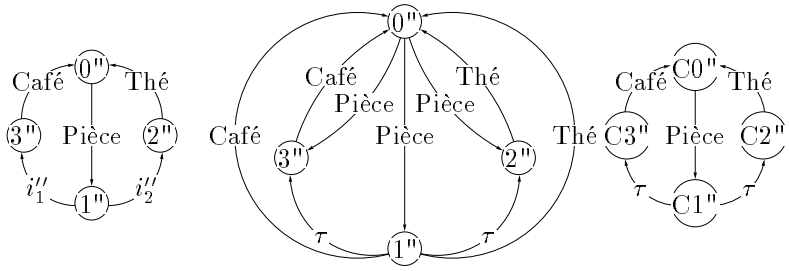


Figure 13: $\langle M'', 0'' \rangle, Sat_{\lfloor \mathcal{O}}(M'')$ et $\langle M''/\sim_{\mathcal{O}}, 0''/\sim_{\mathcal{O}} \rangle$

La projection de $\langle M'/\mathcal{O}, 0'/\mathcal{O} \rangle$ permet de retrouver (à un isomorphisme près) la machine $\langle M, 0 \rangle$ (cf fig 6), celles-ci sont en effet équivalentes observationnellement.

Pour $\langle M', 0' \rangle$, le fait d'obtenir une seule classe regroupant les états 0', 2' et 3' a permis d' "internaliser" les deux transitions (i'_1 et i'_2) dont on avait décidé de faire abstraction. Le calcul de l'équivalence observationnelle permet de légitimer a posteriori ce choix : l'occurrence de ces événements ne modifie pas le comportement "observé"du système.

Pour $\langle M'', 0'' \rangle$, ne pas observer les actions i''_1 et i''_2 n'a pas de sens puisque suivant leurs occurrences le client aura ou n'aura pas le choix de sa boisson.

4.2.5. *Equivalence observationnelle et divergence*

L'exemple précédent a permis de montrer que l'équivalence observationnelle permettait d'opérer une abstraction relativement raisonnée vis à vis des événements que l'on ne désirait pas observer. Contrairement, par exemple, à l'équivalence langage faible qui gomme a priori tous les événements non observés, l'équivalence observationnelle peut conserver "visibles" des événements que l'on ne voulait pas a priori observer, typiquement le cas des événements inobservables qui subsistent sur le STE quotient : ces événements sont précisément ceux qui relient des états non observationnellement équivalents.

Parmi les notions importantes qui sont "masquées" par l'équivalence observationnelle, nous retrouvons la notion de divergence déjà rencontrée dans notre exemple de projection 7. Dans le cas des STE finis, la divergence est directement liée à l'existence de chemins cycliques étiquetés par des labels d'actions inobservables, ce que nous appellerons des τ-cycles.

Le fait d'observer uniquement certains événements nous conduit à affiner la notion standard d'état de blocage. En effet, on peut être amené à vouloir distinguer des états à partir desquels le système (sans être bloqué) ne peut plus qu'exécuter des séquences d'actions inobservables (on parle alors de blocage faible) et des états où le système peut exécuter un nombre infini d'actions inobservables (on parle alors de divergence).

Définition 49 *Blocage faible, divergence*

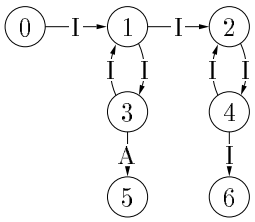
Pour un STE, $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$, $\mathcal{O}$ *un sous-ensemble de* Σ *et* s *un état de* S

1. s *est un blocage faible pour* $\mathcal{O}$ *ssi* $Output_{Sat_{\lfloor\mathcal{O}}(\mathcal{STE})}(s) = \{\tau\}$
2. s *est un état de divergence pour* $\mathcal{O}$ *ssi* $\exists\sigma \in L(\langle S, s\rangle) \cap (\Sigma \setminus \mathcal{O})^\infty$

Un blocage faible correspond à un état où le système ne peut évoluer de façon observable (les seules actions exécutées ne sont pas observables) Un observateur ne peut donc pas faire la différence entre un état de blocage faible et un état de blocage. Contrairement à un blocage faible, à partir d'un état de divergence le système peut évoluer de façon observable mais il peut, tout aussi bien évoluer indéfiniment de façon inobservable.

Exemple 9

On considère le STE $\langle L, 0\rangle$ représenté ci-contre.
Seule l'action A est observée ($\mathcal{O} = \{A\}$).
5 et 6 sont des états de blocage,
2, 4, 5, et 6 sont des états de blocage faible,
0, 1, 2, 3 et 4 sont des états de divergence.



Définition 50 *τ-cycles*

Soit $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ et $\mathcal{O} \subset \Sigma$ un sous-ensemble de labels observables.

Deux états $s_1, s_2 \in S$ sont reliés par un τ-cycle ssi
$\exists\sigma_1 \in (\Sigma\backslash\mathcal{O})^*, \exists\sigma_2 \in (\Sigma\backslash\mathcal{O})^*$ tel que $s_1 \stackrel{\sigma_1}{\rightarrow} s_2$ et $s_2 \stackrel{\sigma_2}{\rightarrow} s_1$

Propriété 51 *τ-cycle et bisimulation faible*

Deux états $s_1, s_2 \in S$ appartenant à un même τ-cycle sont faiblement bisimilaires.

Il suffit de remarquer que les états s_1 et s_2 admettent exactement les mêmes états dérivés i.e : $\forall t \in \mathcal{O}\cup\{\tau\}, \forall s \in S$: $s_1 \stackrel{t}{\Rightarrow} s \Leftrightarrow s_2 \stackrel{t}{\Rightarrow} s$.
Par suite, ces états sont trivialement bisimilaires.

En termes de STE quotient, un corollaire de cette propriété est que si un couple d'états (s_1 et s_2 est relié par un τ-cycle alors toutes les transitions élémentaires constituant ce τ-cycle sont "internes" à la classe d'équivalence de s_1 (i.e celle de s_2). En d'autres termes, une projection observationnelle fait abstraction de tout τ-cycle.

Exemple 10 Equivalence observationnelle et divergence

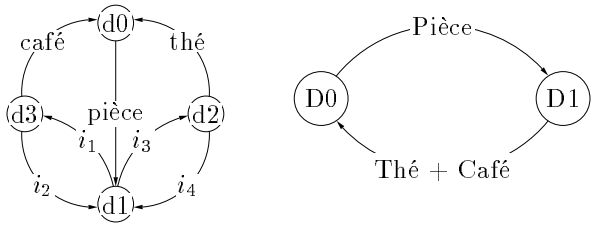


Figure 14: $\langle Div, 0\rangle$ et son STE quotient

On considère la machine à café $\langle Div, do\rangle$ représentée Figure 14. On observe, comme jusqu'à présent $\mathcal{O} = \{\text{Pièce}, \text{Thé}, \text{Café}\}$. Les états $d1, d2$ et $d3$ sont reliés

par un τ-cycle[5] et en appliquant la propriété 51, il vient que $d1 \sim_{\mathcal{O}} d2 \sim_{\mathcal{O}} d3$. Par suite, on obtient $\sim_{\mathcal{O}} = \{\{d0\}, \{d1, d2, d3\}\}$. En notant $D_0 = \{do\}$ et $D_1 = \{d1, d2, d3\}$, on obtient le STE quotient représenté à coté de $\langle Div, do\rangle$ Toutes les transitions inobservables appartiennent à des τ-cycles et sont donc "internalisées" dans des classes d'équivalences.

On a déjà montré (cf section 13) que les STE $\langle D, 0\rangle$ et $\langle D', 0'\rangle$ représentés ci-dessous étaient faiblement bisimilaires. On montrerait de même que $\langle D, 0\rangle \sim_{\mathcal{O}} \langle Div, 0\rangle$. Par transitivité, on a donc 3 STE en bisimulation faible. Pour autant, leurs comportements sont fortement différents : $\langle D, 0\rangle$ et $\langle D', 0'\rangle$ délivrent "inévitablement" une boisson après qu'elle ait été payée, tandis que pour $\langle Div, 0\rangle$, la livraison d'une boisson n'est que "potentielle".

