Automates et logiques temporelles
Vérification des systèmes réactifs

Hubert Comon

21 mars 2002

Ce sont des notes préliminaires. Les commentaires des lecteurs sont bienvenus.

Table des matières

1 Introduction

Une logique devient temporelle quand on ajoute à chacune de ses fonctions (ou chacun de ses prédicats) un paramètre supplémentaire : le temps. Nous nous concentrerons ici sur les logiques temporelles propositionnelles ; celles-ci se plongent naturellement dans des logiques du premier ordre (ou du second ordre) monadiques (c'est à dire chaque symbole ayant pour seul argument le temps).

Les logiques monadiques de second ordre ont été bien étudiées dans les années 60 : à la suite des travaux, entre autres, de Büchi, Elgott et Rabin, les modèles de ces formules sont reconnus par des automates appropriés. C'est ainsi que les automates entrent dans le paysage et fournissent des techniques de décision pour divers problèmes concernant les logiques temporelles tels la satisfaisabilité et la vérification (model-checking).

Nous montrons donc, après les avoir introduites, diverses logiques temporelles et leurs traduction en logique du premier (resp. second) ordre ainsi que le lien entre les logiques monadiques de second ordre et les automates.

Malheureusement, les techniques générales de Büchi ou Rabin donnent des algorithmes de complexité énorme, qui sont bien moins bons que les algorithmes spécialisés dans les logiques temporelles et qui ont été proposés dans les années 80. L'objectif du cours est de montrer comment rétablir la situation en expliquant des techniques optimales pour le model-checking et la satisfaisabilité des logiques temporelles. Ces techniques se fondent sur deux outils (mathématiques) : les *automates alternants* et les *jeux*; pour chaque logique temporelle, l'ensemble des modèles est un langage reconnu par un automate alternant approprié, et les propriétés de ces automates s'expriment à l'aide de jeux à deux joueurs sur un graphe fini. La nature de ce graphe et les conditions de victoire déterminent la complexité du calcul d'une stratégie gagnante, ou, de manière équivalente, la satisfaction de la formule.

Un plan un peu plus détaillé est donné en fin de section.

1.1 Les systèmes réactifs

Un exemple est donné dans la figure 1 ; il s'agit d'un modèle ultra simplifié d'une machine à café. Cet automatisme est supposé communiquer avec un environnement par le biais de signaux émis ou reçus. Les étiquettes de la forme ?l sont des actions de réception de signaux, alors que les actions !l sont des émissions. Les envois/réceptions de tels messages par l'automatisme provoquent des changements d'état. On peut imaginer, par exemple, une synchronisation basée sur le rendez-vous (les changements d'états ne peuvent s'effectuer que si l'émission et la réception simultanée d'un même message est possible).

On s'intéresse à la question de savoir si de tels automates satisfont des propriétés comme :

> Chaque fois que j'ai mis deux francs, alors, plus tard, j'obtiendrai un café.

formule qui reste à préciser, bien sûr.

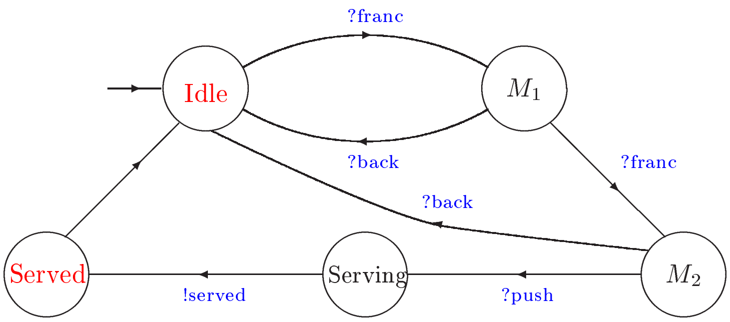

FIG. 1 – Une machine à café très simple

Si l'on compare ce problème à la vérfication de programmes, on peut noter que :
- On s'intéresse essentiellement aux comportements infinis (et donc pas aux fonctions dont le calcul se termine !)
- On s'intéresse aux états successifs et pas au "résultat" d'un calcul

1.2 Les logiques temporelles

Logique temporelle = logique + temps.

Ce qui correspond à l'ajout d'un paramètre "temps" aux symboles de fonction et de prédicat.

Ici, on s'interessera uniquement aux logiques temporelles propositionnelles.

Exemple 1 Reprenons l'exemple précédent (figure 1)

$$\text{Money} \quad \Rightarrow \qquad \text{Served}$$

est une formule du calcul propositionnel sur les variables propositionnelles Money et Served.

$$\forall t.\text{Money}(t) \Rightarrow \exists t' > t.\text{Served}(t')$$

est un énoncé de logique temporelle propositionnelle.

Exemple 2

$$\exists P. \quad (\qquad P \quad \Rightarrow \text{Idle} \quad) \wedge (\qquad \text{Idle} \quad \Rightarrow P \quad)$$

est une formule propositionnelle de second ordre et

$$\exists P.\forall t.(\exists t' > t.P(t) \Rightarrow \text{Idle}(t')) \wedge (\exists t' > t.\text{Idle}(t) \Rightarrow P(t'))$$

est une formule temporelle de second ordre.

1.3 Le temps

Linéaire

vs. Arborescent $\left\{\begin{array}{l}\text{branchement fini}\\ \text{branchement infini}\end{array}\right.$

Discret

vs. Réel

Infini

vs. Transfini

Avec origine du temps

vs. Sans origine du temps

1111 $\mathbb{N}$

2111 $\{0,1\}^*$ avec l'ordre préfixe

1211 $\mathbb{R}_+$

1121 ϵ_0

1112 $\mathbb{Z}$

3221 $\{[0,t] \to \mathbb{R}_+;\ |\ t \in \mathbb{R}_+\}$ (ordre= $t < t'$ et f coïncide avec f' sur $[0,t]$).

Exercice 1 On considère la formule du premier ordre suivante :

$$\begin{array}{ll} & \exists t.P(t) \\ \wedge & \exists t.\neg P(t) \qquad\qquad\qquad\qquad\qquad ds \\ \wedge & \forall t.P(t) \Rightarrow ((\exists t' > t.P(t')) \wedge \forall t' < t.P(t')) \end{array}$$

Dire, suivant les modèles du temps, si cette formule est satisfaisable ou non (et justifier sa réponse soit par une preuve d'insatisfaisabilité, soit en exhibant un modèle).

1.4 Les problèmes

	Donnée	Question
Vérification (modèles "plats")	M, ϕ	$M \models \phi$?
Vérification (modèles "concurrents")	$M_1, \ldots, M_n, \phi$	$M_1 \times \ldots \times M_n \models \phi$?
Satisfaisabilité	ϕ	existe-t-il M tel que $M \models \phi$?
Synthèse	ϕ	Donner M tel que $M \models \phi$

$\times$ étant un produit synchrone qui reste à préciser, comme beaucoup d'autres choses d'ailleurs.

Les formules ϕ considérées seront des formules de logique temporelle propositionnelle (ou de second ordre propositionel) avec divers modèles du temps. Nous nous limiterons dans le cours aux modèles suivants du temps : $\mathbb{N}$, $\{1,\ldots,n\}^*$, $\mathbb{N}^*$. On parlera alors de *temps arborescent* dans le premier cas (sans autre précision) et de *temps arborescent* dans les deux derniers cas (soit à branchement borné, soit à branchement non borné).

Par ailleurs, nous considérerons les logiques suivantes (qui seront définies dans le cours) et qui sont les plus couramment utilisées :

Temps linéaire : LTL, S1S

Temps arborescent : CTL, CTL*, μ-calcul, SkS

On se préoccupera d'*efficacité*.

1.5 Complexité de la vérification

1.5.1 Rappels des classes de complexité

Type de machine :

D = déterministe

N = non-déterministe

A = alternante

Quantité mesurée :

TIME : le temps de calcul

SPACE : l'espace de calcul

La classe de fonctions :

P Polynômial

EXP Exponentiel

2-EXP Doublement exponentiel

Rappelons quelques résultats (cf. [Joh90] pour une synthèse) :

Si $f(n) \geq n$, ATIME$f(n)) \subseteq$ DSPACE$(f(n)) \subseteq \bigcup_{c>0}$ ATIME$(c \times f(n)^2)$.

En particulier, APTIME = DPSPACE.

NSPACE$(f(n)) \subseteq$ DSPACE$(f(n)^2)$

En particulier NPSPACE =DP SPACEsi bien qu'on omet Nou D.

Si $f(n) \geq \log n$, alors ASPACE$(f(n)) = \bigcup_{c>0}$ DTIME$(2^{c \times f(n)})$

En particulier APSPACE= DEXPTIME.

1.5.2 Résumé des résultats connus

Les résultats ci-dessous ont été obtenus dans les années 80 (par ex. [SC85, QS82, CES86, **?**, VW94, Cle93, EJ88]). Nous nous proposons ici de les établir tous d'une manière plus uniforme.

Il faudrait aussi ajouter d'autres résultats pour d'autres modèles du temps. (ex. temps réel).

	Complexité du model-checking	Complexité de la satisfaisabilité	Complexité de la vérification de systèmes concurrents
LTL	PSPACE-complet	PSPACE-complet	PSPACE-complet
CTL	PTIME-complet	DEXPTIME-complete	PSPACE-complet
CTL*	PSPACE-complet	D2-EXPTIME-complet	PSPACE-complet
Alternation-free μ-calculus	Polynomial	DEXPTIME-complet	DEXPTIME-complet
μ-calculus	NPTIME $\cap$ co-NPTIME	DEXPTIME-complet	DEXPTIME-complet

Ces complexités sont toutes optimales, sauf le model-checking pour le μ-calcul : on ne sait pas actuellement s'il existe un algorithme polynômial.

