Un point de vue symbolique sur la logique temporelle linéaire

Jean-Michel Couvreur
LaBRI, Université de Bordeaux I, Talence, France
couvreur@labri.u-bordeaux.fr

Abstract

In this work, we revise the symbolic Büchi automata construction for linear temporal logic. Our main goal was educational: design a technique easy to understand, manage past operators and construct small automata for small formulas. We submit a technique which answers to this three objectives and which is more efficient than the previous symbolic constructions. We generalize our result to an extended temporal logic. This allows us to handle the complementation problem for Büchi automata in an optimal way.

Résumé

Dans ce travail, nous proposons de réviser les méthodes de construction symbolique d'un automate pour une formule de logique temporelle linéaire. Notre objectif initial était pédagogique : développer une technique facile à comprendre intégrant les opérateurs du passé et construisant de petits automates pour de petites formules. La technique que nous proposons répond à ces trois objectifs et est plus efficace que les techniques symboliques existantes. Nous avons généralisé notre résultat à une logique temporelle linéaire étendue. Celle-ci nous permet de traiter le problème du complément d'un automate de Büchi de manière optimum.

1 Introduction

La logique temporelle est un langage puissant permettant de décrire des propriétés de sûreté, d'équité et de vivacité de systèmes. Il est utilisé comme langage de spécification dans des outils tel que SPIN [4] et SMV [6]. Cependant, vérifier qu'un système respecte une telle spécification est PSPACE-complet [8]. En pratique, les techniques de vérification sont confrontées à un problème d'explosion combinatoire du nombre d'états du système et de celui de l'automate codant la formule. De nombreuses techniques ont été élaborées pour faire face à ce problème d'explosion. Nous pouvons noter les techniques de vérification à la volée combinées avec des techniques de réduction *ordre partiel* (SPIN [4]). La représentation symbolique par les diagrammes de décisions permet de coder un système et l'automate d'une formule de manière concise et ainsi de repousser les limites de la vérification (SMV [6]). Chacune de ces méthodes ont leurs succès sur des systèmes industriels prouvant leur bien-fondé.

Dans ce travail, nous proposons de réviser les méthodes de construction symbolique d'un automate pour une formule de logique temporelle linéaire. Notre objectif initial était pédagogique : développer une technique facile à comprendre intégrant les opérateurs du passé et construisant de petits automates pour de petites formules. La

technique que nous proposons répond à ces trois objectifs et est plus efficace que les techniques symboliques existantes [1, 6]. Nous avons généralisé notre résultat à une logique temporelle linéaire étendue. Celle-ci nous permet de traiter le problème du complément d'un automate de Büchi de manière optimum.

2 Préliminaires

2.1 Logique temporelle linéaire étendue

La logique temporelle linéaire permet d'énoncer des propriétés sur les comportements d'un système. Elle permet de parler de l'évolution d'un système au cours du temps en examinant la suite des états que prend ce système. Les propriétés élémentaires que vérifient ces états jouent un rôle fondamental. Elles introduisent les premières formules d'une logique temporelle : "la propriété p est vérifiée à l'instant t", "la propriété p est vérifiée dans un instant après t" et "la propriété p est vérifiée dans un instant avant t" sont des formules de logique temporelle linéaire. Pour un ensemble AP de propositions élémentaires, les modèles de la logique temporelle sont les mots infinis sur l'ensemble 2^{AP} des valuations de AP (i.e application de AP dans $\{0,1\}$). À partir de ces hypothèses, le temps est discret, a un instant initial, est infini et dans un instant donné seulement les valeurs des propositions élémentaires sont examinées.

Nous considérons une logique temporelle linéaire étendue (LTLE) dans laquelle les termes sont construits à partir de propositions élémentaires et d'automates finis étiquetés par des formules. Le critère de satisfaction d'une formule du type *automate* est basé sur la reconnaissance de mots finis ou infinis par des chemins dans l'automate. Cette logique est similaire aux logiques présentées dans [11, 12]. Une des particularités de notre logique est qu'un automate A permet de définir quatre types de formules : formule finitaire du futur A_{F+}, formule infinitaire du futur A_{L+}, formule finitaire du passé A_{F-}, formule infinitaire du passé A_{L-}. Les formules du futur sont évaluées pour un mot $x_0 \cdot x_1 \cdot x_2 \cdots$ et pour un instant i en reconnaissant le suffixe $x_i \cdot x_{i+1} \cdot x_{i+2} \cdots$ dans l'automate, alors que pour les formules passées, on considère le préfixe fini renversé $x_i \cdot x_{i-1} \cdot x_{i-2} \cdots x_0$. Les formules finitaires et infinitaires se distinguent par leurs critères d'acceptation : une formule finitaire est reconnu par un chemin fini dans l'automate terminant dans un état dit d'acception alors qu'une formule infinitaire peut être reconnue par un chemin fini ou infini. La syntaxe d'une formule LTLE est donnée par:

1. Toute proposition $p \in AP$ est une formule LTLE.
2. Si f et g sont des formules LTLE, alors $\neg f$, $f \wedge g$ sont des formules LTLE.
3. Pour tout automate non déterministe étiqueté par des formules LTLE $A = \langle S, \rightarrow, s_0, F \rangle$ où S est un ensemble d'états, $\rightarrow$ une relation de transitions , s_0 est un état initial, $F \subseteq S$ est un ensemble d'états d'acceptations, A_{F+}, A_{L+}, A_{F-}, A_{L-} sont des formules LTLE.

Nous utilisons la notation standard $\sigma, i \models f$ pour représenter la satisfaction d'une formule LTLE pour un mot infini σ et un instant i. L'évaluation de cette relation pour l'instant 0 définit la satisfaction d'une formule pour un mot : $\sigma \models f \equiv \sigma, 0 \models f$. La relation $\models$ est défini par induction de la manière suivante:

1. $\sigma, i \models p$ ssi $p \in x_i$ pour $p \in AP$;
2. $\sigma, i \models \neg f$ ssi $\neg(\sigma, i \models f)$;
3. $\sigma, i \models f \wedge g$ ssi $\sigma, i \models f$ et $\sigma, i \models g$;

4. $\sigma, i \models A_{F+}$ ssi il existe un chemin fini acceptant $\nu = s_0 \xrightarrow{f_0} s_1 \dots s_{n-1} \xrightarrow{f_{n-1}} s_n$ dans A tel que $\forall k < n : \sigma, i+k \models f_k$ and $s_n \in F$;
5. $\sigma, i \models A_{L+}$ ssi $\sigma \models A_{F+}$ ou il existe un chemin infini acceptant $\nu = s_0 \xrightarrow{f_0} s_1 \dots$ dans A tel que $\forall k : \sigma, i+k \models f_k$,
6. $\sigma, i \models A_{F-}$ ssi il existe un chemin fini acceptant $\nu = s_0 \xrightarrow{f_0} s_1 \dots s_{n-1} \xrightarrow{f_{n-1}} s_n$ dans A tel que $\forall k < n : \sigma, i-k \models f_k$ and $s_n \in F$;
7. $\sigma, i \models A_{L-}$ ssi $\sigma, i \models A_{F-}$ ou il existe un chemin fini acceptant $\nu = s_0 \xrightarrow{f_0} s_1 \dots s_i$ de longueur i dans A tel que $\forall k : \sigma, i-k \models f_k$.

On peut aussi définir de manière équivalente LTLE à partir d'opérateurs temporels de la forme $A_{S\pm}(f_1, \dots, f_n)$ (avec $S = F, L$) où l'automate fini A est étiqueté par des formules propositionnelles sur $AP \cup \{f_1, \dots, f_n\}$. Les opérateurs classiques de logique temporelle linéaire "Next" et "Until", ainsi que leurs versions sur le passé, peuvent être redéfinis en LTLE par :

- $fUg = A_{F+}(f, g)$, $fU^-g = A_{F-}(f, g)$avec $A = \langle \{s_1, s_2\}, \{s_1 \xrightarrow{f} s_1, s_1 \xrightarrow{g} s_2\}, s_1, \{s_2\}\rangle$;
- $O(f) = A_{F+}(f)$, $O^-(f) = A_{F-}(f)$ avec $A = \langle \{s_1, s_2, s_3\}, \{s_1 \xrightarrow{1} s_2, s_2 \xrightarrow{f} s_3\}, s_1, \{s_3\}\rangle$;