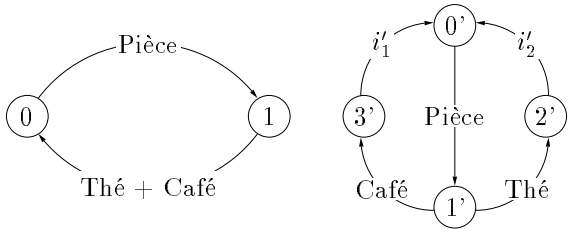


Figure 15: $\langle D, 0\rangle$ et $\langle D', 0'\rangle$: 2 STE bisimilaires à $\langle Div, do\rangle$

4.3. *Caractérisations modales des équivalences de comportement*

La vérification logique met en oeuvre des propriétés explicitement énoncées dans un langage spécifique possédant une sémantique précise. Vérifier que le système satisfait ces propriétés revient à s'assurer que le système est un modèle de ces propriétés.

L'approche comportementale ne manipule que des comportements. On étudie l'équivalence entre les comportements du système et de sa spécification. On déduit de cette équivalence que le système satisfait (modulo le critère d'abstraction retenu) ses spécifications sans jamais avoir explicitement énoncé les propriétés associées à la spécification ou préciser la nature des propriétés préservées par l'équivalence utilisée.

Le travail de M. Hennessy et R.Milner [HM 85] permet de mieux comprendre les liens entre ces deux approches de la vérification et mieux cerner ce qui les unit. Il fournit en particulier une définition "logique" des équivalences de comportement qui permet de préciser le type de propriétés que l'on peut vérifier.

[5] Plus précisément, on a ici 2 τ-cycles "élémentaires" s'intersectant sur $d1$

4.3.1. *Définition de $\mathcal{HML}$*

De façon intuitive, une logique peut être associée à une équivalence de comportement donnée (on parle de logique adéquate) de telle sorte que les comportements équivalents sont les comportements satisfaisant les mêmes énoncés de la logique adéquate.

Définition 52 *$\mathcal{HML}$: Logique de Hennessy-Milner*

Syntaxe : *$\mathcal{HML}$ est le plus petit ensemble vérifiant :*
$true \in \mathcal{HML}$, $f, g \in \mathcal{HML} \Rightarrow f \wedge g, \neg f \in \mathcal{HML}$
$f \in \mathcal{HML}, a \in \Sigma \Rightarrow <a> f \in \mathcal{HML}$

Sémantique : *La sémantique des formules $\mathcal{HML}$ est définie vis à vis d'un STE $\mathcal{STE}$. Comme pour les logiques modales, on notera pour $s \in S$ et $f \in \mathcal{HML}$: $\mathcal{STE}, s \models f$ pour indiquer que la formule f est satisfaite dans l'état s de la structure (ici du STE) $\mathcal{STE}$. Comme usuellement, la sémantique d'une formule de $\mathcal{HML}$ est définie par induction sur la structure des termes. Dans ce qui suit : f et $g \in \mathcal{HML}$ et $a \in \Sigma$.*

$\models$, la relation de satisfaction, est la plus petite relation vérifiant :
$\mathcal{STE}, s \models true \quad \forall s \in S$
$\mathcal{STE}, s \models f \wedge g$ *ssi* $\mathcal{STE}, s \models f$ *et* $\mathcal{STE}, s \models g$
$\mathcal{STE}, s \models \neg f$ *ssi Non* $(\mathcal{STE}, s \models \neg f)$
$\mathcal{STE}, s \models <a> f$ *ssi* $\exists s' \in S$ *tel que* $s \xrightarrow{a} s'$ *et* $\mathcal{STE}, s' \models f$

Abréviations : $false \equiv \neg true$, $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $[a]f \equiv \neg <a> \neg f$
$<\sigma> f \equiv <a_1><a_2> \ldots <a_n> f \qquad$ *pour* $\sigma = a_1.a_2.\ldots.a_n$

La sémantique d'une formule de $\mathcal{HML}$ (i.e $\models$, la relation de satisfaisabilité) est définie comme pour les logiques modales en considérant les STE comme des structure de Kripke. Deux différences sont à noter : $\mathcal{HML}$ ne fait pas intervenir de propositions atomiques. Plus exactement, l'ensemble de variables propositionnelles est réduit à la variable *true* qui est vraie dans tout état. En reprenant les notations introduites dans la section 2 on aurait : $\mathcal{P} = \{true\}$ et $\nu(s) = \{true\} \quad : \forall s \in S$. En revanche, la relation d'accessibilité entre les "mondes" de la structure est maintenant étiquetée.

Exemple 11 Exemples de propriétés exprimées en $\mathcal{HML}$

$\mathcal{STE}, s \models <a> true$ *Une a-expérimentation est possible à partir de s*

$\mathcal{STE}, s \models <a> (<b> true \wedge <c> true)$
A partir de s, une a-expérimentation est possible menant dans un état pour lequel à la fois sont possibles une b-expérimentation et une c-expérimentation.

$\mathcal{STE}, s \models [a] false$ *Aucune a-expérimentation n'est possible partant de s*

On considère de nouveau les STE $\langle D, 0\rangle, \langle D', 0'\rangle$ et $\langle D'', 0''\rangle$ représentés figure 6 et les énoncés $F_i \ : i \in [1,4]$ ci-dessous. La table ci-dessous donne les résultats de l'évaluation des formules F_i pour chacun des STE.

$F_1 \equiv <a>\ [b]F$
$F_2 \equiv <a>\ (<b>\ T \wedge <c>\ T)$
$F_3 \equiv [a](<b>\ T \wedge <c>\ T)$
$F_4 \equiv <a>\ (\ (<b>\ T \wedge [c]F) \vee (<c>\ T \wedge [b]F)\ )$

$\models$	F_1	F_2	F_3	F_4
$D, 0$	F	V	V	F
$D', 0'$	V	F	F	V
$D'', 0''$	V	V	F	F

4.3.2. *Caractérisation Modale de la bisimulation*

Théorie d'un état : Pour une logique $\mathcal{L}$, on note $TH \ : S \mapsto \mathcal{L}$, l'application qui associe à un état s de S l'ensemble des énoncés f de $\mathcal{L}$ qu'il satisfait (sa théorie). $TH_{\mathcal{L}}(s) =_{Def} \{f \in \mathcal{L} \ : s \models f\}$

Propriété 53 *Théorème de Hennessy-Milner [HM 85]*

$\mathcal{HML}$ caractérise **Modalement** *la bisimulation. Deux états sont bisimilaires s'ils satisfont les mêmes énoncés de la logique $\mathcal{HML}$.*

$s \sim q$ ssi $TH_{\mathcal{HML}}(p) = TH_{\mathcal{HML}}(q)$

Exemple 12 Retour sur les exemples 6, 7 et 5

Les STE de la figure 6 ne sont pas bisimilaires. En reprenant les énoncés de la table 11, on peut conclure que ces STE sont 2 à 2 non équivalents observationnellement :
- D est le seul STE satisfaisant F_3 donc D n'est équivalent ni à D' ni à D''.
- D' est le seul STE satisfaisant F_4 donc D' n'est équivalent ni à D ni à D''.

Les STE de la figure 7 ne sont pas bisimilaires. Considérons l'énoncé suivant de $\mathcal{HML}$: - $G \equiv$<Pièce> ((<Café> <Sucre> T) $\wedge$ (<Café> [Sucre]F)). On a $M, 0 \not\models G$ tandis que $M', 0' \models G$

Les STE de la figure 5 ne sont pas bisimilaires. Soit $f \equiv$ [Pièce](<Café> $true \wedge$ <Thé> $true$). f peut se formuler ainsi : après l'action Pièce on a toujours la possibilité d'exécuter les actions Café et Thé. On a $M', 0' \models f$ tandis que $M'', 0'' \not\models f$.

4.3.3. *$\mathcal{HML}$ et proposition d'états*

Comme nous l'avons vu la logique $\mathcal{HML}$, dans son énoncé originel, n'est basée que sur les événements, la relation de π-bisimulation (cf définition 37)

peut être utilisée pour permettre de prendre en compte explicitement la notion d'états. Au lieu de considérer uniquement un système de transitions étiquetées, nous considérons maintenant une structure de Kripke étiquetée
$\mathcal{SKE} = \langle AP, \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}, \nu\rangle$ où à chaque état de S, la valuation ν associe un ensemble de variables propositionnelles $\in 2^{AP}$ (cf définition 1).