1.6 Plan du cours

Le cours commencera par introduire les concepts de base dans le cas du temps linéaire : logique du temps linéaire (LTL), logique monadique de second ordre à un successeur (S1S), automates de Büchi et correspondance entre S1S et automates de Büchi, en laissant de côté le théorème de complémentation. Ensuite, nous introduirons la notion clef d'automate alternant. En découlent des algorithmes simples et optimaux pour les problèmes de décision de LTL. Nous reviendrons alors sur le théorème de complémentation : nous introduirons les jeux à parité en montrant le lien avec les automates alternants. La complémentation des automates alternants à parité découle d'un théorème simple de détermination des jeux.

Nous nous consacrerons ensuite au temps arborescent, en commençant cette fois par la logique monadique de second ordre à k successeur et les automates d'arbres, sans toutefois prouver le théorème de complémentation (voir [Tho97] pour une preuve accessible basée sur un théorème de détermination des jeux). Puis nous introduisons les logiques CTL et CTL*, les automates alternants d'arbres, et leur lien avec les logiques temporelles. Nous en déduisons des algorithmes optimaux pour les divers prblèmes de décision. Enfin, nous terminons par le μ-calcul, reprenant les techniques déjà mises en place, en particulier les jeux à parité.

Les sources bibliographiques sont essentiellement :

- Thomas (1997) pour les logiques monadiques de second ordre [Tho97]
- Vardi (1994) pour la logique du temps linéaire [Var96]
- Kupferman, Vardi, Wolper (2000) pour les logiques du temps arborescent [KVW00]
- Thomas (1997–1999) [Tho99, Tho97] Pour les liens logique-automates alternants-jeux.

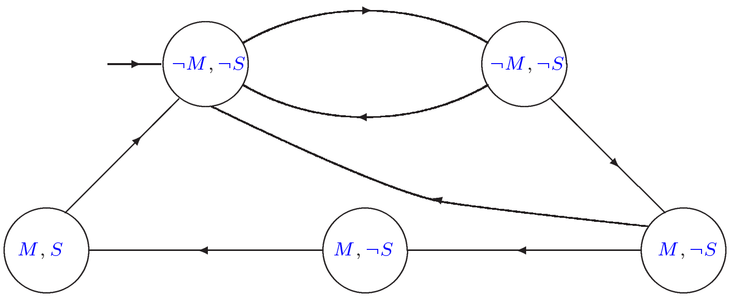

Fig. 2 – Une structure de Kripke finie

2 Structures de Kripke

2.1 Définition

$\mathcal{K} = (S, R, s^0, f, \mathcal{P})$ où $R \subseteq S \times S$, $s^0 \in S$ et $f : S \to \mathcal{P}$. On suppose que, pour tout $s \in S$, il existe au moins un $s' \in S$ tel que $R(s, s')$.

Exemple 3 Si l'on ne tient pas compte des actions et que l'on étiquette chacun des états de l'automate de la figure 1 par un sous-ensemble de $\{M, S\}$, on obtient l'automate et le marquages comme indiqués sur la figure 2 ; M est supposé indiquer que l'on a déjà mis deux francs dans la machine et S que le café est servi.

Exemple 4 Si l'on complique un peu le modèle de la machine à café en acceptant de pouvoir mettre un nombre arbitraire de pièces, on obtient le modèle suivant : $S = \{s^i \mid i \in \mathbb{N}\} \cup \{t^i \mid i \geq 2\} \cup \{u^i \mid i \geq 2\}$,

$$\begin{array}{lcll} s^i & \to & s^{i+1} & \text{pour } i \in \mathbb{N} \\ s^i & \to & s^0 & \text{pour } i \in \mathbb{N} \\ s^i & \to & t^i & \text{pour } i \geq 2 \\ t^i & \to & u^i & \text{pour } i \geq 2 \\ u^i & \to & s^{i-2} & \text{pour } i \geq 2 \end{array}$$

et $f(t^i) = \{M\}$, $f(u^i) = \{S, M\}$ et $f(s^i) = \emptyset$.

2.2 Plusieurs modalités

Si l'on veut tenir compte des émissions/réceptions de messages, on peut considérer le modèle plus général (plusieurs modalités) : on considère un nombre fini d'actions $A = \{a_1, \ldots, a_n\}$ et une relation binaire par action. Par exemple, si on reprend l'automate de la figure 1, on peut considérer les relations binaires $R_{?franc}, R_{?back}, \ldots$.

Elargir le modèle à plusieurs modalités ne présente pas beaucoup d'intérêt car on peut "traduire" le cas multimodal en une seule modalité en remplaçant S par $S \times A$ (soit en devinant à chaque étape l'action qui va suivre, soit en se rappelant de l'action qui précède).

2.3 Executions et dépliage

Une *exécution* de la structure de Kripke est une séquence infinie $w \in S^\omega$ telle que pour tout $i \in \mathbb{N}$, $w(i)Rw(i+1)$ et $w(0) = s^0$

Un sous-ensemble D de $\mathbb{N}_+^*$ est un *domaine d'arbre* s' il est

– clos par préfixe

– clos lexicographiquement : $p \cdot i \in D$ entraine $\forall j, 1 \leq j \leq i, p \cdot j \in D$.

Un *arbre* est une application T d'un domaine d'arbre D dans un ensemble d'étiquettes A.

Si $\mathcal{K}$ est une structure de Kripke de branchement fini (i.e. pour tout $s \in S$, au plus un nombre fini d'éléments de S sont en relation avec s), l'ensemble des branches issues de chaque sommet est totalement ordonné de façon arbitraire. Le *dépliage* de la structure de Kripke est un arbre (éventuellement infini) étiqueté par S défini comme suit : $\epsilon \in D$ et $T(\epsilon) = s^0$. Si $p \in D$ et $T(p) = s$ et $\{s_1, \ldots, s_n\}$ sont les éléments de S tels que sRs_i pour tout i, alors $p \cdot 1, \ldots p \cdot n \in D$ et, pour tout i, $T(p \cdot i) = s_i$.

Le dépliage à partir de $s \in S$ s'obtient de la même façon en remplaçant s^0 par s.

3 Logique du temps linéaire (LTL)

3.1 Syntaxe

Etant donné un ensemble de variables propositionnelles $\mathcal{P}$, l'ensemble des formules de LTL est le plus petit ensemble tel que, si ϕ, ψ sont des formules, alors

true, **false**

P $(P \in \mathcal{P})$

$\mathbf{X}\phi$

$\phi \wedge \psi$, $\phi \vee \psi$

$\neg\phi$

$\phi \, \mathcal{U} \psi$

sont des formules.

On utilise couramment les abréviations suivantes :

$\Diamond\phi \stackrel{\text{def}}{=} \mathbf{true} \, \mathcal{U} \phi$

$\Box\phi \stackrel{\text{def}}{=} \neg(\mathbf{true} \, \mathcal{U} \neg\phi)$

3.2 Sémantique

Soit $\mathcal{K}$ une structure de Kripke. Par définition, $\mathcal{K}$ satisfait ϕ si, pour toute exécution w de $\mathcal{K}$, $(w, 0) \models \phi$, où $\models$ est définie par :

$w, t \models \mathbf{true}$ et $w, t \not\models \mathbf{false}$

$w, t \models P$ ssi $P \in f(w(t))$

$w, t \models \mathbf{X}\phi$ ssi $w, t+1 \models \phi$

$w, t \models \phi \wedge \psi$ ssi $w, t \models \phi$ et $w, t \models \psi$

$w, t \models \phi \vee \psi$ ssi $w, t \models \phi$ ou $w, t \models \psi$

$w, t \models \neg\phi$ ssi $w, t \not\models \phi$.

$w, t \models \phi \, \mathcal{U} \psi$ ssi
- il existe $t' \geq t$ tel que $w, t' \models \psi$
- et pour tout $t'' \in [t, t'[$ on a $w, t'' \models \phi$.

Exercice 2 On note $\phi \equiv \psi$ lorsque ϕ et ψ ont les mêmes modèles. Démontrer ou invalider les équivalences suivantes :

1. $\mathbf{X}\neg\phi \equiv \neg\mathbf{X}\phi$
2. $\mathbf{X}\Diamond\phi \equiv \Diamond\mathbf{X}\phi$
3. $\Box\mathbf{X}\phi \equiv \mathbf{X}\Box\phi$
4. $\mathbf{X}\phi \wedge \mathbf{X}\psi \equiv \mathbf{X}(\phi \wedge \psi)$
5. $\mathbf{X}\phi \vee \mathbf{X}\psi \equiv \mathbf{X}(\phi \vee \psi)$

6. $(\mathbf{X}\phi)\,\mathcal{U}\psi \equiv \psi \vee \mathbf{X}(\phi\,\mathcal{U}(\phi \wedge \psi))$
7. $\neg\Diamond\phi \equiv \Box\neg\phi$
8. $\neg\Box\phi \equiv \Diamond\neg\phi$

Exercice 3 Montrer que $\mathcal{U}$ n'est pas associatif :

$$(\phi_1\,\mathcal{U}\phi_2)\,\mathcal{U}\phi_3 \not\equiv \phi_1\,\mathcal{U}(\phi_2\,\mathcal{U}\phi_3)$$

Il suffira d' exhiber des formules ϕ_1, ϕ_2 et ϕ_3 bien choisies.

3.3 Langage défini par une formule de LTL

Un *mot infini* sur l'alphabet A est une application de $\mathbb{N}$ dans A. L'ensemble des mots infinis construits sur A est noté A^ω. Si $w \in A^\omega$, $w(i)$ pour $i \in \mathbb{N}$ est la $i$ème lettre de w.