2.2 Automates de Büchi à transitions

Le principe des méthodes de vérification automatique est d'associer à toute formule un automate fini acceptant les mêmes mots. L'acceptation d'un mot par ces automates repose sur l'existence d'un chemin infini reconnaissant le mot tel que l'ensemble des états parcourus infiniment souvent vérifie un certain critère. Un critère simple est celui des automates de Büchi : le chemin doit passer infiniment souvent par au moins un état d'acceptation. Dans le cadre de notre étude, nous utilisons les automates de Büchi à transitions. Ils possèdent un ensemble de conditions d'acceptions et chaque condition d'acception définit un ensemble de transitions. Un chemin est acceptant si il passe infiniment souvent par au moins une transition de chaque condition d'acception. Formellement, un automate de Büchi à transition possède les composantes suivantes :

- S est un ensemble fini d'états;
- $\rightarrow \subseteq S \times 2^{AP} \times S$ est une relation de transitions
- Acc est un ensemble de conditions d'acceptation : $\forall a \in Acc, a \subseteq \rightarrow$;
- S_0 est un ensemble d'états initiaux.

Un mot infini $\sigma = x_0.x_1.x_2 \dots$ sur l'alphabet 2^{AP} est accepté par un automate de Büchi à transitions si et seulement si il existe un chemin infini dans A

$$\rho = s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} \dots$$

tel que $\forall a \in Acc, \forall i \geq 0, \exists j \geq i : \langle s_j, x_j, s_{j+1}\rangle \in a$.

Un automate de Büchi à transitions peut être défini symboliquement en identifiant les états par des vecteurs de booléens. Cela offre un moyen simple et concis de les construire et de les définir. Considérons un tableau de variables booléennes Now de la taille des vecteurs de booléens. Un état associe une valeur booléenne à chaque variable booléenne. Un ensemble d'états est alors représenté par une formule booléenne des

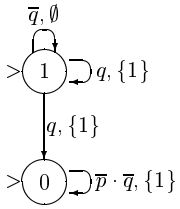

Figure 1: Représentation explicite d'un automate de Büchi à transitions

variables du tableau Now, cette formule rendant vrai pour les états de l'ensemble. De la même manière, une transition est un vecteur de booléens codant l'état source, l'état destination et les valeurs des propositions élémentaires. En utilisant un tableau $Next$ pour identifier l'état destination, une transition associe des valeurs à chaque variable booléenne des tableaux Now et $Next$, ainsi qu'à toutes les propositions élémentaires. Une relation de transition et une condition d'acception peuvent être représentées par des formules booléennes. Nous utilisons les notations suivantes pour définir symboliquement un automate :

- Now et $Next$ sont des tableaux de variables booléennes indicées par un ensemble fini VAR,
- S est une formule booléenne des variables du tableau Now. Elle identifie l'ensemble des états
- R est une formule booléenne des variables de AP, Now et $Next$. Elle représente la relation de transition.
- Acc est un tableau de formules booléennes des variables de AP, Now et $Next$, indicées par un ensemble ACC. Il code les conditions d'acceptation.
- S_0 est une formule booléenne des variables du tableau Now. Elle identifie l'ensemble des états initiaux.

Exemple 1. Considérons l'automate donné par sa représentation symbolique :

$$\begin{aligned} AP &= \{p, q\} \\ VAR &= \{1\} \\ ACC &= \{1\} \\ S &= 1 \\ S_0 &= 1 \\ R &= (\overline{p} + Now[1]) \cdot (Now[1] \Leftrightarrow q + Next[1]) \\ Acc[1] &= \neg Now[1] + q \end{aligned}$$

Nous pouvons obtenir une représentation explicite de l'automate en évaluant dans un premier temps la relation de transitions pour toutes les valeurs des états sources et destinations : $0 \xrightarrow{\overline{p}\cdot\overline{q}} 0$, $1 \xrightarrow{q} 0$ et $1 \xrightarrow{1} 1$. Dans un deuxième temps, on décompose les transitions pour faire apparaître celles qui sont dans l'ensemble $Acc[1]$.