Définition 54 $\mathcal{HML}(AP)$

On note $\mathcal{HML}(AP)$ l'extension de $\mathcal{HML}$ aux propositions de AP définie ainsi :

Syntaxe : *$\mathcal{HML}(AP)$ est le plus petit ensemble vérifiant :*
$AP \subset \mathcal{HML}(AP)$, $f, g \in \mathcal{HML}(AP) \Rightarrow f \wedge g \in \mathcal{HML}(AP), \neg f \in \mathcal{HML}(AP)$
$f \in \mathcal{HML}(AP), a \in \Sigma \Rightarrow <a> f \in \mathcal{HML}(AP)$

Sémantique : *La sémantique des formules $\mathcal{HML}(AP)$ est définie vis à vis d'une structure de Kripke étiquetée $\mathcal{SKE} = \langle AP, \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}, \nu\rangle$*

$\models$, la relation de satisfaction, est la plus petite relation vérifiant :
$\mathcal{SKE}, s \models P$ *ssi* $P \in \nu(P)$
$\mathcal{SKE}, s \models f \wedge g$ *ssi* $\mathcal{SKE}, s \models f$ *et* $\mathcal{SKE}, s \models g$
$\mathcal{SKE}, s \models \neg f$ *ssi Non* $(\mathcal{SKE}, s \models \neg f)$
$\mathcal{SKE}, s \models <a> f$ *ssi* $s \xrightarrow{a} s'$ *et* $\mathcal{SKE}, s' \models f$

Propriété 55 *Caractérisation modale de la π-bisimulation*

En reprenant les notations introduites dans la définition 33, on considère la partition $\pi^\nu(S)$ de S définie par l'application ν et on considère la π-bismulation associée (cf def 37).

$\mathcal{HML}(AP)$ caractérise modalement la $\pi^\nu(S)$-bisimulation : deux états sont $\pi^\nu(S)$- bisimilaires s'ils satisfont les mêmes énoncés de la logique $\mathcal{HML}(AP)$.

$$\forall p, q \in S : p \sim_{\pi^\nu} q \text{ ssi } TH_{\mathcal{HML}(AP)}(q) = TH_{\mathcal{HML}(AP)}(q)$$

Exemple 13 $\mathcal{HML}$ et divergence

Nous avons vu que l'équivalence observationnelle masquait les évolutions divergentes (cf 4.2.5) : il s'ensuit que la notion d'événement inévitable n'est pas exprimable en $\mathcal{HML}$.

En reprenant les notations de la section 3.3, on note $\langle \mathcal{M}, s\rangle \models \mathbf{AFX}_{\{t\}} true$ pour signifier que l'exécution de l événement t est inévitable à partir de l'état s. La propriété d'adéquation (prop 53) nous fournit un moyen simple pour montrer que $\mathbf{AFX}_{\{t\}} true$ n'est pas exprimable en $\mathcal{HML}$.

Considérons le STE représenté figure 16. On a $1 \models \mathbf{AFX}_{\{A\}} true$, $0 \not\models \mathbf{AFX}_{\{A\}} true$, de toute évidence nous avons $1 \sim 0$ et par suite $TH_{\mathcal{HML}}(1) = TH_{\mathcal{HML}}(0)$. Forcément $\mathbf{AFX}_{\{t\}} true$ ne peut s'exprimer en $\mathcal{HML}$.

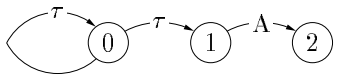


Figure 16: Divergence et Inévitabilité

Considérons de nouveau les trois machines à café représentées figure 10, tous les états de celles-ci satisfont la propriété $\mathbf{AFX}_{\{\text{Thé,Café}\}} true$, et en particulier les états 1, 1'et 1" qui correspondent à des états où une boisson a été payée et non encore délivrée. Considérons le STE $\langle Div, do\rangle$ présenté figure 14, aucun des états de STE ne satisfait $\mathbf{AFX}_{\{\text{Thé,Café}\}} true$ et en particulier l'état $d1$. Le fait d'avoir payé la boisson ne garantit pas qu'on l'obtienne en un temps fini.

L'extension de $\mathcal{HML}$ aux propositions d'états (cf 4.3.3), nous fournit un moyen simple, dans le cas des STE finis, d'étendre $\mathcal{HML}$ pour prendre en compte la notion de divergence.

Soit $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ et $\mathcal{O} \subset \Sigma$ un sous-ensemble de labels observables. On note $Div(S)$ le sous-ensemble de S défini ainsi : $Div(S) =_{Def} \{s \in S : \exists \omega \in \mathcal{O}^{\infty} \text{ et } s \stackrel{\omega}{\rightarrow}\}$. Ainsi, pour le STE de la figure 10 nous avons $Div = \{d1, d2, d3\}$.

On considère un ensemble de variables propositionnelles $AP_{Div} =_{Def} \{true, Div\}$ et la valuation ν définie par $true \in \nu(s)\ \forall s \in S$ et $Div \in \nu(s)$ ssi $s \in Div(S)$. Par construction la logique $\mathcal{HML}(AP_{Div})$ et $\sim_{Div}$, la $\pi-$bisimulation qui lui est associée, sont sensibles à la divergence.

A titre d'exemple nous pouvons comparer les STEI $\langle M', 0'\rangle$ et $\langle Div, do\rangle$ présentés respectivement figures 10 et 14. On obtient :

$\sim_{Div\ 0} = \{\{d1, d2, d3\}, \{d0, 0', 1', 2', 3'\}\}$
$\sim_{Div\ 1} = \{\{d1, d2, d3\}, \{d0, 0', 2', 3'\}, \{1'\}\}$
$\sim_{Div\ 2} = \{\{d1, d2, d3\}, \{d0\}\{0', 2', 3'\}, \{1'\}\}$
$\sim_{Div\ 3} = \sim_{Div\ 2}$ et par suite $\sim_{Div} = \sim_{Div\ 2}$

Les états initiaux respectifs des deux STE $(0, d0)$ ne sont pas dans la relation de bisimulation $(0 \not\sim_{Div} d0)$ et par suite $\langle M', 0'\rangle$ et $\langle Div, do\rangle$ ne sont pas "Div"-bisimilaires.

4.3.4. *Caractérisations modales d'autres relations d'équivalence*

On considère le sous-ensembles de $\mathcal{HML}$ définis ainsi :

$\mathcal{M} =_{Def} \{f \in \mathcal{HML}, f$ ne contenant pas de $\wedge\}$ et

$\mathcal{N} =_{Def} \{f \in \mathcal{M}, f$ ne contenant pas de $\neg\}$

[HM 85] montrent que que $\mathcal{M}$ caractérise **modalement** la relation de co-simulation l'équivalence langage

L'inclusion stricte entre $\mathcal{N}$ et $\mathcal{HML}$ traduit le fait que l'équivalence langage est strictement plus grossière que l'équivalence observationnelle. Ainsi, $\mathcal{N}$ ne permet plus d'exprimer $[A]$ ou $false$ qui sont essentielles pour définir des propriétés de deadlock : l'équivalence observationnelle préserve les blocages ce qui n'est pas le cas pour l'équivalence langage. L'équivalence observationnelle ne préserve pas la divergence mais peut être renforcée à cet effet [NV 90].

A contrario, le travail de [BCG 91] s'attache à caractériser "comportementalement" la logique $\mathcal{CTL}^*$. La relation d'équivalence opère maintenant entre des structures de Kripke présentées définition 1. Sa présentation est très proche des $\sim_n$ équivalences donnée définition 31.

Définition 56 *Equivalences de structures de Kripke*

Soient $\mathcal{SK} = \langle AP, S, \rightarrow, \nu\rangle$ *et* $\mathcal{SK} = \langle AP, S', \rightarrow', \nu'\rangle$ *deux structures de Kripke partageant le même ensemble de variables propositionnelles* AP

On définit une suite de relations d'équivalences E_{K_0} , E_{K_1} , ... *sur* $S \times S'$ *de façon suivante :*

$s\ E_{K_0}\ s'$ *ssi* $\nu(s) = \nu'(s')$

$s\ E_{K_{n+1}}\ s'$ *ssi*

1. $\nu(s) = \nu'(s')$
2. $\forall s_1 \in S\ : (s \rightarrow s_1) \Rightarrow \exists s'_1 \in S' : s' \rightarrow s'_1$ *et* $s\ E_{K_n}\ s'$
3. $\forall s'_1 \in S' : (s' \rightarrow s'_1) \Rightarrow \exists s_1 \in S : s \rightarrow s_1$ *et* $s_1\ E_{K_n}\ s_1$

Finalement, l'équivalence entre structures de Kripke est définie ainsi :
$s\ E_K\ s'$ *ssi* $s\ E_{K_i}\ s' : \forall i \geq 0$

La caractérisation comportementale de la logique $\mathcal{CTL}^*$est donnée par la propriété suivante :

Propriété 57 *Caractérisation comportementale de* $\mathcal{CTL}^*$

$s\ E_K\ s' \Rightarrow \forall f \in \mathcal{CTL}^*[s \models f \Leftrightarrow s' \models f]$

[BCG 91] introduit aussi l'équivalence bégayante (stuttering) qui caractérise comportementalement logique temporelle $\mathcal{CTL}^*\_X$, à savoir la logique $\mathcal{CTL}^*$ privée de son nexttime opérateur.