L'ensemble des variables propositionnelles étant ordonné de manière arbitraire, une affectation de ces variables est identifiée à un mot de $\{0,1\}^n$. La notion de satisfaction d'une formule ϕ de LTL pour un mot $w \in (\{0,1\}^k)^\omega$ est définie comme dans la section 3.2, la fonction f étant définie par $f(\alpha) = \{P_i \mid \alpha(i) = 1\}$.

Le *langage* L_ϕ défini par une formule LTL ϕ est l'ensemble des mots $w \in (\{0,1\}^k)^\omega$ qui satisfont ϕ.

Exemple 5 Supposant qu'il y a une seule proposition P, si ϕ est la formule $\Box\Diamond P$, $L_\phi = (0^*1)^\omega$, c'est à dire l'ensemble des mots contenant une infinité de 1.

De même, si $\mathcal{K}$ est une structure de Kripke, on note $L_\mathcal{K}$ le langage qu'elle définit sur l'alphabet $\{0,1\}^k$: $L_\mathcal{K}$ est l'ensemble des images par f des exécutions de $\mathcal{K}$.

On peut alors reformuler les problèmes de satisfaisabilité et de vérification de modèles dans ce contexte :

$$\boxed{\phi \text{ est satisfaisable} \quad \Leftrightarrow \quad L_\phi \neq \emptyset}$$

$$\boxed{\mathcal{K} \models \phi \quad \Leftrightarrow \quad L_\mathcal{K} \subseteq L_\phi}$$

3.4 logique monadique du premier ordre à un successeur

On considère la logique du premier ordre *monadique*, à un successeur construite sur un ensemble fini de prédicats unaires $\mathcal{Q}$: les termes sont construits à partir de $0, S$ et des variables. Les formules atomiques sont de la forme $t \geq t'$ ou bien $t = t'$ ou bien $P(t)$, $P \in \mathcal{Q}$, t, t' étant des termes.

On interprète $0, S, \geq, =$ dans les entiers naturels : S est le successeur. Si $\mathcal{Q} = \{P_1, \ldots, P_k\}$, une interprétation est représentée par k ensembles d'entiers, ou, si l'on superpose leurs fonctions caractéristiques, par des mots de $(\{0,1\}^k)^\omega$.

Par exemple, si $\mathcal{Q} = \{P_1, P_2\}$, et $\phi \stackrel{\text{def}}{=} \forall t.P_1(0) \wedge P_1(t) \Rightarrow P_2(S(t)) \wedge P_2(t) \Rightarrow P_1(S(t))$, alors $w \stackrel{\text{def}}{=} \begin{smallmatrix}1\\0\end{smallmatrix}\,\begin{smallmatrix}0\\1\end{smallmatrix}\,\begin{smallmatrix}1\\0\end{smallmatrix}\,\begin{smallmatrix}0\\1\end{smallmatrix}\ \ldots = (\begin{smallmatrix}1\\0\end{smallmatrix}\,\begin{smallmatrix}0\\1\end{smallmatrix})^\omega$ satisfait ϕ.

Exercice 4 Donner d'autres modèles de la formule ϕ ci-dessus.

3.5 Traduction de LTL en logique du premier ordre

Si ϕ est une formule de LTL construite sur les variables propositionnelles $\mathcal{P}$, on construit une formule $\tilde{\phi}(t)$ de la logique monadique de premier ordre sur les prédicats unaires $\{Q_P \mid P \in \mathcal{P}\}$, avec une variable libre t de telle manière que, pour toute structure de Kripke $\mathcal{K}$ (sur l'ensemble de variables propositionnelles $\mathcal{P}$) :

$$\mathcal{K} \models \phi \quad \Leftrightarrow \quad L_{\mathcal{K}} = \{w \mid w \models \tilde{\phi}(0)\}$$

$\widetilde{\mathbf{true}} \stackrel{\text{def}}{=} \mathbf{true}$

$\widetilde{\mathbf{false}} \stackrel{\text{def}}{=} \mathbf{false}$

$\widetilde{P}(t) \stackrel{\text{def}}{=} Q_P(t).$

$\widetilde{\phi \wedge \psi}(t) \stackrel{\text{def}}{=} \widetilde{\phi}(t) \wedge \widetilde{\psi}(t)$

$\widetilde{\phi \vee \psi}(t) \stackrel{\text{def}}{=} \widetilde{\phi}(t) \vee \widetilde{\psi}(t)$

$\widetilde{\neg \phi}(t) \stackrel{\text{def}}{=} \neg \widetilde{\phi}(t)$

$\widetilde{\mathbf{X}\phi}(t) \stackrel{\text{def}}{=} \widetilde{\phi}(t+1)$

$\widetilde{\phi\, \mathcal{U} \psi}(t) \stackrel{\text{def}}{=} \exists t'.t' \geq t \wedge \widetilde{\psi}(t') \wedge \forall t''.(t \leq t'' < t' \Rightarrow \widetilde{\phi}(t''))$

Theorème 6 (Kamp) *L est définissable en logique monadique du premier ordre si et seulement si L est définissable en LTL.*

Plus précisément : on peut traduire les formules de LTL en logique monadique du premier ordre (c'est ce qui est fait dans le paragraphe précédent) et réciproquement (on l'admettra), il est possible de traduire toute formule du premier ordre monadique en une formule de LTL qui définit le même langage. Cette dernière traduction est, en fait, non élémentaire.

4 La logique monadique de second ordre à un successeur : S1S

S1S généralise la logique monadique de premier ordre du paragraphe précédent en autorisant les quantification de second ordre. Vu sous un autre angle, S1S généralise le calcul propositionnel de second ordre en ajoutant une dimension temporelle.

4.1 Syntaxe et sémantique

Syntaxe L'ensemble des formules est le plus petit ensemble tel que, si ϕ, ψ sont des formules, alors les constructions suivantes sont aussi des formules :

- **true**, **false**
- $t \in X$
- $t = 0, t = t' + 1$, $t = t'$, $t \geq t'$
- $\phi \vee \psi$, $\phi \wedge \psi$, $\neg \phi$
- $\forall t.\phi$, $\exists t.\phi$
- $\exists^2 X.\phi$, $\forall^2 X.\phi$

Sémantique Domaine d'interprétation des variables du premier ordre : $\mathbb{N}$ avec les opérations successeur, zéro, l'inégalité.

Domaine d'interprétation des variables de second ordre : sous-ensembles de $\mathbb{N}$.

Exercice 5 Définir en S1S les prédicats suivants :

- Infini(X) qui est satisfait par les ensembles infinis et seulement ceux-là.
- Pair(X) qui est satisfait par une seul ensemble : les entiers pairs.
- $x \geq y$, sans utiliser $\geq$ bien sûr !
- Partition$(X_1, \ldots, X_k)$ qui est satisfait par les tuples d'ensembles formant une partition de $\mathbb{N}$

4.2 Langage défini par une formule

On suppose ici que toutes les variables libres sont des variables d'ensemble. Les variables du premier ordre sont alors assimiliées à des ensembles singletons.

Si $X_1, \ldots, X_n$ sont les variables libres de ϕ, ϕ définit un langage sur l'alphabet $\{0,1\}^n$: si w est un mot infini sur cet alphabet, notons $\pi_i(w)$ $(1 \leq i \leq n)$ l'ensemble d'entiers naturels k tels que la $k$ième lettre de w ait pour $i$ème composante 1. w *satisfait* ϕ ssi $\{X_1 \mapsto \pi_1(w); \ldots; X_n \mapsto \pi_n(w)\} \models \phi$

Exemple 7 Le mot infini 10101010... satisfait la formule Pair(X).

Exemple 8 Prenons un autre exemple, à peine moins trivial. Si $\phi \stackrel{\text{def}}{=} x \geq y$, les mots $w_1 \stackrel{\text{def}}{=} \begin{smallmatrix} 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \end{smallmatrix} \ldots$ et $w_2 \stackrel{\text{def}}{=} \begin{smallmatrix} 1&0&0&0 \\ 1&0&0&0 \end{smallmatrix} \ldots$ satisfont ϕ : en les décryptant, w_1 code la paire d'ensembles $(\{4\}, \{2\})$ et w_2 code la paire $(\{0\}, \{0\})$.

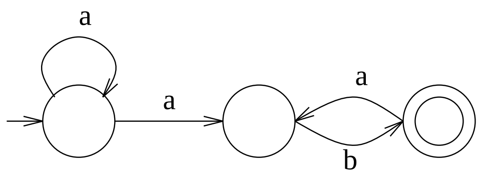

FIG. 3 – Automate de Büchi acceptant $a^*(ab)^\omega$

Inversement, étant donné un alphabet A à n éléments, la ième lettre de A peut être identifiée à $(0,\ldots,0,1,0,\ldots,0) \in \{0,1\}^n$, le 1 se trouvant en position i. Un mot infini w_1 sur l'alphabet A est alors identifié à un mot w_2 sur l'alphabet $\{0,1\}^n$ tel que les ensembles $\pi_i(w_2)$ forment une partition des entiers naturels.

Les formules de S1S constituent donc un (autre) moyen de définir des langages de mots infinis.

Le *langage* défini par une formule ϕ de S1S est l'ensemble des mots (infinis) qui satisfont ϕ.

4.3 Automates de Büchi

Un *automate de Büchi* sur un alphabet A est un quadruplet (Q, q_0, Q_f, R) où Q est un ensemble fini d'*états*, $q_0 \in Q$ est l'*état initial*, Q_f est un sous-ensemble d'*états finaux* de Q et R est un ensemble fini de règles de production de la forme $qa \vdash q'$ où $q, q' \in Q$ et $a \in A$.