3 Construction d'un automate pour une formule LTL

Le principe de construction d'un automate pour une formule LTL est dû essentiellement à Lichtestein, Pnueli, Vardi et Wolper [5, 10]. Une approche symbolique de ces construc-

$Formule$	VAR	S_0
f	f	$Now[f]$
$O(g)$	g	
gUh	gUh	
$O^-(g)$	$O^-(g)$	$\neg Now[O^-(g)]$
gU^-h	$O^-(gU^-h)$	$\neg Now[O^-(gU^-h)]$

Table 1: Définition du vecteur d'état et des états initiaux

$Formule$	R			Acc
f	$Now[f]$	$\Leftrightarrow$	f	
$O(g)$	$Now[g]$	$\Leftrightarrow$	g	
gUh	$Now[gUh]$	$\Leftrightarrow$	$h + g \cdot Next[gUh]$	$\neg Now[gUh] + h$
$O^-(g)$	$Next[O^-(g)]$	$\Leftrightarrow$	g	
gU^-h	$Next[O^-(gU^-h)]$	$\Leftrightarrow$	$h + g \cdot Now[O^-(gU^-h)]$	

Table 2: Relation de transitions et conditions d'acceptation

tions peut être trouvée dans [1, 6]. Notre méthode ne diffère que très légèrement des constructions précédentes. La différence vient du type d'automate que l'on construit. Ces techniques sont basées sur les deux assertions suivantes:

$$\begin{aligned} gUh &\equiv h + g \cdot O(gUh) \\ gU^-h &\equiv h + g \cdot O^-(gU^-h) \end{aligned}$$

La valeur d'une formule LTL à l'instant 0 pour un mot infini donné dépend uniquement des valeurs de ces sous-formules temporelles $O(g)$, gUh, $O^-(g)$ et gU^-h pour tous les instants du mot ainsi que des valeurs des propositions élémentaires. Leurs valeurs sont régies par les assertions précédentes. Dans notre construction symbolique, chaque sous-formule est représentée par un bit dans le vecteur d'état et les assertions sont considérées comme des contraintes de la relation de transitions. Afin que ces assertions ne fassent référence qu'à l'instant présent et suivant, nous associons aux bits des formules $O(g)$ et gU^-h les formules g et $O^-(gU^-f)$. Un bit spécial est associé à la formule initiale f dont l'objet est d'identifier les états initiaux. Nous ajoutons comme contraintes que les formules du type $O^-(g)$ et $O^-(gU^-h)$ ne peuvent être vérifiée à un instant 0. Nous associons en plus une condition d'acception pour toute sous-formule gUh. En effet une formule gUh est vérifiée si elle respecte son assertion et si la formule h est vérifiée dans un instant futur. Notre construction peut se résumer par les deux tables 1, 2 dans laquelle la formule f représente la formule initiale.

Les formules f, g et h apparaissant dans les contraintes de la relation de transitions et les conditions d'acceptation doivent être considérées comme des formules propositionnelles sur les propositions élémentaires et les variables des tableaux Now et $Next$. Il suffit d'identifier leurs sous-formules du type $O(l)$, l_1Ul_2, $O^-(l)$ et $l_1U^-l_2$ par les variables $Next[l]$, $Now[l_1Ul_2]$, $Now[O^-(l)]$ et $Next[O^-(l_1Ul_2)]$.

Exemple 2. Construction d'un automate pour la formule $f = \overline{p}U(p \cdot (\overline{q}U^-q))$

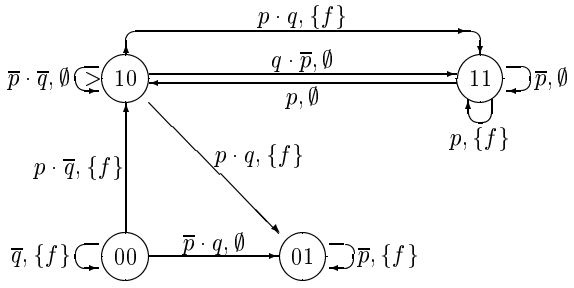

Figure 2: Automate de Büchi à transitions pour $f = \overline{p}U(p \cdot (\overline{q}U^{-}q))$

Cette formule accepte les mots tel que la propriété q est vraie avant la propriété p. Les états de l'automate sont des vecteurs de bits à deux dimensions : $VAR = \{f, g\}$ avec $g = O^{-}(\overline{q}U^{-}q))$. L'ensemble des conditions d'acception est réduit à $\{f\}$. En effet, la formule ne contient qu'un opérateur du type U. En appliquant les règles de construction d'un automate, nous obtenons :

$$\begin{array}{rcl} S & = & 1 \\ S_0 & = & Now[f] \cdot \neg Now[g] \\ R & = & (Now[f] \Leftrightarrow p \cdot Next[g] + \overline{p} \cdot Next[f]) \\ & \cdot & (Next[g] \Leftrightarrow q + \overline{q} \cdot Now[g]) \\ Acc[f] & = & \neg Now[f] + p \cdot Next[g] \end{array}$$