5. Décidabilité de la bisimulation et de l'évaluation de formules

Nous allons maintenant exposer les résultats fondamentaux concernant la décidabilité de la bisimulation de réseaux de Petri et de l'évaluation de formules de logique temporelle pour un réseau de Petri. Au chapitre 4 du volume traitant des Réseaux de Petri [HAD 01], nous avons vu que les propriétés génériques étaient toutes décidables (caractère borné, accessibilité,...). De plus la complexité de la vérification de certaines propriétés est complètement caractérisée (le caractère borné est $\mathcal{EXP}space$-complet) tandis que pour d'autres, le problème reste ouvert (l'accessibilité est $\mathcal{EXP}space$-difficile mais l'algorithme de décision n'est pas primitif récursif). Comme les sections précédentes l'ont montré, la logique temporelle et la bisimulation permettent de caractériser le comportement d'une structure de Kripke étiquetée de manière plus fine qu'à travers les propriétés génériques. Aussi on peut s'attendre à ce que les procédures de décision soient plus difficiles à obtenir. Par exemple, le problème de l'accessibilité s'exprime facilement en $\mathcal{LTL}$ et en $\mathcal{CTL}$.

Exemple

- en $\mathcal{LTL}$,
  m non accessible depuis $(R, m_0) \Leftrightarrow (R, m_0) \models \mathbf{G}\ \mathbf{OR}_{p \in P}\ p \neq m(p)$
- en $\mathcal{CTL}$,
  m non accessible depuis $(R, m_0) \Leftrightarrow (R, m_0) \models \mathbf{AG}\ \mathbf{OR}_{p \in P}\ p \neq m(p)$

Il s'avère effectivement que selon la variante adoptée, certains problèmes sont indécidables tandis que, pour les variantes décidables, la plupart des méthodes de décision reposent sur la décidabilité de l'accessibilité impliquant par la même une grande complexité. L'obtention des résultats d'indécidabilité et de décidabilité sont de nature très différente. Nous suivrons donc ce découpage dans la suite.

5.1. *Résultats d'indécidabilité*

La technique usuelle pour démontrer qu'un problème est indécidable consiste à réduire un autre problème indécidable au problème initial. Cette technique

suppose que l'on a préalablement déterminé d'une autre manière l'indécidabilité d'un problème. Le problème plus général que nous étudions est celui de l'arrêt d'un programme.

Théorème 58 (Arrêt d'un programme à paramètres) *Le problème de l'arrêt d'un programme prog, prenant en entrée un paramètre entier x est indécidable.*

Preuve
Nous allons démontrer ce résultat par l'absurde. Supposons qu'il existe un programme $testarret$ prenant en entrée deux paramètres entiers : une représentation (par un entier) d'un programme $prog$ et une valeur d'entrée de ce programme. Le choix de la représentation du programme est ici sans importance ; par exemple, on pourrait choisir comme représentation le nombre entier correspondant à la suite de bits du programme. On notera $\overline{prog}$ cette représentation. $testarret$ renvoie vrai si $prog$ s'arrête avec la valeur fournie et sinon renvoie faux. Le comportement de $testarret$ est indéterminé si le premier paramètre n'est pas la représentation d'un programme.
Nous construisons alors un programme fou à un paramètre entier qui fonctionne ainsi.

- fou vérifie que son paramètre x est bien la représentation d'un programme $prog$ (comme le fait un compilateur). Si ce n'est pas le cas, il s'arrête.
- fou appelle $testarret(x, x)$. Autrement dit, il teste si le programme $prog$ s'arrête en prenant comme entrée sa représentation.
- Si $testarret(x, x)$ renvoie vrai, alors fou boucle sans fin sinon il s'arrête.

Examinons le comportement de $fou(\overline{fou})$.
Si $fou(\overline{fou})$ s'arrête alors $testarret(\overline{fou}, \overline{fou})$ renvoie vrai et par conséquent $fou(\overline{fou})$ ne s'arrête pas ce qui est absurde.
Dans le cas contraire, $testarret(\overline{fou}, \overline{fou})$ renvoie faux et par conséquent $fou(\overline{fou})$ s'arrête ce qui est absurde. Il n'existe donc pas de programme $testarret$. ◇

Le fait que le programme ait un paramètre en entrée est tout à fait accessoire comme l'indique le corollaire suivant. Par ailleurs, celui-ci illustre le principe de réduction.

Corollaire 59 (Arrêt d'un programme sans paramètre) *Le problème de l'arrêt d'un programme prog sans paramètre est indécidable.*

Preuve
Montrons que le problème de l'arrêt d'un programme à un paramètre est réductible au problème de l'arrêt d'un programme sans paramètre. Nous supposons donc qu'il existe un programme $testarretbis$ pour le problème du corollaire

et nous décrivons comment construire un programme $testarret$. Soit $prog$ un programme à un paramètre et x une valeur entière. Alors $testarret$ fonctionne comme suit :

- $testarret$ construit la représentation du programme $prog'$ sans paramètre qui consiste à appeler $prog(x)$.
- Puis $testarret$ appelle $testarretbis(\overline{prog'})$ et renvoie le résultat correspondant.

Donc $testarretbis$ ne peut exister. ♢

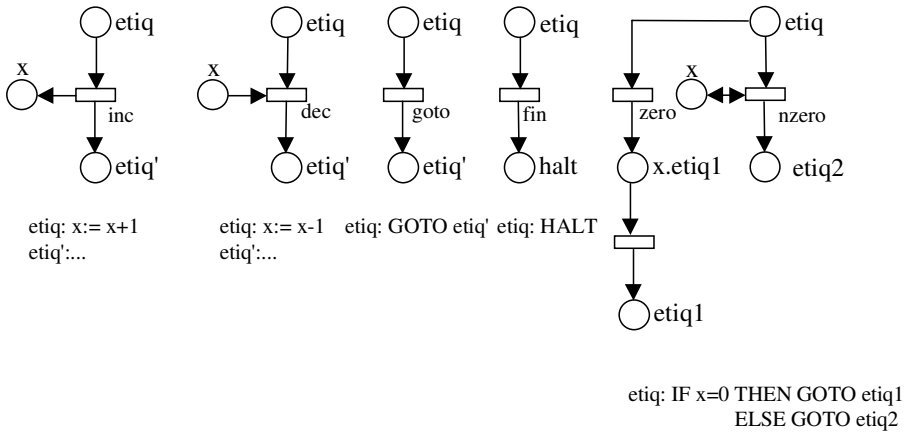


Figure 17: Simulation faible d'un programme à compteurs

Le choix du langage de programmation (ou du modèle de calcul) est indifférent à partir du moment où celui-ci possède les constructeurs minimaux lui conférant une expressivité équivalente aux machines de Turing. Pour s'appuyer sur les résultats précédents, on recherche des langages adaptés aux problèmes étudiés. Dans notre cas, nous choisirons le modèle des *programmes à compteurs*. Les variables d'un programme à compteurs sont des entiers positifs initialisés à 0 appelés compteurs. Le programme est une suite d'instructions ; chaque instruction est précédée d'une étiquette (à la manière du langage *Basic*). Les différents types d'instruction sont :

- l'incrémentation $etiq : x := x + 1$
- la décrémentation $etiq : x := x - 1$
  Lorsque la décrémentation est appliquée à un compteur nul, elle provoque un avortement du programme **considéré comme différent de l'arrêt**.
- le branchement inconditionnel $etiq$: **GOTO** $etiq'$
- le branchement conditionnel
  $etiq$: **IF** $x = 0$ **THEN GOTO** $etiq1$ **ELSE GOTO** $etiq2$

- la terminaison $etiq$: **HALT**
  Cette instruction est nécessairement la dernière instruction du programme.