Un mot infini w est *accepté* par l'automate (Q, Q_f, R) s'il existe une application ρ des entiers naturels dans Q telle que

- $\rho(0) = q_0$
- pour tout i, $\rho(i) = q, w(i) = a, \rho(i+1) = q' \Rightarrow qa \vdash q' \in R$ (ces deux conditions assurent que ρ est un *calcul* de l'automate sur w)
- il existe un ensemble infini d'indices i tels que $\rho(i) \in Q_f$ (c'est la condition d'acceptance de Büchi : elle exprime que le calcul est un succès).

Exemple 9 L'automate de la figure 3 accepte l'ensemble des mots infinis qui sont constitués d'une suite finie de a suivie d'une séquence infinie $abababab\ldots$. Ce que l'on note $a^*(ab)^\omega$.

En effet, pour de tels mots, on constuit un calcul de l'automate par $\rho(i) = q_1$ si $w(i+1) = a$ et ($i = 0$ ou $\rho(i-1) = q_1$), $\rho(i) = q_2$ si $w(i+1) = b$, $\rho(i) = q_3$ si $w(i) = b$.

Inversement, si w est accepté par l'automate, alors $\rho(i) = q_1 \Rightarrow \forall j < i, \rho(j) = q_1$ et $\rho(i) = q_2 \Rightarrow w(i) = a \wedge w(i+1) = b$, $\rho(i) = q_3 \Rightarrow w(i) = b \wedge w(i+1) = a$. Ce qu'on montre par récurrence sur i.

L'automate de la figure 4 accepte aussi le même langage. Il est obtenu en prenant q_2 au lieu de q_3 pour seul état final. Comme automates de mots finis, il accepterait un langage distinct de l'automate de la figure 3.

Remarquons que les automates des figures 3 et 4 sont non-déterministes. Il est malgré tout possible ici de calculer un automate déterministe acceptant le

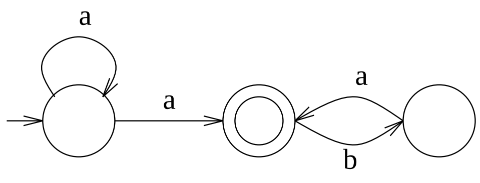

FIG. 4 – Un autre automate acceptant $a^*(ab)^\omega$

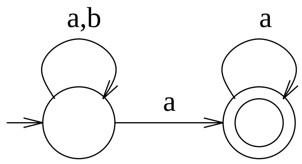

FIG. 5 – Un autre automate acceptant $(a+b)^*a^\omega$

même langage (ce qui est laissé en exercice). Mais il existe des langages reconnus par un automate de Büchi qui sont intrinsèquement non-déterministes, comme l'automate de la figure 5.

Mais si l'on excepte la déterminisation, les langages Büchi-reconnaissables possèdent les mêmes propriétés essentielles que les langages reconnaissables de mots finis : clôture par intersection, union (ce qui est facile à montrer), clôture par complémentation (ce qui est assez difficile à montrer, du fait de l'impossibilité de déterminiser), ainsi que la décidabilité du vide.

Proposition 10 *La classe des langages Büchi-reconnaissable est effectivement close par union et intersection.*

Exercice 6 Démontrer la proposition 10.

Proposition 11 *Le vide des automates de Büchi est dans NLOGSPACE.*

En fait, le problème est complet pour NLOGSPACE, ce que nous admettons.

Preuve: Il suffit de déterminer s'il existe un état final accessible et une boucle de l'automate sur cet état.

La mémoire de la machine de Turing contiendra trois données (outre l'entrée) : deux états q_1, q_2 et un compteur c borné par le nombre d'états (donc dont la représentation en binaire a une taille bornée par le logarithme de la taille de l'automate). La mémoire utilisée est donc logarithmique en la donnée.

La machine commence par écrire dans q_1 l'état initial de l'automate et affecte 0 à c.

Dans la première phase, la machine répète les opérations suivantes :

- si q_1 est un état final, on choisit de façon non-déterministe de passer (ou non) à la deuxième phase
- on choisit de façon non-déterministe une transition de l'automate $q_1 \xrightarrow{a} q_1'$ et on remplace q_1 par q_1'
- on incrémente c de 1.
- si c dépasse le nombre d'états on s'arrête en échec

Durant la deuxième phase, on commence par affecter q_1 à q_2 et on remet c à 0. Puis on répète les opérations suivantes :
- si $q_1 = q_2$ et $c > 0$ on s'arrête en succès
- on choisit de façon non-déterministe une transition de l'automate $q_2 \xrightarrow{a} q_2'$ et on remplace q_2 par q_2'
- on incrémente c de 1.
- si c dépasse le nombre d'états on s'arrête en échec

La correction d'un tel algorithme est assurée par le fait que si l'automate accepte au moins un mot infini, alors il existe un état final q_f tel que :
- q_f est accessible à partir de l'état initial par un chemin de longueur inférieure au nombre d'états
- Il existe une boucle sur q_f de longueur inférieure au nombre d'états.

□

La stabilité de la classe des langages Büchi reconnaissables par complémentaire fera l'objet d'un paragraphe spécialisé (cf. paragraphe 6) et nous l'admettons pour l'instant.

4.4 Théorème de Büchi

Le théorème de Büchi établit la décidabilité de S1S, en montrant qu'un langage de mots infinis est définissable en S1S si et seulement si il est accepté par un *automate de Büchi*.

Theorème 12 *Un langage est définissable en S1S si et seulement si il est accepté par un automate de Büchi.*

Preuve: Supposons tout d'abord que L est accepté par un automate d'états finis (c'est le sens facile de l'implication). Soit alors $Q = \{q_0, \ldots, q_m\}$ l'ensemble d'états, Q_f l'ensemble des états finaux, q_0 l'état initial, $A = \{a_1, \ldots, a_n\}$ l'alphabet et R l'ensemble de règles de production d'un automate (sans ϵ-transitions) acceptant L. L est alors défini par la formule :

$$
\begin{array}{ll}
\phi_L(X_{a_1},\ldots,X_{a_n}) \stackrel{\text{def}}{=} & \\
\exists X_{q_0}, X_{q_1},\ldots,X_{q_m}. & \text{partition}(X_{q_0},\ldots,X_{q_m}) \\
& \wedge \text{partition}(X_{a_1},\ldots,X_{a_n}) \\
& \wedge 0 \in X_{q_0} \\
& \wedge \displaystyle\bigwedge_{q\in Q, a\in A} \forall x \in X_q . x \in X_a \Rightarrow (\bigvee_{qa\vdash q'\in R} S(x) \in X_{q'}) \\
& \wedge \displaystyle\bigvee_{q\in Q_f} \text{Infini}(X_q)
\end{array}
$$

La deuxième ligne de cette formule exprime que les assignations qui satisfont ϕ_L sont bien des mots. Les lignes 1 et 3 construisent un *calcul* de l'automate en marquant chaque lettre par un état et en assurant que ce marquage est conforme aux transitions de l'automate. Enfin, la dernière ligne assure qu'on arrive dans un état final.

Pour l'implication inverse, on commence par se ramener à des formules plus simples : on applique d'abord le système de réécriture suivant (de simplification de formules) :

$$
\begin{array}{rcl}
S(t) = S(u) & \to & t = u \\
S(t) = 0 & \to & \bot \\
x = S(t) & \to & \exists y . y = t \wedge x = S(y) \\
& & \qquad\qquad \text{Si } t \text{ n'est pas une variable et } y \notin \mathcal{V}\text{ar}(x = t) \\
t \in X & \to & \exists y . y = t \wedge y \in X \\
& & \qquad\qquad \text{Si } t \text{ n'est pas une variable et } y \notin \mathcal{V}\text{ar}(t)
\end{array}
$$

Ces règles sont évidemment correctes. De plus, elles terminent. Les formules irréductibles pour ce système de règles ne contiennent que des formules atomiques d'une des formes suivantes : $x = 0, x = S(y), x \in X, x = y$.

On ajoute ensuite la construction syntaxique $X = S(Y)$ qui doit être interprétée comme : X et Y sont des singletons $\{x\}$ et $\{y\}$ respectivement et $x = y + 1$, la construction $X = Y$, interprétée de façon similaire ainsi que la construction $X = 0$ qui est interprétée comme : X est le singleton 0. Toutes les variables d'individus sont alors remplacées par des variables d'ensemble : $\exists x.\phi$ est remplacée par $\exists X.\phi\{x \mapsto X\}$ où $X \notin \mathcal{V}\text{ar}(\phi)$ et de même pour les quantifications universelles où les variables libres. Les expressions $X \in Y$ sont remplacées par $X \subseteq Y$. On obtient ainsi une formule équivalente dans laquelle les seules formules atomiques sont d'une des formes suivantes : $X = 0, X = S(Y), X = Y, X \subseteq Y$.

Montrons maintenant comment construire un automate (de paires) acceptant les ensembles d'ensembles satisfaisant chacune de ces formules atomiques. Ceux-ci sont donnés dans les figures 6, 7, 8 et 9 respectivement.