La représentation explicite de cet automate est donnée figure 2. Nous pouvons noter que l'état 00 peut être éliminé de l'automate car il est non accessible à partir de l'état initial. Cependant, notre construction code la formule et la négation de formule : en prenant l'état 00 comme état initial, l'automate code $\neg f$. □

Exemple 3. Construction d'un automate pour la formule $f = \Box(p \Rightarrow \Diamond q)$:

Les opérateurs temporels $\Diamond$ et $\Box$ sont définis par $\Diamond(g) = 1Ug$ et $\Box(g) = \neg\Diamond(\neg g)$. $\Diamond(g)$ exprime que g sera vérifié au moins à un instant et $\Box(g)$ exprime l'invariant "g est vérifié à tout instant". Dans le cas d'une formule invariante $\Box(g)$, nous pouvons simplifier la construction en n'associant pas $\Box(g)$ à un bit du vecteur d'états, en ajoutant g comme contrainte de la relation de transition, et en prenant tous les états pour des états initiaux. Ainsi pour la formule $f = \Box(p \Rightarrow \Diamond q)$, nous retrouvons l'automate de l'exemple 1 (Figure 1). L'unique bit du vecteur d'états est associé à la formule $\Diamond q$. □

La justification de notre construction symbolique est simple. Un chemin infini acceptant dans un automate décrit un mot infini vérifiant à chaque instant les formules définies par le vecteur de bit de l'état correspondant. En particulier, le mot infini doit vérifier la formule initiale f. Cette preuve peut être obtenue par récurrence sur la taille des formules associées aux états. Réciproquement, à un mot infini vérifiant f correspond un chemin infini dans l'automate tel qu'à tout instant le mot vérifie les mêmes formules que l'état correspondant.

La table 3 compare notre construction avec les principales techniques de la littérature en se basant sur un ensemble de formules proposées dans [3]. La première colonne reprend les résultats données dans [3] pour la première construction développé par Vardi et Wolper [10]. Nous donnons comme résultat le nombre d'états accessibles de l'automate obtenu. Nous distinguons ensuite les constructions locales des construc-

Formules	Première	Locale		Symbolique	
	$a \subseteq S$ [10]	$a \subseteq S$ [3]	$a \subseteq \rightarrow$ [2]	$a \subseteq S$ [1]	$a \subseteq \rightarrow$
pUq	8	3	2	8	2
$pU(qUs)$	26	4	3	32	3
$\neg((pU(qUs))$	26	7	3	32	3
$GFp \Rightarrow GFq$	114	9	5	28	7
$FpUGq$	56	8	4	32	3
$GpUq$	13	5	4	16	4
$\neg(FFp \Leftrightarrow Fp)$	-	22	3	1	1

Table 3: Comparaison des techniques de construction d'un automate pour LTL

tions symboliques : ces dernières définissent directement l'automate par des contraintes symboliques alors que les premières calculent les états de proche en proche en cherchant à limiter le nombre d'états. Elles se distinguent aussi par le type des conditions d'acceptation : soit elles sont définies par un ensemble d'états, soit par un ensemble de transitions. Bien que ces formules soient minuscules, nous pouvons conclure qu'il est plus efficace de construire des automates de Büchi à transitions. Nous pouvons aussi noter que notre nouvelle technique concurrence les méthodes de construction locale. Ce dernier point constitue une petite surprise dans le sens où cette technique n'a pas été développée pour minimiser le nombre d'états mais pour améliorer et simplifier les techniques de vérification symbolique proposées dans [1].