Ce programme *prog* possède une seule exécution (démarrant à la première instruction) qui peut soit être infinie, soit avorter soit s'arrêter lorsque le programme atteint la dernière instruction. Nous allons proposer une simulation faible d'un programme à compteurs par un réseau de Petri noté R_{prog} (ce nom s'appliquera aussi aux différentes variantes de la simulation). On associe à chaque étiquette, une place qui lorsqu'elle est marquée indique que l'instruction est la prochaine instruction à exécuter ; initialement seule la place de l'étiquette de la première instruction contient un jeton. Chaque compteur est traduit par une place initialement non marquée. La traduction des instructions introduit les transitions comme indiqué à la figure 5.1. Chaque transition est étiquetée par le type d'instruction (inc, dec, $goto$, fin, $zero$, $nzero$). La simulation est faible au sens où une transition étiquetée $zero$ peut être franchie bien que la place x soit marquée. Une simulation exacte nécessiterait un arc inhibiteur de la place x vers la transition étiquetée $zero$. Autrement dit, parmi les séquences maximales (finies ou infinies), une seule d'entre elles correspond à une simulation exacte du programme tandis que les autres séquences "trichent" en franchissant de manière inconsidérée au moins une transition étiquetée $zero$ alors que le compteur correspondant est non nul. Il est alors immédiat que :

prog s'arrête

$\Leftrightarrow$

Toutes les séquences maximales de R_{prog} "trichent" ou marquent la place *halt*

Ceci nous conduit aux premiers résultats d'indécidabilité.

Théorème 60 (Evaluation d'une formule propositionnelle $\mathcal{LTL}$ ou $\mathcal{CTL}$)
Dans un réseau de Petri, le problème de l'évaluation d'une formule propositionnelle $\mathcal{LTL}$ ou $\mathcal{CTL}$ est indécidable.

Preuve
Il nous suffit d'exprimer le deuxième terme de l'équivalence.
Soit en $\mathcal{LTL}$: **F** (**OR**$_{x.etiq \in P}$ ($x.etiq = 1$ **AND** $x > 0$) **OR** $halt = 1$)
Et en $\mathcal{CTL}$: **AF** (**OR**$_{x.etiq \in P}$ ($x.etiq = 1$ **AND** $x > 0$) **OR** $halt = 1$)

♢

Remarquons que ce résultat est présenté dans le cadre d'une sémantique sur les séquences maximales finies ou infinies. On peut facilement se restreindre aux séquences infinies en ajoutant une transition qui boucle autour de la place *halt*. Cette remarque vaut pour le résultat suivant.

En modifiant la traduction du branchement conditionnel comme indiqué sur la figure 18, nous obtenons un résultat complémentaire.

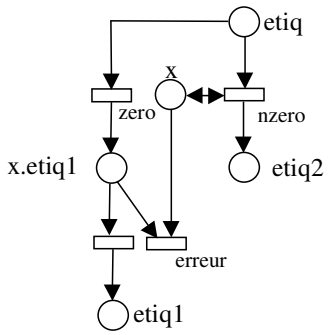


Figure 18: Une autre simulation faible du branchement conditionnel

Théorème 61 (Evaluation d'une formule événementielle $\mathcal{CTL}$) *Dans un réseau de Petri, le problème de l'évaluation d'une formule événementielle $\mathcal{CTL}$ est indécidable.*

Preuve
L'équivalence indiquée plus haut reste vérifiée et il nous suffit encore d'exprimer le deuxième terme de l'équivalence dans la logique événementielle $\mathcal{CTL}$. Soit :
AF (**EX**$_{\{erreur\}}$ *true* **OR EX**$_{\{fin\}}$ *true*)

$\diamondsuit$

Remarquons que l'opérateur de la logique arborescente **EX** permet de tester la franchissabilité ce qui n'est pas possible avec une logique linéaire événementielle.

Nous allons maintenant transformer une nouvelle fois notre simulation faible pour traiter le cas de la bisimulation. Nous ajoutons à notre réseau deux nouvelles places "complémentaires" y et y' de telle sorte que qu'un jeton soit présent soit dans y soit dans y' mais jamais simultanément dans les deux places. Etant donné un marquage m de cette nature, on notera $\overline{m}$ le marquage obtenu en inversant le contenu de y et de y'. On modifie une nouvelle fois le branchement conditionnel mais aussi la dernière instruction comme indiqué sur la figure 19.

Le marquage initial m_0 est défini par un jeton dans l'étiquette de la première instruction et un jeton dans la place y. Ici encore, l'une des séquences d'exécution maximales correspond à la simulation du programme à compteurs. Lorsqu'on "dévie" de la simulation exacte en franchissant une transition étiquetée *zero* alors que le compteur correspondant est marqué, on peut soit inverser

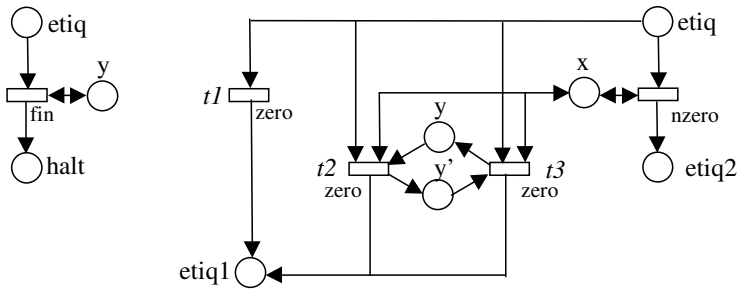


Figure 19: Une troisième simulation faible

le contenu de y et y'(par t_2 ou t_3), soit le laisser inchangé (par t_1). Les transitions de type t_2 et t_3 ne sont pas utilisées par la simulation exacte.

Théorème 62 (Bisimulation de réseaux de Petri) *Le problème de la bisimulation de deux réseaux marqués (R, m_0) et (R', m'_0) est indécidable.*

Preuve
Nous allons démontrer qu'un programme à compteurs *prog* s'arrête si et seulement si (R_{prog}, m_0) et $(R_{prog}, \overline{m}_0)$ ne se bisimulent pas.

1. *prog* **s'arrête**
Supposons que m_0 et $\overline{m}_0$ se bisimulent. Soit $m_0, m_1, \ldots, m_n$ la suite des maquages correspondant à la simulation exacte de *prog*. Nous démontrons par récurrence que pour $i < n$, m_i et $\overline{m}_i$ se bisimulent. Puisque m_i correspond à une partie de la simulation exacte de *prog*, on peut parler de la prochaine instruction à exécuter. Si cette instruction est une incrémentation, une décrémentation, un branchement inconditionnel ou la branche non zéro d'un branchement conditionnel alors une unique transition portant l'étiquette correspondante est franchissable pour m_i et $\overline{m}_i$ conduisant à m_{i+1} et $\overline{m}_{i+1}$. Si cette instruction correspond à la branche zéro d'un branchement conditionnel, alors ici aussi une seule transition (t_1) est franchissable pour m_i et $\overline{m}_i$ puisque le compteur testé (x) n'est pas marqué ; ce franchissement conduisant encore à m_{i+1} et $\overline{m}_{i+1}$. Examinons maintenant m_{n-1} et $\overline{m}_{n-1}$. La transition étiquetée par fin est franchissable à partir de m_{n-1} mais ne l'est pas à partir de $\overline{m}_{n-1}$ car y n'est pas marquée. Les marquages m_{n-1} et $\overline{m}_{n-1}$ ne se bisimulent pas et par conséquent il en est de même pour m_0 et $\overline{m}_0$.

2. *prog* **ne s'arrête pas**
Nous définissons une relation $\mathcal{R}$ qui contient $(m_0, \overline{m}_0)$ et nous démontrons que

c'est une relation de bisimulation. $\mathcal{R} = \{(m, m') \mid m = m'$ ou m est un marquage atteint par la simulation exacte de $prog$ et $m' = \overline{m}\}$. Bien entendu, il suffit de prouver que $\mathcal{R}$ est une relation de bisimulation uniquement pour le deuxième type de couple de marquages. Soit m un marquage atteint par la simulation exacte de $prog$. Puisque $prog$ ne s'arrête pas, la transition étiquetée par fin n'est pas franchissable à partir de m. Dans le cas d'un avortement, aucune transition n'est franchissable ni de m ni de $\overline{m}$. Dans le cas où la prochaine instruction à exécuter n'est pas la branche non zéro d'un branchement conditionnel, une unique transition (correspondant à la simulation) est franchissable à partir de m et de $\overline{m}$. Cette transition correspond à la simulation de $prog$ et par conséquent le couple de marquages atteints appartient à $\mathcal{R}$. Si la prochaine instruction est une branche non zéro, alors deux choix sont possibles à partir de m (respectivement de $\overline{m}$), continuer la simulation en franchissant la transition étiquetée par $nzero$ ou s'écarter de la simulation en franchissant une transition étiquetée par $zero : t_1$ ou t_2 (respectivement t_1 ou t_3). Montrons comment $\overline{m}$ simule m (la réciproque est symétrique).

- Si de m on franchit la transition étiquetée $nzero$, on fait de même à partir de $\overline{m}$ et le couple atteint correspond à la suite de la simulation de $prog$.
- Si de m on franchit la transition étiquetée t_1, on franchit t_3 à partir de $\overline{m}$ et les marquages atteints sont identiques.
- Si de m on franchit la transition étiquetée t_2, on franchit t_1 à partir de $\overline{m}$ et les marquages atteints sont identiques.