Ayant montré comment calculer les automates correspondant à chaque formule atomique, il nous reste à combiner ces formules atomiques. Tout d'abord,

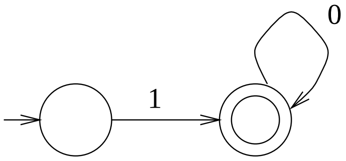

FIG. 6 – Automate acceptant le langage défini par $X = 0$

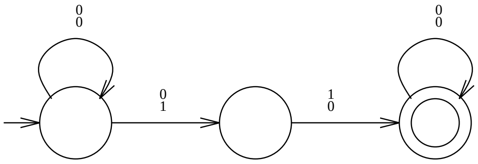

FIG. 7 – Automate acceptant le langage défini par $X = S(Y)$

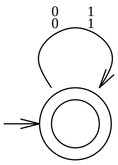

FIG. 8 – Automate acceptant le langage défini par $X = Y$

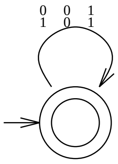

FIG. 9 – Automate acceptant le langage défini par $X \subseteq Y$

nous avons besoin d'une opération appelée *cylindrification* des automates. Etant donné un automate $\mathcal{A}$ sur l'alphabet $\{0,1\}^k$, le cylindrifié de $\mathcal{A}$ par rapport à $i \in \{1,\ldots,k+1\}$ est l'automate $\mathcal{A}'$ sur $\{0,1\}^{k+1}$ dont les états sont les mêmes que ceux de $\mathcal{A}$, les états finaux sont les mêmes que ceux de $\mathcal{A}$ et les transitions sont définies par :

$$q(a_1,\ldots,a_k) \vdash_{\mathcal{A}} q' \quad \text{ssi} \quad \forall a \in \{0,1\},\ q(a_1,\ldots,a_{i-1},a,a_i,\ldots,a_k) \vdash_{\mathcal{A}'} q'$$

Un mot w est alors accepté par $\mathcal{A}'$ ssi le mot w défini par $w(n) = (a_1,\ldots,a_k)$ ssi $w'(n) = (a_1,\ldots,a_{i-1},a,a_i,\ldots,a_k)$ pour un certain $a \in \{0,1\}$ est accepté par $\mathcal{A}$.

Supposons maintenant que les formules $\phi_1(\vec{Z},X_1,\ldots,X_n)$ et $\phi_2(\vec{Z},Y_1,\ldots,Y_m)$ ($\vec{Z}$ est l'ensemble des variables libres communes aux deux formules, que nous supposons ici sans perte de généralité, être listées en premier pour chacune des formules) définissent des langages L_1 et L_2 reconnus par les automates $\mathcal{A}_1$ et $\mathcal{A}_2$ respectivement. Par cylindrifications, on construit les automates $\mathcal{A}'_1$ et $\mathcal{A}'_2$ reconnaissant les langages définis par les formules $\phi'_1(\vec{Z},\vec{X},\vec{Y})$ et $\phi'_2(\vec{Z},\vec{X},\vec{Y})$ respectivement, où ϕ_i et ϕ'_i sont identiques aux variables libres près. Par intersection (resp. union) des automates $\mathcal{A}'_1$ et $\mathcal{A}'_2$ on obtient alors un automate qui accepte le langage défini par $\phi_1 \wedge \phi_2$ (resp. $\phi_1 \vee \phi_2$). On utilise ici la clôture par intersection et union de la classe des langages reconnus par les automates de Büchi.

Étant donné un automate acceptant le langage défini par ϕ, on peut construire un automate acceptant le langage défini par $\neg\phi$ par complémentation de l'automate. On utilise ici la clôture par complémentaire de la classe des langages. (Ce que nous ne montrons pas ici).

Étant donné un automate acceptant le langage défini par $\phi(X_1,\ldots,X_n)$, l'automate acceptant le langage défini par $\exists X_i,\ \phi(X_1,\ldots,X_n)$ s'obtient par *projection* selon la ie'me composante : Il suffit de conserver les mêmes états, les mêmes états finaux, et de remplacer chaque transition $q(a_1,\ldots,a_n) \vdash q'$ par $q(a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_n) \vdash q'$.

Il reste enfin la quantification universelle, mais celle-ci s'obtient par double complémentation : $\forall X,\ \phi$ est logiquement équivalente à $\neg\exists X,\ \neg\phi$.

Ainsi, par récurrence sur la structure des formules, tout langage définissable est accepté par un automate d'états finis, ce qui complète la preuve. □

Exercice 7

1. Calculer directement un automate de Büchi correspondant à la fromule

$$\text{Cone}(x,X) \stackrel{\text{def}}{=} \forall y \in X.y \geq x \wedge \forall z.(z \geq y) \Rightarrow z \in X)$$

2. Calculer directement un automate de Büchi pour la formule

$$\text{IdProj}(X,Y,Z) \stackrel{\text{def}}{=} X \cap Z = Y \cap Z$$

3. Calculer l'automate de Büchi correspondant à la formule suivante de S1S :

$$\exists Z.\exists x.x \in X \wedge S(x) \in Z \wedge \mathrm{Cone}(S(x), Z) \wedge \mathrm{IdProj}(X, Y, Z)$$

Corollaire 13 *S1S est décidable.*

Cependant un résultat de Meyer et Stockmeyer (1973) montre que cette logique, bien que décidable, est non-élémentaire (i.e. on ne peut borner la complexité par une tour d'exponentielles).

4.5 Pouvoir d'expression

On remarque que la preuve du théorème de Büchi montre aussi que toute formule de S1S est équivalente à une formule purement existentielle.

Par contre, S1S est strictement plus expressive que la logique monadique du premier ordre (et donc que LTL par le théorème de Kamp). En fait, il existe de nombreuses caractérisations des langages définissables au premier ordre, en particulier la célèbre caractérisation de McNaughton : "un langage est définissable au premier ordre si et seulement si il est sans étoile" (voir par exemple [Tho90] pour une brève synthèse). On sait aussi décider si un langage est sans étoile, grâce à la caractérisation : un langage est sans étoile si et seulement si son monoide syntaxique est sans-groupe".

Par exemple, on peut écrire une formule de S1S correspondant à l'énoncé :

> X et Y sont des ensembles tels que, entre deux entiers quelconques qui appartiennent à X et à Y, il y a un nombre pair d'entiers qui appartiennent à X.

mais pas une formule de LTL.

4.6 Logique monadique faible de second ordre (WS1S)

La syntaxe de WS1S est la même que celle de S1S. L'interprétation des formules est la même, excepté que les variables de second ordre sont interprétées comme des sous-ensembles *finis* de $\mathbb{N}$.

Le langage associé à une formule est donc désormais un ensemble de mots finis. Un analogue du théorème de Büchi est le résultat (plus simple à prouver à cause du complémentaire) :

Theorème 14 *Un ensemble de mots est définissable en WS1S si et seulement si il est reconnu par un automate de mots (finis).*

Malheureusement, WS1S est aussi non-élémentaire.

Une application de ce résultat est un algorithme de décision pour une extension de l'arithmétique de Presburger : les entiers peuvent être vus comme mots sur $\{0, 1\}$ (écriture en binaire), le bit de poids fort étant à droite. Par exemple treize s'écrira 1011. Le codage n'est pas unique. Par exemple 10110 représentera aussi le nombre treize. En fait, à ces représentations binaires peuvent aussi être associées des ensembles finis d'entiers, ce qui définit une application bijective des ensembles finis d'entiers dans les entiers : à un mot sur $\{0,1\}$ est associé l'ensemble des positions de 1. Par exemple, 1011 représente l'ensemble $\{0,2,3\}$.

Exercice 8 Si x est un mot sur $\{0,1\}$ on note $\widehat{x}$ l'entier dont x est une écriture en binaire de droite à gauche.

1. Donner un automate (à deux états) acceptant les triplets de mots $x, y, z \in \{0,1\}^*$ tels que $\widehat{x} = \widehat{y} + \widehat{z}$.
2. En déduire que l'arithmétique de Presburger est décidable.
3. Donner un exemple d'ensemble d'entiers définissable en WS1S, mais pas dans l'arithmétique de Presburger.

5 Satisfaisabilité et vérification par des techniques d'automates : le cas du temps linéaire

La notion clef pour obtenir des algorithmes efficaces est celle d'automates alternants : ceux-ci permettent de décrire de façon compacte les modèles d'une formule de logique temporelle. C'est ce que nous montrons tout d'abord. Puis nous étudions l'algorithmique des automates alternants, montrant qu'on obtient bien des algorithmes optimaux pour le model-checking (PSPACE-complet) et la satisfaisabilité (PSPACE-complet). (cf. [Var96]).

5.1 Automates de Büchi alternants

Un automate de Büchi alternant est un tuple $(Q, \Sigma, Q_f, q_0, \delta)$ où S ensemble d'états (finis), Σ alphabet, $Q_f \subseteq Q$, $q_0 \in Q$, transitions : $\delta : S \times Q \to \mathcal{B}^+(Q)$. $\mathcal{B}^+(Q)$ est l'ensemble des combinaisons Booléennes positives d'éléments de Q : ce sont des formules propositionnelles sans négation sur l'alphabet Q de variables propositionnelles,

Un *calcul* de A sur $w \in \Sigma^\omega$ est un arbre (fini ou infini) étiqueté par Q et tel que : $r(\epsilon) = q_0$ et si p est une position de longueur i telle que $r(p) = q$ et $w(i) = a$ et $\delta(q, a) = \phi$, l'ensemble $\{p \cdot 1, ..., p \cdot n\}$ des successeurs de p est tel que $r(p \cdot 1), ..., r(p \cdot n) \models \phi$.

La notion de satisfaction est ici définie par : $q_1, \ldots, q_n \models q$ ssi $q \in \{q_1, \ldots, q_n\}$ puis s´étend au calcul Booléen comme d'habitude. (En particulier, pour tout ensemble d'états E, $E \models$ **true** et $E \not\models$ **false**).

Un calcul de l'automate alternant est *réussi* si tout chemin infini contient une infinité d'états finaux. (NB : il peut ne pas exister de calcul au cas où **false** est présent comme résultat d'une transition ; il peut exister des calculs non réussis. Les calculs peuvent être des arbres finis, au quel cas ils sont nécessairement réussis).

Exemple 15 Soit l'automate alternant défini par $Q = \{q_0, q_1\}$, $\Sigma = \{a, b, c, d\}$ et la fonction de transition donnée par la table :

δ	a	b	c	d
q_0	**true**	**true**	$q_0 \wedge q_1$	q_0
q_1	**false**	**true**	q_1	**true**

Un exemple de calcul sur le mot $cccdcdad \cdots$ est représenté dans la figure 10.