4 Construction d'un automate pour une formule LTLE

La première construction d'un automate pour une formule LTLE est due à Vardi et Wolper [11]. Cette construction se limite aux formules du futur. Nous proposons une construction symbolique simple et plus efficace que la précédente. Elle se base sur les assertions induites par les formules du type automate :

$$\begin{array}{rcl} A_{S+}(s) & \equiv & \sum_{s \xrightarrow{g} r : r \notin F} g \cdot O(A_{S+}(r)) + \sum_{s \xrightarrow{g} r : r \in F} g \\ A_{S-}(s) & \equiv & \sum_{s \xrightarrow{g} r : r \notin F} g \cdot O^{-}(A_{S-}(r)) + \sum_{s \xrightarrow{g} r : r \in F} g \end{array}$$

Les formules $A_{S+}(s)$ et $A_{S-}(s)$ représentent des formules finitaires ou infinitaires du futur et du passé dans lequel l'automate A a s comme état initial. La valeur d'une formule LTLE à l'instant 0 pour un mot infini dépend des valeurs des sous-formules $A_{S+}(s)$ et $A_{S-}(s)$ pour tous les instants du mot. Leurs valeurs sont régies par les assertions précédentes. Comme pour la construction symbolique d'une formule LTL, chacune de ces sous-formules correspond à un bit dans le vecteur d'état et les assertions définissent des contraintes de la relation de transitions. Pour les formules du passé, la valeur du bit associée à $A_{S-}(s)$ est celui de la formule $O^{-}(A_{S-}(s))$. Pour traiter les formules du futur, nous introduisons un nouveau bit $A^{\alpha}_{S+}(s)$ pour chaque sous-formule $A_{S+}(s)$. Il est utilisé pour *marquer* les bits valant 1 pour les formules finitaires $A_{F+}(s)$ et 0 pour les formules infinitaires $A_{L+}(s)$. Leurs valeurs ne respectent les assertions que lorsqu'elles correspondent à un bit marqué. Ce procédé permet de suivre l'évolution d'un sous-ensemble de formules $A_{F+}(s)$ (resp. $A_{L+}(s)$) valant 1 (resp. 0) pour chaque formule A_{F+} (resp. A_{L+}). Quand une transition permet de prouver localement la validité de cet ensemble de sous-formules, nous marquons dans l'état suivant toutes

$Formule$	VAR	S	S_0
f	f		$Now[f]$
A_{F+}	$A_{F+}(s), A^{\alpha}_{F+}(s)$	$Now[A^{\alpha}_{F+}(s)] \Rightarrow Now[A_{F+}(s)]$	
A_{L+}	$A_{L+}(s), A^{\alpha}_{L+}(s)$	$\neg Now[A^{\alpha}_{L+}(s)] \Rightarrow \neg Now[A_{L+}(s)]$	
A_{F-}	$O^-(A_{F-}(s))$		$\neg Now[O^-(A_{F-}(s))]$
A_{L-}	$O^-(A_{L-}(s))$		$\neg Now[O^-(A_{L-}(s))]$

Table 4: Définition du vecteur d'état, des états de l'automate et des états initiaux

$Formule$	Acc
A_{F+}	$\prod_{s \notin F} (\neg Now[A^{\alpha}_{F+}(s)] + \sum_{(g,r):r \in F \wedge s \xrightarrow{g} r} g)$
A_{L+}	$\prod_{s \notin F} (Now[A^{\alpha}_{L+}(s)] + \neg \sum_{(g,r):r \in S \wedge s \xrightarrow{g} r} g)$

Table 5: Définition des conditions d'acceptation

les formules $A_{S+}(s)$. Ces transitions définissent les conditions d'acceptations de notre automate ; il en existe une pour chaque sous-formule A_{S+}. Notre construction peut se résumer par les tables 4, 5 et 6.

La justification de notre construction est plus difficile que pour celle de LTL. Dans un sens, il est simple de prouver qu'un chemin infini acceptant dans un automate décrit un mot infini vérifiant à chaque instant les formules définies par le vecteur de bits de l'état correspondant. Réciproquement, à un mot infini vérifiant f correspond plusieurs chemins infinis dans l'automate tel qu'à tout instant le mot vérifie les mêmes formules que l'état correspondant. Nous pouvons cependant construire un chemin acceptant de proche en proche en effectuant le lien entre les formules marquées et des chemins dans les automates des formules A_{S+}.