♢

Le lecteur attentif aura remarqué que la preuve s'applique à toute équivalence comprise entre l'équivalence de langage et la bisimulation puisque si $prog$ s'arrête alors les deux réseaux ne sont pas équivalents en terme de langage. De même du fait que, dans les réseaux marqués de la preuve, deux transitions t et t' ne sont jamais franchissables de manière concurrente (i.e. $m \geq Pré(t)+Pré(t')$), le résultat reste valide pour les équivalences qui prennent en compte le franchissement multiple.

5.2. *Résultats de décidabilité*

Au cours de ce paragraphe, nous ferons appel à différentes notions introduites au chapitre 4 du volume traitant des Réseaux de Petri [HAD 01] (ensemble semi-linéaire, technique des plus courtes séquences, ...). Nous conseillons fortement au lecteur de s'y reporter pour mieux apprécier ce qui suit.

5.2.1. *Formules de $\mathcal{LTL}$*

Nous étudions d'abord la vérification d'une formule de logique temporelle événementielle $\mathcal{LTL}$ et de manière plus générale de toute formule d'un langage (par exemple le μ-calcul linéaire) susceptible d'être représentée par un automate [DAM 92]. Les états terminaux s'interprètent de manière usuelle dans le cas d'une sémantique en terme de séquences finies ou comme ceux d'un automate de Büchi si la sémantique s'exprime à l'aide des séquences infinies.

La procédure de vérification dans le cas des systèmes finis consiste à :

- construire l'automate correspondant à la négation de la formule,
- effectuer le produit synchronisé du système de transitions étiqueté du modèle (i.e. le graphe d'accessibilité) avec l'automate,
- établir l'existence d'une séquence finie (respectivement infinie) atteignant (respectivement rencontrant une infinité de fois) un état terminal.

Cette procédure n'est évidemment pas possible dans le cas des systèmes de transitions infinis, mais la clef de la méthode que nous allons exposer consiste à construire un réseau de Petri qui engendre le système de transitions étiqueté produit puis à tester dans ce réseau l'existence d'une séquence adéquate.

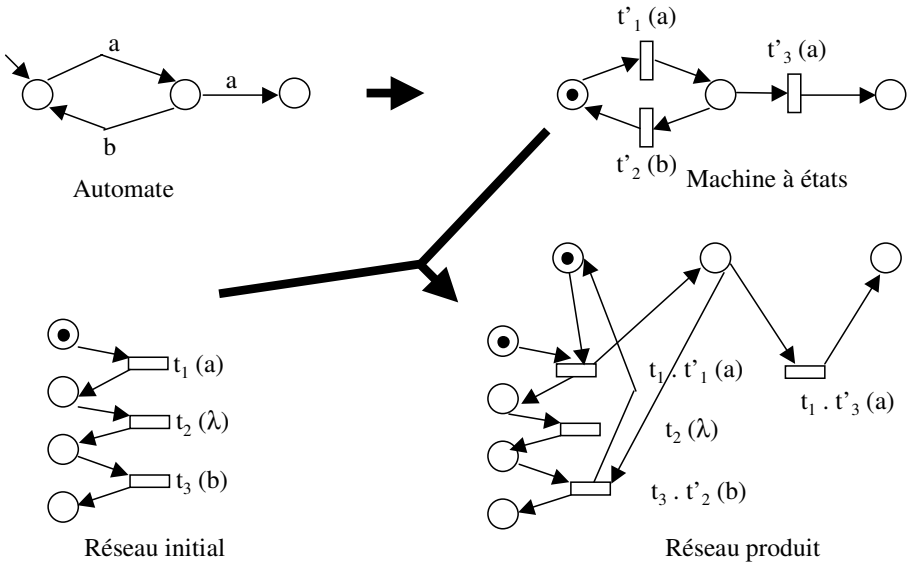


Figure 20: Produit synchronisé d'un RdP et d'un automate

La construction du réseau de Petri "produit" a déjà été vue au chapitre dans l'étude des langages et nous la rappelons ici brièvement (voir la figure 20).

- On transforme l'automate en un réseau de Petri (plus précisément en une machine à états mono-marquée),
- on construit un réseau "produit" à partir du réseau du modèle et de la machine à états comme suit :
- on définit l'ensemble des places comme l'union disjointe des places des deux réseaux et le marquage initial comme la somme des marquages initiaux,
- on crée une transition par couple de transitions de même étiquette. Les arcs entrée et sortie de cette transition sont obtenus par union des arcs des deux transitions dans les réseaux initiaux. On laisse inchangées les transitions du réseau initial étiquetées par le mot vide.

Il est immédiat que les traces observables des séquences de ce réseau sont exactement les mots engendrés par le réseau initial et reconnus par l'automate (sans tenir compte des états terminaux). Ceci est le point de départ de la méthode d'évaluation.

Théorème 63 (Evaluation d'une formule événementielle $\mathcal{LTL}$) *Dans un réseau de Petri, le problème de l'évaluation d'une formule événementielle $\mathcal{LTL}$ est décidable (et plus généralement d'une formule quelconque dont la négation est représentable par un automate).*

Preuve
Ce résultat est valide pour tous les types de séquences considérés : séquences finies, séquences maximales finies, séquences infinies, séquences divergentes. Nous nous limiterons aux trois premiers types, n'ayant pas défini précisément une sémantique de satisfaction pour les séquences divergentes.

1. Cas des séquences finies
On recherche l'existence d'une séquence finie dans le réseau produit qui marque une place correspondant à un état terminal de l'automate. Autrement dit, pour chacune de ces places, on cherche à couvrir le marquage constitué d'un jeton dans cette place. Le problème de la couverture a été traité de manière approfondie dans [HAD 01] et la méthode des plus courtes séquences nous fournit une procédure de complexité $\mathcal{EXP}space$.

2. Cas des séquences maximales finies
On recherche l'existence d'une séquence maximale finie dans le réseau produit qui marque une place correspondant à un état terminal de l'automate. On note ce sous-ensemble de places $Term$. Autrement dit, le réseau doit se bloquer dans un marquage où l'une des places de $Term$ est marquée. L'ensemble de ces marquages est un ensemble semi-linéaire calculable ($\cap_{t\in T}\{m \mid \mathbf{NOT}\ m \geq Pré(t)\} \cap \cup_{p\in Term}\{m \mid m \geq \overrightarrow{p}\}$. Il nous faut donc savoir si l'un quelconque des marquages d'un ensemble semi-linéaire est accessible. Puisqu'un ensemble

semi-linéaire est une union finie d'ensemble linéaire, on teste successivement l'accessibilité de chaque ensemble linéaire. Enfin pour chaque ensemble linéaire $E = \{w \mid \exists\, \lambda_1, \ldots, \lambda_m \ \in \mathbb{N}, t.q.\ w = u + \sum_{i=1}^{m} \lambda_i.v_i\}$, on ajoute au réseau une transition t_i pour i de 1 à m telle que $Pré(t_i) = v_i$ et $Post(t_i) = \overrightarrow{0}$. Puis on teste, dans ce réseau modifié, si u est accessible. Ce qui malheureusement se fait par un algorithme non primitif récursif.

3. Cas des séquences infinies
On recherche l'existence d'une séquence infinie dans le réseau produit qui produit une marque une infinité de fois dans une place correspondant à un état terminal de l'automate. Ce qui signifie qu'une des transitions, entrée d'une de ces places, est franchie une infinité de fois. Nous sommes ramenés au problème de l'existence d'une séquence infinie comprenant une infinité d'occurences d'une transition t donnée. Autrement dit une telle séquence σ doit s'écrire $\sigma = \sigma_1.\sigma_2.\ldots.\sigma_i.\ldots$ avec t apparaissant dans chaque σ_i. A l'aide du lemme d'extraction du chapitre 3 de [HV 01] appliqué aux marquages intermédiaires atteints par les séquences $\sigma_1.\sigma_2.\ldots.\sigma_i$, on en déduit que l'existence d'une telle séquence infinie est équivalente à l'existence d'une séquence $m_0[\sigma_1\rangle m_1[\sigma_2\rangle m_2$ avec $m_1 \leq m_2$ et t apparaissant dans σ_2. Enfin en ajoutant une place sortie p_t à t, on se ramène, dans ce réseau modifié, à la recherche d'une séquence $m_0[\sigma_1\rangle m_1[\sigma_2\rangle m_2$ avec $m_1 \leq m_2$ et $m_1(p_t) < m_2(p_t)$. Ce dernier problème se résout aussi avec la technique des plus courtes séquences (voir [RAC 78, YEN 92] pour plus de détails) et conduit à nouveau à une procédure de complexité $\mathcal{EXP}space$. ♢

Une question intéressante est de savoir si cette décidabilité est préservée pour les extensions de réseau de Petri. Il s'avère que cette évaluation devient indécidable pour la quasi-totalité des extensions usuelles. Tel est le cas par exemple, pour les réseaux de Petri récursifs et même pour des modèles restreints [BOU 96]. Cependant avec une sémantique "séquentielle" du franchissement d'une transition abstraite, ce problème reste décidable [HAD 00].