On peut passer des automates alternants aux automates non-déterministes au prix d'une exponentielle :

Lemme 16 *Soit $\mathcal{A}$ un automate de Büchi alternant, d'ensemble d'états Q et acceptant le langage L. Il existe un automate de Büchi non déterministe $\mathcal{A}'$ dont l'ensemble des états est $2^Q \times 2^Q$ et qui accepte L.*

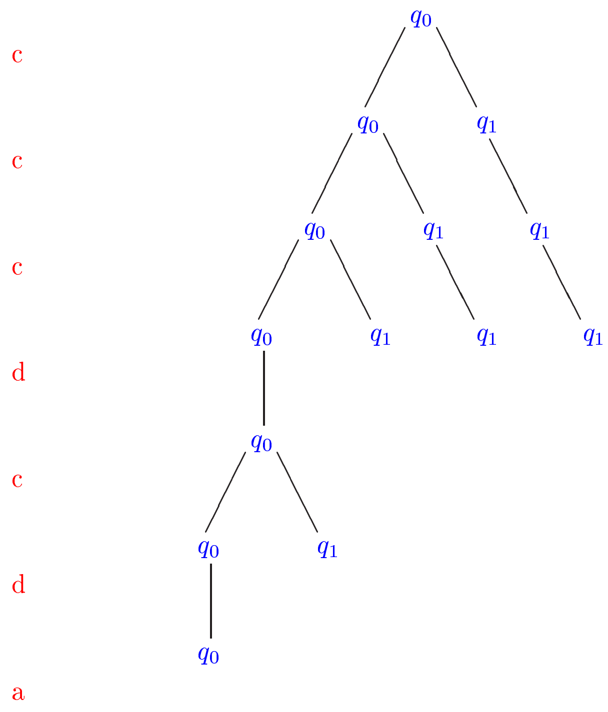

FIG. 10 – Un calcul de l'automate alternant sur le mot $ccc\ldots$

Preuve: On construit $\mathcal{A}'$ de la manière suivante : les états sont constitués des paires d'ensembles d'états de $\mathcal{A}$. L'état initial est $(\{q_0\}, \emptyset)$ et les états finaux sont les paires $(\emptyset, S)$. Si F est l'ensemble des états finaux de $\mathcal{A}$, les transitions $((S, S'), a, (S_1, S_1')) \in \delta'$ sont données par :

Si $S \neq \emptyset$ alors S_1 et S_1' sont tels que

$$\bigwedge_{q_1 \in S_1} q_1 \models \bigwedge_{q \in S \setminus F} \delta(q, a)$$

$$\bigwedge_{q_1 \in S_1'} q_1 \models \bigwedge_{q \in S'} \delta(q, a) \wedge \bigwedge_{q \in S \cap F} \delta(q, a)$$

Si $S = \emptyset$ alors $S_1' = \emptyset$ et S_1 est tel que

$$\bigwedge_{q_1 \in S_1} q_1 \models \bigwedge_{q \in S'} \delta(q, a)$$

Supposons d'abord que ρ est un calcul réussi de $\mathcal{A}$ sur w et construisons un calcul réussi r de $\mathcal{A}'$ sur w. (S_i, S_i') est défini par récurrence : $S_0 = \{q_0\}$ et $S_0' = \emptyset$. $S_i \cup S_i'$ restera l'ensemble des étiquettes des positions de longueur i dans ρ. On partage les positions de longueur i en deux ensembles disjoints : P_i et P_i' qui sont définis par récurrence par $P_0 = \{\epsilon\}$, $P_0' = \emptyset$.

Si $S_i \neq \emptyset$

$$P_{i+1} \stackrel{\text{def}}{=} \{p \cdot j \mid p \in P_i, \rho(p) \notin F, p \cdot j \in \mathrm{Pos}(\rho)\}$$

$$P_{i+1}' \stackrel{\text{def}}{=} \{p \cdot j \mid p \in P_i', p \cdot j \in \mathrm{Pos}(\rho)\} \cup \{p \cdot j \mid p \in P_i, \rho(p) \in F, p \cdot j \in \mathrm{Pos}(\rho)\}$$

Si $S_i = \emptyset$ P_{i+1} est l'ensemble des positions de longueur $i + 1$ de ρ et $P_{i+1}' = \emptyset$.

Dans tous les cas, $S_{i+1} \stackrel{\text{def}}{=} \{\rho(p) \mid p \in P_{i+1}\}, S_{i+1}' \stackrel{\text{def}}{=} \{\rho(p) \mid p \in P_{i+1}'\}$. On vérifie d'abord que $((S_i, S_i'), a, (S_{i+1}, S_{i+1}')) \in \delta'$ pour tout i, ce qui est une conséquence des définitions de δ' et de ρ. Il reste à montrer que le calcul r défini par $r(i) = (S_i, S_i')$ est un calcul réussi. Par l'absurde : si r ne contient qu'un nombre fini de lettres telles que $S_i = \emptyset$, alors, pour $i > i_0$, $S_i \neq \emptyset$. Donc, pour tout $i > i_0$, $P_i \neq \emptyset$. Soit ρ' le sous-arbre de ρ ne contenant que les positions de P_i, pour $i > i_0$ (et les mêmes positions que ρ pour $i \leq i_0$). ρ' est bien un arbre puisque, si $p \cdot j \in P_{i+1}$ et $i > i + 0$, alors $p \in P_i$. D'autre part ρ' est un arbre infini puisque P_i est non vide pour tout i. Donc ρ' contient un chemin infini (lemme de König). Sur ce chemin, il n'y a aucun état final, ce qui est absurde.

Réciproquement, supposons que r est un calcul réussi de $\mathcal{A}'$ sur w. On construit par récurrence un calcul ρ de l'automate alternant, gardant comme invariant le fait que $S_i \cup S_i'$ est l'ensemble des étiquettes des positions de longueur i de ρ. Si $\rho(p) = q$, $w(|p|) = a$ et $r(|p|) = (S, S')$, $r(|p| + 1) = (S_1, S_1')$, par définition, il existe un sous-ensemble de $S_1 \cup S_1'$, soit S_q, tel que $\bigwedge_{q' \in S_q} q' \models \delta(q, a)$. De plus, si $q \in S_i \setminus F$, on peut supposer $S_q \subseteq S_{i+1}$. Alors, on ajoute à ρ les positions $p \cdot 1 \ldots p \cdot m$ ($m = |S_q|$) étiquetées par les éléments de S_q. ρ est bien un calcul par construction. Il reste à montrer que chaque branche contient une infinité d'états finaux. Par l'absurde : supposons qu'il existe un chemin infini C de ρ qui ne contient qu'un nombre fini d'états finaux : pour $p \in C$ et $|p| > n$, $\rho(p) \notin F$. Par récurence sur la longueur de p, on montre $\rho(p) \in S_i \setminus F$ puisqu'alors $S_{\rho(p)} \subseteq S_{i+1}$ et $S_{\rho(p)} \cap C \neq \emptyset$. On obtient ainsi que, pour $i > n$, $S_i \neq \emptyset$, ce qui est absurde. $\square$

5.2 LTL et automates alternants

La correspondance entre automates alternants et formules temporelles est donné par le résultat très simple :

Theorème 17 *Soit ϕ une formule de LTL. Il existe un automate de Büchi alternant $\mathcal{A}_\phi$ de taille $O(|\phi|)$ qui accepte exactement les mots de $(\{0,1\}^n)^\omega$ qui satisfont ϕ.*

Preuve: Les états seront les sous-formules de ϕ et leur négation. Les états finaux sont les formules de la forme $\neg(\phi_1 \,\mathcal{U} \psi_1)$. L'état initial est ϕ.

La relation de transition est donnée par :

$\delta(p, a) \stackrel{\text{def}}{=} \textbf{true}$ si $p \in a$

$\delta(p, a) \stackrel{\text{def}}{=} \textbf{false}$ si $p \notin a$

$\delta(\neg p, a) \stackrel{\text{def}}{=} \textbf{true}$ si $p \notin a$

$\delta(\neg p, a) \stackrel{\text{def}}{=} \textbf{false}$ si $p \in a$

$\delta(\phi_1 \wedge \phi_2, a) \stackrel{\text{def}}{=} \delta(\phi_1, a) \wedge \delta(\phi_2, a)$

$\delta(\phi_1 \vee \phi_2, a) \stackrel{\text{def}}{=} \delta(\phi_1, a) \vee \delta(\phi_2, a)$

$\delta(\neg\phi_1, a) \stackrel{\text{def}}{=} \overline{\neg\delta(\phi_1, a)}$

$\delta(\mathbf{X}\phi_1, a) \stackrel{\text{def}}{=} \phi_1$

$\delta(\phi_1 \,\mathcal{U} \phi_2, a) \stackrel{\text{def}}{=} \delta(\phi_2, a) \vee (\delta(\phi_1, a) \wedge \phi_1 \,\mathcal{U} \phi_2)$

où $\overline{\psi}$ est la normalisation de ψ par les règles : $\neg(\phi \vee \psi) \to \neg\phi \wedge \neg\psi$, $\neg(\phi \wedge \psi) \to \neg\phi \vee \neg\psi$, $\neg\neg\phi \to \phi$.

La preuve du théorème s'effectue alors par récurrence sur la taille de la formule : on montre par récurrence sur ϕ que $\mathcal{A}_\phi$ accepte les mots sur $(\{0,1\}^k)^\omega$ qui satisfont ϕ.

Cas de base. Si $\phi = \textbf{true}$ (resp. **false**), la transition par toutes les lettres est **true** (resp. **false**) et tous les mots sont acceptés (resp. non acceptés).