Exemple 4. Construction d'un automate pour la formule $\Box(B_{F+})$
avec $B = \langle \{s_1, s_2, s_3\}, \{s_1 \xrightarrow{p} s_2, s_2 \xrightarrow{p} s_1, s_2 \xrightarrow{q} s_3\}, s_1, \{s_3\} \rangle$;

$Formule$	R		
f	$Now[f]$	$\Leftrightarrow$	f
A_{F+}	$Now[A_{F+}(s)]$	$\Leftrightarrow$	$\sum_{s \xrightarrow{g} r:r \notin F} g \cdot Next[A_{F+}(r)] + \sum_{s \xrightarrow{g} r:r \in F} g$
	$Now[A^{\alpha}_{F+}(s)]$	$\Rightarrow$	$\sum_{s \xrightarrow{g} r:r \notin F} g \cdot Next[A_{F+}(r)] + \sum_{s \xrightarrow{g} r:r \in F} g$
	$Acc(A_{F+})$	$\Rightarrow$	$\prod_{s \in S} Next[A^{\alpha}_{F+}(s)] \Leftrightarrow Next[A_{F+}(s)]$
A_{L+}	$Now[A_{L+}(s)]$	$\Leftrightarrow$	$\sum_{s \xrightarrow{g} r:r \notin F} g \cdot Next[A_{L+}(r)] + \sum_{s \xrightarrow{g} r:r \in F} g$
	$Now[A^{\alpha}_{L+}(s)]$	$\Leftarrow$	$\sum_{s \xrightarrow{g} r:r \notin F} g \cdot Next[A_{L+}(r)] + \sum_{s \xrightarrow{g} r:r \in F} g$
	$Acc(A_{L+})$	$\Rightarrow$	$\prod_{s \in S} Next[A^{\alpha}_{L+}(s)] \Leftrightarrow Next[A_{L+}(s)]$
A_{F-}	$Next[O^-(A_{F-}(s))]$	$\Leftrightarrow$	$\sum_{s \xrightarrow{g} r:r \notin F} g \cdot Now[O^-(A_{F-}(r))] + \sum_{s \xrightarrow{g} r:r \in F} g$
A_{L-}	$Next[O^-(A_{L-}(s))]$	$\Leftrightarrow$	$\sum_{s \xrightarrow{g} r:r \notin F} g \cdot Now[O^-(A_{L-}(r))] + \sum_{s \xrightarrow{g} r:r \in F} g$

Table 6: Définition de la relation de transitions

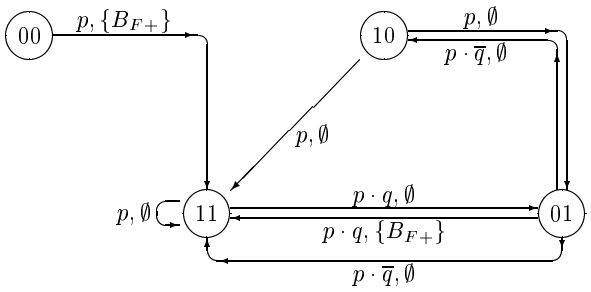

Figure 3: Automate de Büchi à transitions pour $\Box(B_{F+})$

Comme notre formule est du type invariant, nous simplifions notre construction en ajoutant $Now[B_{F+}(s_1)]$ comme contrainte à la relation de transitions et en prenant tous les états comme états initiaux.

Les états de l'automate sont des vecteurs de bits à quatre dimensions : $VAR = \{B_{F+}(s_1), B^{\alpha}_{F+}(s_1), B_{F+}(s_2), B^{\alpha}_{F+}(s_2)\}$. L'ensemble des conditions d'acceptation est réduit à $\{B_{F+}\}$. En appliquant nos règles de construction, nous obtenons :

$$
\begin{array}{rcl}
S & = & (Now[B^{\alpha}_{F+}(s_1)] \Rightarrow Now[B_{F+}(s_1)]).(Now[B^{\alpha}_{F+}(s_2)] \Rightarrow Now[B_{F+}(s_2)]) \\
S_0 & = & 1 \\
Acc(B_{F+}) & = & \neg Now[B_{F+}(s_1)] \cdot (\neg Now[B_{F+}(s_2)] + q) \\
R & = & Now[B_{F+}(s_1)] \\
 & \cdot & (Now[B_{F+}(s_1)] \Leftrightarrow p \cdot Next[B_{F+}(s_2)]) \\
 & \cdot & (Now[B^{\alpha}_{F+}(s_1)] \Rightarrow p \cdot Next[B^{\alpha}_{F+}(s_2)]) \\
 & \cdot & (Now[B_{F+}(s_2)] \Leftrightarrow p \cdot Next[B_{F+}(s_1)] + q) \\
 & \cdot & (Now[B^{\alpha}_{F+}(s_2)] \Rightarrow p \cdot Next[B^{\alpha}_{F+}(s_1)] + q) \\
 & \cdot & (Acc(B_{F+}) \Rightarrow (Now[B_{F+}(s_1)] \Leftrightarrow Now[B^{\alpha}_{F+}(s_1)]) \cdot \\
 & & (Now[B_{F+}(s_2)] \Leftrightarrow Now[B^{\alpha}_{F+}(s_2)]))
\end{array}
$$