5.2.2. *Bisimulation*

Nous étudions maintenant la bisimulation d'un réseau marqué et d'un système de transitions fini. Nous allons utiliser les $\sim_n$-équivalences, introduites par la définition 31, permettant de caractériser dans le cas de STE finis la bisimulation (cf prop 32).

Pour qu'íl n'y ait pas d'ambiguïté dans nos notations, en particulier vis à vis de la provenance des états que nous considèrerons, nous mentionnerons explicitement le nom du STE. Ainsi nous noterons $\langle \mathcal{ST\!E}, s\rangle$ pour expliciter le fait que le sommet s est un sommet du STE $\mathcal{ST\!E}$.

Nous introduisons deux notations utiles pour les prochains développements. Soit $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ un système de transitions étiqueté,

- $Inc_n^{\mathcal{STE}}$ désigne l'ensemble des systèmes initialisés incompatibles avec $\mathcal{STE}$ pour $\sim_n$:
$$Inc_n^{\mathcal{STE}} = \{\langle \mathcal{STE}', s'\rangle \mid \forall s \in S \; \mathbf{NOT}\, \langle \mathcal{STE}', s'\rangle \sim_n \langle \mathcal{STE}, s\rangle)\}$$
- $\stackrel{*}{\longrightarrow}$ désigne la fermeture réflexive et transitive de l'union des $\stackrel{a}{\longrightarrow}$. Autrement dit, $\langle \mathcal{STE}, s\rangle \stackrel{*}{\longrightarrow} \langle \mathcal{STE}, s'\rangle$ si et seulement si s' est accessible depuis s.

Lemme 64 *Soient $\mathcal{STE} = \langle \Sigma, S, \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma}\rangle$ un système de transition étiqueté fini ($n_s = |S|$) et $\mathcal{STE}' = \langle \Sigma', S', \{\stackrel{a}{\rightarrow}\}_{a\in\Sigma'}\rangle$ un système de transitions étiqueté quelconque, alors : $\forall s \in S$, $\forall s' \in S'$,*

$$\langle \mathcal{STE}, s\rangle \sim \langle \mathcal{STE}', s'\rangle \Leftrightarrow \left\{ \begin{array}{l} \langle \mathcal{STE}, s\rangle \sim_{n_s} \langle \mathcal{STE}', s'\rangle \\ \textbf{\textit{AND}} \\ \nexists\, \langle \mathcal{STE}', s"\rangle \in Inc_{n_s}^{STE}\, t.q.\, \langle \mathcal{STE}', s'\rangle \stackrel{*}{\longrightarrow} \langle \mathcal{STE}', s"\rangle \end{array} \right.$$

Preuve
Pour l'implication de gauche à droite, si $\langle \mathcal{STE}, s\rangle \sim \langle \mathcal{STE}', s'\rangle$ alors d'après la propriété 32 $\langle \mathcal{STE}, s\rangle \sim_{n_s} \langle \mathcal{STE}', s'\rangle$. D'autre part, par une récurrence immédiate sur le nombre de transitions $\stackrel{a}{\longrightarrow}$ qui conduisent de $\langle \mathcal{STE}', s'\rangle$ à $\langle \mathcal{STE}', s"\rangle$ en utilisant la définition de $\sim$, on établit qu'il existe s_1 accessible depuis s (avec le même nombre de transitions) tel que $\langle \mathcal{STE}', s"\rangle \sim \langle \mathcal{STE}, s_1\rangle)$ et par conséquent $\langle \mathcal{STE}', s"\rangle \sim_{n_s} \langle \mathcal{STE}, s_1\rangle$.

Pour l'implication de droite à gauche, nous définissons la relation $\mathcal{R}$ suivante :

$$\langle \mathcal{STE}, s_1\rangle \,\mathcal{R}\, \langle \mathcal{STE}', s_1'\rangle \; ssi \left\{ \begin{array}{l} \langle \mathcal{STE}, s_1\rangle \sim_{n_s} \langle \mathcal{STE}', s_1'\rangle \\ \textbf{AND} \\ \nexists\, \langle \mathcal{STE}', s"\rangle \in Inc_{n_s}^{\mathcal{STE}}\, t.q.\, \langle \mathcal{STE}', s_1'\rangle \stackrel{*}{\longrightarrow} \langle \mathcal{STE}', s"\rangle \end{array} \right.$$

Montrons que $\mathcal{R}$ est une relation de bisimulation.
Supposons que $\langle \mathcal{STE}, s_1\rangle \stackrel{a}{\longrightarrow} \langle \mathcal{STE}, s_2\rangle$; alors il existe $\langle \mathcal{STE}', s_2'\rangle$ tel que $\langle \mathcal{STE}', s_1'\rangle \stackrel{a}{\longrightarrow} \langle \mathcal{STE}', s_2'\rangle$ et $\langle \mathcal{STE}, s_2\rangle \sim_{n_s-1} \langle \mathcal{STE}', s_2'\rangle$.
Puisque $\langle \mathcal{STE}', s_2'\rangle$ est accessible de $\langle \mathcal{STE}', s_1'\rangle$, on a :
$\nexists\, \langle \mathcal{STE}', s"\rangle \in Inc_{n_s}^{\mathcal{STE}}\, t.q.\, \langle \mathcal{STE}', s_2'\rangle \stackrel{*}{\longrightarrow} \langle \mathcal{STE}', s"\rangle$. Et en particulier, $\langle \mathcal{STE}', s_2'\rangle \notin Inc_{n_s}^{\mathcal{STE}}$. Donc il existe $\langle \mathcal{STE}, s_3\rangle \sim_{n_s} \langle \mathcal{STE}', s_2'\rangle$.

Par transitivité et le fait que $\sim_n \subset \sim_{n-1}$, $\langle \mathcal{STE}, s_3\rangle \sim_{n_s-1} \langle \mathcal{STE}, s_2\rangle$ et en vertu de la propriété 32, $\langle \mathcal{STE}, s_3\rangle \sim_{n_s} \langle \mathcal{STE}, s_2\rangle$. De nouveau par transitivité, on obtient donc $\langle \mathcal{STE}, s_2\rangle \sim_{n_s} \langle \mathcal{STE}', s_2'\rangle$.
Le cas $\langle \mathcal{STE}', s_1'\rangle \stackrel{a}{\longrightarrow} \langle \mathcal{STE}', s_2'\rangle$ est analogue. $\diamondsuit$

Cette caractérisation est à la base du résultat suivant.

Théorème 65 (Bisimulation d'un réseau et d'un système fini) *Le problème de la bisimulation d'un réseau marqué $\langle R, m_0\rangle$ dont aucune transition n'est étiquetée par le mot vide et d'un système de transition étiqueté fini $\langle \mathcal{STE}, s_0\rangle$ est décidable.*

Preuve
Comme précédemment $n_s = |S|$. Pour décider si $\langle R, m_0\rangle \sim_{n_s} \langle \mathcal{STE}, s_0\rangle$, il suffit de tester si :

- pour toute transition d'étiquette a franchissable, depuis m_0 et conduisant à m_1, s'il existe un s_1 tel que $\langle \mathcal{STE}, s_0\rangle \xrightarrow{a} \langle \mathcal{STE}, s_1\rangle$ et $\langle \mathcal{STE}, s_1\rangle \sim_{n_s-1} \langle R, m_1\rangle$,
- pour tout s_1 tel que $\langle \mathcal{STE}, s_0\rangle \xrightarrow{a} \langle \mathcal{STE}, s_1\rangle$, s'il existe une transition d'étiquette a franchissable, depuis m_0 et conduisant à m_1 tel que
  $\langle \mathcal{STE}, s_1\rangle \sim_{n_s-1} \langle R, m_1\rangle$.

Ceci conduit de manière évidente à une procédure récursive dont la profondeur d'appel est limitée à n_s.