Si ϕ est une variable propositionnelle p (resp. sa négation $\neg p$), $w \models \phi$ si et seulement si $p \in w(0)$ (resp. $p \notin w(0)$). Un calcul ρ de $\mathcal{A}_\phi$ sur w est un mot d'une lettre : ϕ lui-même et tous les calculs sont réussis. Par définition de δ, il existe un calcul sur w ssi $p \in w(0)$.

récurrence On envisage tous les cas de construction de ϕ :

Xϕ_1 Par définition de $\mathcal{A}_\phi$, l'automate est identique à $\mathcal{A}_{\phi_1}$, avec un état en plus (qui devient l'état initial) : ϕ, et une transition supplémentaire $\delta(\phi, a) \stackrel{\text{def}}{=} \phi_1$ pour tout a.

$w \models \phi$ ssi $w|_1 \models \phi_1$ ssi (par hypothèse de récurrence) il existe un calcul réussi ρ_1 de $\mathcal{A}_{\phi_1}$ sur $w|_1$ ssi le calcul $\rho \stackrel{\text{def}}{=} (\phi) \cdot \rho_1$ est un calcul réussi de $\mathcal{A}_\phi$ sur w.

$\phi_1 \wedge \phi_2$ $w \models \phi_1 \wedge \phi_2$ ssi $w \models \phi_1$ et $w \models \phi_2$, ssi, par hypothèse de récurrence, il existe des calculs réussis ρ_1 et ρ_2 de $\mathcal{A}_{\phi_1}$ et $\mathcal{A}_{\phi_2}$ sur w.

Etant donnés deux calculs ρ_1 de $\mathcal{A}_{\phi_1}$ et ρ_2 de $\mathcal{A}_{\phi_2}$ respectivement sur w, $\rho_1 \oplus \rho_2$ est l'arbre dont la racine est étiquetée par $\phi_1 \wedge \phi_2$, les positions sont $Pos(\rho_1) \cup \{(k+i) \cdot q \mid i \cdot q \in Pos(\rho_2)\}$ si k est le branchement à la racine de ρ_1. De plus $(\rho_1 \oplus \rho_2)(i \cdot q) = \rho_1(i \cdot q)$ si $i \leq k$ et $(\rho_1 \oplus \rho_2)(i \cdot q) = \rho_2((i-k) \cdot q)$ sinon. Par définition de δ, $\rho_1 \oplus \rho_2$ est un calcul de $\mathcal{A}$ sur w et, inversement, tout calcul ρ de $\mathcal{A}_\phi$ sur w se décompose en $\rho_1 \oplus \rho_2$. Enfin, $\rho_1 \oplus \rho_2$ est réussi si et seulement si tous les chemins de $\rho_1 \oplus \rho_2$ contiennent une infinité d'états finaux, si et seulement si tous les chemins de ρ_1 et tous les chemins de ρ_2 contiennent une infinité d'états finaux si et seulement si ρ_1 et ρ_2 sont des calculs réussis.

Ainsi, $w \models \phi_1 \wedge \phi_2$ ssi il existe des calculs réussis ρ_1 et ρ_2 sur w ssi il existe un calcul réussi $\rho = \rho_1 \oplus \rho_2$ de $\mathcal{A}_{\phi_1 \wedge \phi_2}$ sur w.

$\phi_1 \vee \phi_2$ $w \models \phi_1 \vee \phi_2$ si et seulement si $w \models \phi_1$ ou $w \models \phi_2$ ssi il existe un calcul réussi de $\mathcal{A}_{\phi_1}$ sur w ou il existe un calcul réussi de $\mathcal{A}_{\phi_2}$ sur w. Or les calculs réussis de $\mathcal{A}_{\phi_1 \vee \phi_2}$ sont ceux de $\mathcal{A}_{\phi_1}$ et ceux de $\mathcal{A}_{\phi_2}$, remplaçant seulement l'étiquetage de la racine par $\phi_1 \vee \phi_2$.

DOnc $w \models \phi_1 \vee \phi_2$ ssi il existe un calcul réussi de $\mathcal{A}_{\phi_1 \vee \phi_2}$ sur w.

$\phi\, \mathcal{U} \phi_2$ Supposons d'abord que $w \models \phi_1 \,\mathcal{U} \phi_2$. Alors, par définition de la relation de satisfaction, il existe un entier n tel que $w|_n \models \phi_2$ et, pour tout $0 \leq i < n$, $w|_i \models \phi_1$. Montrons par récurrence sur n qu'il existe un calcul réussi de $\mathcal{A}_\phi$ sur w.

Si $n = 0$ alors $w \models \phi_2$ et, par la première hypothèse de récurrence, il existe un calcul réussi ρ_2 de $\mathcal{A}_{\phi_2}$ sur w. Ce calcul, à l'étiquetage de la racine près, est aussi un calcul de $\mathcal{A}_\phi$, par définition de $\delta(\phi_1 \,\mathcal{U} \phi_2)$.

Si $n > 0$ par définition de la relation de satisfaction, $w \models \phi_1$ et $w|_1 \models \phi_1 \,\mathcal{U} \phi_2$. Alors, par la première hypothèse de récurrence, il existe un calcul réussi ρ_1 de $\mathcal{A}_{\phi_1}$ sur w et, par la deuxième hypothèse de récurrence, il existe un calcul réussi ρ_2 de $\mathcal{A}_\phi$ sur $w|_1$. On obtient alors un calcul réussi $(1 \cdot \rho_2) \oplus \rho_1$ de $\mathcal{A}_\phi$ sur w.

Réciproquement, s'il existe un calcul réussi ρ de $\mathcal{A}_\phi$ sur w, par définition des états finaux, ρ ne contient aucun chemin qui soit étiqueté par ϕ

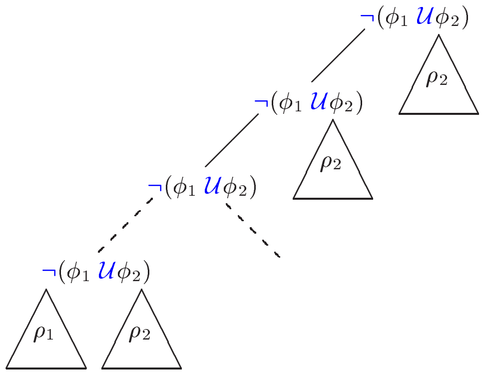

FIG. 11 – Un calcul réussi de $\mathcal{A}_{\neg(\phi_1\,\mathcal{U}\phi_2)}$ sur w

exclusivement. Soit alors n la longueur maximale d'une position de ρ étiquetée par ϕ. Montrons, par récurrence sur n, que $w \models \phi$.

Si $n = 0$ alors, par définition de δ sur $\phi_1\,\mathcal{U}\phi_2$, en remplaçant ϕ par ϕ_2 à la racine de ρ, on obtient encore un calcul réussi sur w. Par hypothèse de récurrence on a alors $w \models \phi_2$ et donc $w \models \phi_1\,\mathcal{U}\phi_2$.

Si $n > 0$ alors, par définition de δ, ρ se décompose $\rho = 1 \cdot \rho_1 \oplus \rho_2$ où ρ_1 est un calcul réussi de $\mathcal{A}_\phi$ sur $w|_1$ et ρ_2 est un calcul réussi de $\mathcal{A}_{\phi_1}$ sur w. Par la première hypothèse de récurrence on a alors $w \models \phi_1$ et, par la deuxième hypothèse de récurrence, $w|_1 \models \phi_1\,\mathcal{U}\phi_2$. D'où $w \models \phi_1\,\mathcal{U}\phi_2$.

$\neg(\phi_1\,\mathcal{U}\phi_2)$

Si $w \models \neg(\phi_1\,\mathcal{U}\phi_2)$, alors ou bien , pour tout i, $w|_i \models \neg\phi_2$, ou bien il existe un n tel que $w|_n \models \neg\phi_1 \wedge \neg\phi_2$ et, pour tout $i < n$, $w \models \neg\phi_2$.

Dans le premier cas, par hypothèse de récurrence, pour tout i, il existe un calcul réussi ρ_i de $\mathcal{A}_{\neg\phi_2}$ sur $w|_i$ et on construit ρ de la manière suivante : $Pos(\rho) = \bigcup_{i\in\mathbb{N}}\{1^i \cdot (j+1) \cdot q \mid j \cdot q \in Pos(\rho_i)\}$ et $\rho(1^i \cdot (j+1 \cdot q)) = \rho_i(j \cdot q)$, $\rho(1^i) = \phi$. ρ est bien un calcul réussi, par définition des états finaux et puisque $\delta(\neg(\phi_1\,\mathcal{U}\phi_2), a) = \delta(\neg\phi_2, a) \wedge (\delta(\neg\phi_1, a) \vee \neg(\phi_1\,\mathcal{U}\phi_2))$.

Dans le deuxième cas, représenté sur la figure 11, on procède par récurrence sur n pour construire un calcul réussi sur w :

Si $n = 0$, $w \models \neg\phi_1 \wedge \neg\phi_2$ et, par hypothèse de récurrence, il existe un calcul réussi de $\mathcal{A}_{\neg\phi_1}$ sur w et un calcul réussi de $\mathcal{A}_{\neg\phi_2}$ sur w, il est alors facile de construire un calcul réussi $\mathcal{A}_\phi$: on procède comme pour l'intersection. (Rappelons aussi que $\delta(\neg\phi_2, a) \wedge \delta(\neg\phi_1, a) \models \delta(\phi, a)$ par définition).

Si $n > 0$ On construit le calcul ρ à partir du calcul réussi sur $w|_1$ (utilisant la deuxième hypothèse de récurrence) et du calcul réussi de $\mathcal{A}_{\neg\phi_2}$ sur w (utilisant la prmière hypothèse de récurrence).