La représentation explicite de cet automate est donnée figure 3. Nous n'avons représenté dans les états que les valeurs des bits associées aux indices $B^{\alpha}_{F+}(s_1)$ et $B^{\alpha}_{F+}(s_2)$. En effet les deux autres bits valent toujours 1. □

Une application directe de notre technique est la construction de la négation d'un automate de Büchi. Récemment C. Löding [9] a développé une méthode de transformation d'un automate de Büchi vers un autre type d'automate : les automates faibles alternants. Cette transformation s'adapte parfaitement à notre problème. En effet un automate de Büchi à n états $A = \langle S, \rightarrow, F, S_0 \rangle$ s'écrit A_{2nL+} en LTLE à partir des $2n+1$ formules A_{0L+}, A_{1F+}, A_{2L+}, $A_{3F+} \ldots A_{2nL+}$. Ces automates se déduisent de l'automate A de la manière suivante :

- Pour l'indice 0, l'automate est simplement défini par $A_0 = \langle S, \rightarrow, \emptyset, S_0 \rangle$,
- Pour les indices pairs autre que 0, l'automate est donné par $A_{2*i} = \langle S, \rightarrow_{2*i}, \emptyset, S_0 \rangle$ où chaque transition $\langle s, x, r \rangle$ de A produit une transition $\langle s, x \cdot A_{2i-1\,L+}, r \rangle$ dans A_{2*i},
- Pour les indices impairs, l'automate est donné $A_{2*i-1} = \langle S \cup \{0\}, \rightarrow_{2*i-1}, \{0\}, S_0 \rangle$ où chaque transition $\langle s, x, r \rangle$ de A dont la destination est acceptante ($r \in F$) produit une transition $\langle s, x \cdot A_{2i-2\,F+}, 0 \rangle$ dans A_{2*i-1}, les autres donnent la même transition $\langle s, x, r \rangle$ dans A_{2*i-1}.

Notre construction déduit directement pour la formule $\neg A_{2nL+}$ un automate de Büchi à transitions avec au maximum $3^{n\cdot(2n+1)}$ états. Nous pouvons réduire le nombre d'états en $2^{O(n \log n)}$ en remarquant que $A_{0L+}(s) \Leftarrow A_{1F+}(s) \Leftarrow A_{2L+}(s) \Leftarrow A_{3F+}(s) \Leftarrow \ldots \Leftarrow A_{2nL+}(s)$ et en imposant que des formules d'automates différents ne peuvent pas être marquées dans un même état. La transformation d'un automate à transitions en un automate de Büchi laisse le nombre d'états en $2^{O(n \log n)}$. Ce coût est essentiellement égal à la borne inférieure du nombre d'états établie par M. Michel et a déjà été atteint par S. Safra [7] par une technique de déterminisation d'automates. Nous obtenons par une méthode symbolique une construction essentiellement optimale de la négation d'un automate de Büchi.

References

[1] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. *Lecture Notes in Computer Science*, 803, 1994.

[2] J.-M. Couvreur. On-the-fly verification of linear temporal logic. In *FM'99*, volume 1708 of *Lecture Notes in Computer Science*, pages 253–271, 1999.

[3] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV95, Protocole Specification Testing and Verification*, pages 3–18, 1995.

[4] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[5] O. Lichtenstein and A. Pnueli. Checking the finite-state concurrent programs satisfy their linear specifications. In *popl85*, pages 97–107, 1985.

[6] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publisher, Boston, MA, 1994.

[7] S. Safra. On the complexity of ω-automata. In *29th Ann. Symp. on Foundations of Computer Science*, IEEE Computer Society, pages 319–327, 1988.

[8] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *Journal of the Association for Computing Machinery*, 32(3):733–749, July 1985.

[9] W. Thomas. Complementation of Büchi automata revisited. *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, Springer-Verlag*, pages 109–122, 1999.

[10] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *1th Symposium on Logic in Computer Science*, pages 322–331, 1986.

[11] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[12] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.