Il nous reste à tester s'il existe un marquage accessible m_1 depuis m_0 tel que $m_1 \in Inc_{n_s}^{\mathcal{STE}}$. Etudions d'abord les marquages appartenant à $Inc_{n_s}^{\mathcal{STE}}$. D'après la procédure récursive précédente, pour tester $\sim_{n_s}$ on n'examine que des séquences de franchissement de longueur inférieure ou égale à n_s. Posons v la valuation maximum d'un arc de R et $B = v.n_s$. Prenons deux marquages m et m' tels que $\forall p \in P,\ m(p) \neq m'(p) \Rightarrow m(p) \geq B$ **AND** $m'(p) \geq B$. Ces deux marquages sont équivalents pour $\sim_{n_s}$. Soit donc un marquage m borné par B, posons $Sup^B(m) = \{m' \mid m' \geq m$ **AND** $\forall p \in P,\ m(p) < B \Rightarrow m'(p) = m(p)\}$. Clairement tous les marquages de $Sup^B(m)$ sont équivalents pour $\sim_{n_s}$. Notons qu'un marquage quelconque appartient nécessairement à $Sup^B(m)$ pour un m borné par B.

La procédure de décision opère alors en deux temps. D'abord elle examine pour chaque marquage m borné par B si $m \in Inc_{n_s}^{\mathcal{STE}}$ et ceci par la procédure précédente. Puis elle teste pour chaque m ainsi trouvé s'il existe $m' \in Sup^B(m)$ accessible depuis m_0. Cette recherche se réduit au test de l'accessibilité de m dans un réseau où on ajoute une transition t_p par place p telle que $m(p) = B$, cette transition étant définie par $Pré(t_p) = \overrightarrow{p}$ et $Post(t_p) = \overrightarrow{0}$. ♢

Dans [JAN 99], on pourra trouver des résultats plus techniques comme le test d'existence d'un système de transitions étiqueté fini qui bisimule un réseau de Petri.

Références

[ARN 92] A. Arnold Systèmes de transitions finis et sémantique des processus communicants Masson Paris, 1992.

[AHO 74] Alfred V. Aho, John E. Hopcroft et J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, 1974.

[BCG 91] M.C. Browne, E.M. Clarke, O. Grumberg Characterizing finite Kripke structures in propositional temporal logic *Theoretical Computer Science*, 59 :115-131, 1988

[BOU 96] A. Bouajjani et Habermehl P. Constraint properties, semi-linear systems and Petri nets. In *CONCUR'96*, volume 1119 of *LNCS*, 1996.

[BRI 88] E. Brinksma A theory for the derivation of tests In *PSTV'88*, Elsevier Science Publishers B.V., North Holland, 1988

[BRY 86] R. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, 1986.

[BUC 62] J.R. Buchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, USA, 1962. Stanford University Press.

[CLE 89] R. Cleaveland. Testing equivalence as a bisimulation equivalence. In *Proc. of CAV '89*, volume 407 of *LNCS*, pages 24–37. Springer-Verlag, 1989.

[COU 99] J-M. Couvreur. On-the-fly verification of linear temporal logic. In *Proc. of the Formal Methods'99*, volume 1708 of *LNCS*, pages 253–271. Springer-Verlag, 1999.

[DAM 92] M. Dam. Fixed points of Büchi automata. In *Foundations of Software Technology and Theoretical Computer Science, 12th Conference*, volume 652 of *LNCS*, pages 39–50, New Delhi, India, Décembre 1992.

[DE 87] R. De Nicola Extensional Equivalences for Transitions Systems *Acta Informatica* 24 1987

[DIA 01] M. Diaz Les Réseaux de Petri. Modèles fondamentaux Hermes Science, Traité IC2 Information-Commande-Communication, NřISBN 2-7462-0250-6, 2001, 384p.

[NV 90] R. De Nicola, F. Vaandrager Three Logics for branching bissimulation In *Proc of 5th IEEE Symp. on Logic in Computer Science, 1990*

[DI 86] A. Dicky An algebraic and algorithmic method for analyzing transition s ystems *Theoretical Computer Science*, Vol n^o 46, 1986

[DRI 92] K. Drira Transformation et composition des graphes de refus : analyse de la testabilité Thèse de l'Université P. Sabatier, Toulouse 1992 Rapport LAAS : 92435

[EC 81] Emerson, E. A. and Clarke, E. M. Characterizing Correctness Properties of Parallel Programs as Fixpoints In *Proc. of ICALP'81*, volume 85 of *LNCS*, Springer-Verlag, 1981.

[EC 82] Emerson, E. A. and Clarke, E. M. Using Branching Time Temporal Logic to Synthesize Synchronisation Skeletons Science of Computer Programming, volume 2, pages 241–266, 1982

[EH 85] Emerson, E. A. and Halpern, J. Y. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time Journal of Computer and System Sciences, volume 30 :1, pages 1–24, 1985.

[EME 96] E.A. Emerson. Automated temporal reasonning about reactive systems. In *Logics for Concurrency : Structure versus Automata*, volume 1043 of *LNCS*, pages 41–101. Springer Verlag, 1996.

[ESP 97] J. Esparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34 :85–107, 1997.

[ESP 98] J. Esparza. Decidability and complexity of Petri net problems - an introduction. In *Lectures on Petri Nets I : Basic Models*, volume 1491 of *LNCS*, pages 374–428. Springer Verlag, 1998.

[FER 89] J. C Fernandez An Implementation of an efficient algorithm for bissimulation e quivalence *Science Computer Programming*, 13 :219-236, 1989

[GER 95] R. Gerth, D. Peled, M.Y. Vardi et P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc.* 15^{th} *Workshop on Protocol Specification, Testing and Verification (PSTV'95)*, Varsovie, Pologne, Juin 1995. North-Holland.

[GOD 93] P. Godefroid et G.J. Holzmann. On the verification of temporal properties. In *Proc.* 13^{th} *Workshop on Protocol Specification, Testing and Verification (PSTV'93)*, pages 109–124, Liege, Belgique, Mai 1993.

[HAD 00] S. Haddad et D. Poitrenaud. A model checking decision procedure for sequential recursive Petri nets. Technical Report 2000/024, LIP6 - Université P. et M. Curie, septembre 2000.

[HM 85] M. Hennessy, R. Milner Algebraic Laws for Nondeterminism and Concurrency *Journal of the A.C.M*, Volume 32(1) :137–161, 1985

[HEN 85] M. Hennessy Acceptance trees *Journal of the A.C.M*, Volume 32(4) :896–928, 1985

[HV 01] S.Haddad, F.Vernadat Méthodes d'analyse des réseaux de Petri In Les Réseaux de Petri. Modèles fondamentaux, Hermes Science, Traité IC2 Information-Commande-Communication, ISBN 2-7462-0250-6, 2001, Chapitre 3, pp.69-117

[HAD 01] S.Haddad Décidabilité et complexité de problèmes de réseaux de Petri In Les Réseaux de Petri. Modèles fondamentaux, Hermes Science, Traité IC2 Information-Commande-Communication, ISBN 2-7462-0250-6, 2001, Chapitre 3, pp.69-117

[JAN 95] P. Jančar. Undecidability of bisimilarity for **P**etri nets and some related problems. *Theoretical Computer Science*, 148 :281–301, 1995.

[JAN 99] P. Jančar, J. Esparza et F. Moller. Petri nets and regular processes. *Journal of Computer and System Sciences*, 59(3) :476–503, 1999.

[LED 90] G. Leduc. Comformance relation, associated equivalence and new canonical tester in Lotos In *PSTV'91*, Elsevier Science Publishers B.V., North Holland, 1991

[MAY 84] E.W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13 :441–460, 1984.

[MIL 89] R. Milner *Communication and Concurrency* Prentice Hall, 1989

[OH 86] E.R. Olderog, C.A. Hoare Specification-Oriented Semantics for Communicating Processes *Acta Informatica* 23 :9–66 , 1986,

[PARK 81] D. Park Concurrency and automata on infinite sequences 5th Conf. On theoretical Computer Sciences pages 167-183 volume 104 of *LNCS*, pages 167–183. Springer Verlag, 1981

[PT 87] R. Paige, R.E. Tarjan Three partition refinement algorithms *SIAM J. Comput*, 1987

[RAC 78] C. Rackoff. The covering and boudedness problems for vector addition systems. *Theoretical Computer Science*, 6(2) :223–231, 1978.

[GLA 90] Rob J. van Glabbeek The Linear Time-Branching Time Spectrum. CONCUR 1990 : 278-297

[VAR 96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency : Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266. Springer Verlag, 1996.

[WOL 83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1-2) :72–93, 1983.

[YEN 92] H-C. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96 :119–137, 1992.