Réciproquement, si l'on suppose que w est accepté par $\mathcal{A}_{\neg(\phi_1 \mathcal{U} \phi_2)}$, soit ρ un calcul réussi sur w. Par définition de la relation de transition ρ est de la forme représentée sur la figure 11, où ρ_1 peut le cas échéant ne jamais être présent (condition d'acceptance). Dans chacun des cas (cas où il n'y a pas de ρ_1 et cas où il y en a un) on montre que $w \models \neg(\phi_1 \mathcal{U} \phi_2)$, par récurrence dans le deuxième cas.

Remarquons, qu'on peut facilement calculer les transitions pour les modalités dérivées $\square$ et $\lozenge$:

$$\begin{array}{rcl} \delta(\square\phi_1, a) & \stackrel{\text{def}}{=} & \delta(\phi_1, a) \wedge \square\phi_1 \\ \delta(\lozenge\phi_1, a) & \stackrel{\text{def}}{=} & \delta(\phi_1, a) \vee \lozenge\phi_1 \end{array}$$

Exemple 18 Soit $\phi = \lozenge\square\neg Q$. On obtient l'automate associé suivant, après avoir retiré les états inaccessibles : $Q = \{\phi, \square\neg Q\}$ avec pour seul état final $\square\neg Q$ et la relation de transition (on suppose qu'il y a deux variables propositionnelles P et Q, mais comme P ne joue aucun rôle, seules les transitions pour Q sont précisées) :

δ	1	0
$\square\neg Q$	**false**	$\square\neg Q$
ϕ	ϕ	$\square\neg Q \vee \phi$

Dans cet exemple, l'automate est en fait un automate de Büchi non-déterministe.

Exercice 9 Construire les automates alternants acceptant les mots qui satisfont :

1. $(P \mathcal{U} Q) \mathcal{U} (\neg P)$
2. $(\mathbf{X}\neg P) \mathcal{U} Q$

5.3 Satisfaisabilité de LTL et problème du non-vide des automates alternants

Le théorème 17 permet de ramener le problème de la satisfaisabilité d'une formule de LTL au problème du vide des automates alternants ; commençons donc par établir :

Theorème 19 *Le vide des automates de Büchi alternants est dans PSPACE.*

Preuve: En fait, il suffit d'utiliser la proposition 11 le lemme 16 : l'automate du lemme 16 est calculé à la volée (par nécessité) et on utilise l'algorithme de la proposition 11 pour lequel un espace logarithmique suffit.

Theorème 20 *La satisfaisabilité de LTL est PSPACE-complète*

Preuve: L'appartenance à PSPACE résulte des théorèmes 17 et 19.

Reste à montrer que le problème est PSPACE-difficile. Il y a plusieurs preuves, plus ou moins compliquées selon les fragments de LTL que l'on veut montrer difficiles pour PSPACE (cf. [DS02]). Nous choisissons ici une preuve peu raffinée mais simple, par codage direct des machines de Turing déterministes en espace polynômial.

Si T est une machine de Turing travaillant en espace polynômial, et w un mot, on calcule en temps polynômial une formule de LTL $\phi_{T,w}$ telle que T accepte w si et seulement si $\phi_{T,w}$ est satisfaisable.

Si l'espace de travail de T est consitué des positions $1, \ldots, n$ du ruban, on considère les variables propositionnelles $R_{a,i}$ pour $i \in \{1, \ldots, n\}$ et a dans l'alphabet d'entrée, ainsi que les variables $P_{q,i}$ pour q état de la machine. Une configuration est alors représentée par une conjonction de variables propositionnelles : $R_{a,i}$ code le fait que la $i$ème lettre du ruban est a et $P_{q,i}$ code le fait que la tête de lecture est en i et la machine dans l'état q.

Chaque position d'un mot infini doit représenter une configuration valide :

– A chaque position du ruban, il y a une et une seule lettre :

$$\Box \bigwedge_{i=1}^{n} (\bigvee_{a \in A} R_{a,i}) \wedge \bigwedge_{a \neq b} \neg R_{a,i} \vee \neg R_{b,i})$$

– Il y a une et une seule position du ruban sur laquelle pointe la tête de lecture et la machine est à chaque instant dans un et un seul état :

$$\Box(\bigvee_{q \in Q} \bigvee_{i=1}^{n} P_{q,i}) \wedge (\bigwedge_{q \neq q'} \bigwedge_{i} \neg P_{q,i} \vee \neg P_{q',i}) \wedge (\bigwedge_{i \neq j} \bigwedge_{q \in Q} \neg P_{q,i} \vee \neg P_{q,j})$$

On exprime ensuite que l'on commence par la configuration initiale :

$$P_{q_0,1} \wedge R_{a_1,1} \wedge \ldots \wedge R_{a_n,n}$$

si $w = a_1 \ldots a_n$.

Un mot satisfaisant la formule doit correspondre à un calcul de la machine. Par exemple, pour les déplacements à droite de la tête de lecture :

$$\Box \bigwedge_{(q,a,q',R,b) \in \Delta} \bigwedge_{i=1}^{n} (P_{q,i} \wedge R_{a,i}) \Rightarrow \mathbf{X}(P_{q',i+1} \wedge R_{b,i})$$

Enfin, on exprime qu'un calcul est réussi :

$$\Diamond \bigvee_{q_f \in Q_f} \bigvee_{i=1}^{n} P_{q_f,i}$$

ce qui termine le codage.

On peut remarquer que, si une formule de LTL est satisfaisable, alors elle admet un modèle fini : *LTL a la propriété de modèles finis.* Par contre :

Exercice 10 Donner deux structures de Kripke distinctes qui satisfont les mêmes formules de LTL.

5.4 LTL :Vérification de modèles. Une technique optimale utilisant les automates

Soit $\mathcal{K}$ une structure de Kripke et ϕ une formule de logique temporelle. À ϕ est associé l'automate $\mathcal{A}_\phi$ comme au théorème 17. $\mathcal{K} \models \phi$ si et seulement si $L_\mathcal{K}$, ensemble des executions de $\mathcal{K}$ est contenu dans $L(\mathcal{A}_\phi)$, ou, de façon équivalente, si

$$\mathcal{K} \models \phi \quad \text{ssi} \quad L_\mathcal{K} \cap L(\mathcal{A}_{\neg\phi}) = \emptyset$$

Comme l'intersection de deux automates de Büchi $\mathcal{A}_1$ et $\mathcal{A}_2$ peut être calculée en temps $O(|\mathcal{A}_1| \times |\mathcal{A}_2|)$, on obtient ainsi que le problème de model-checking est dans PSPACE. La completude pour PSPACE s'obtient simplement en considérant pour $\mathcal{K}$ l'automate de Büchi qui accepte tous les mots sur $\{0,1\}^n$.

Theorème 21 *Le model-checking pour LTL est PSPACE-complet.*

Si l'on fixe une formule ϕ, savoir si un automate de Büchi $\mathcal{A}$ satisfait ϕ est NLOGSPACE-complet.

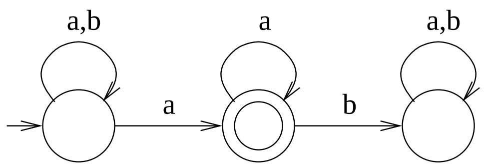

FIG. 12 – Un automate de Büchi acceptant $(a+b)^*a^\omega$

6 Jeux à parité et le théorème de complémentation

L'objectif de ce paragraphe est de démontrer la clôture par complémentaire de la classe des langages reconnus par un automate de Büchi. Nous avons choisi la preuve donnée dans [Tho99] parce qu'elle utilise les automates alternants et permet d'introduire des notions utiles pour la suite, typiquement la vérification en μ-calcul. Il s'agit des conditions d'acceptance à parité, des automates faibles, des jeux à deux joueurs et du théorème de détermination des jeux.

6.1 Quelques remarques sur les automates de Büchi

Tout d'abord, il n'est pas toujours possible de déterminiser un automate de Büchi :

Exercice 11 Montrer qu'il n'existe aucun automate de Büchi déterministe qui accepte le langage défini par l'automate non-déterministe de la figure 12.

En fait, ce problème de non-déterminisme va pouvoir être résolu grâce aux automates alternants : les automates non-déterministes correspondent à des automates alternants ne comportant que des disjonctions dans les transitions (et pas de conjonction). L'introduction de formules Booléennes positives arbitraires permet de symétriser les transitions et on peut donc espérer que la complémentation est plus simple.

Mais il existe un deuxième problème orthogonal, lié à la condition d'acceptance de Büchi elle-même :

Exercice 12 Montrer que, pour chacun des automates donnés dans les figures 13 et 14, bien que déterministes, il ne suffit pas d'échanger les états finaux et non finux pour obtenir un automate qui reconnaisse le complémentaire.

Ces exemples montrent que le problème de complémentation ne peut être traîté comme dans le cas des automates de mots finis.

Il existe cependant un cas où l'on sait bien complémenter les langages de mots infinis. Il a été vu dans la section 5 : si $\mathcal{A}_\phi$ est un automate (alternant) acceptant les modèles d'une formule LTL ϕ, $\mathcal{A}_{\neg\phi}$ accepte le complémentaire. La raison pour laquelle ce passage au complémentaire est simple est dû à la *faiblesse* de l'automate. On dira qu'un automate ed Büchi alternant est *faible* s'il existe un ordre sur les états tel que, pour tout état q, tout symbole a de l'alphabet et

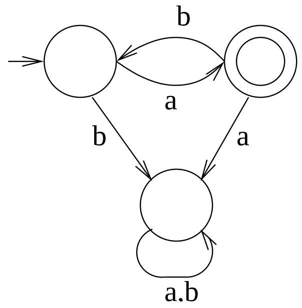

FIG. 13 – Un automate de Büchi déterministe acceptant $(ab)^\omega$

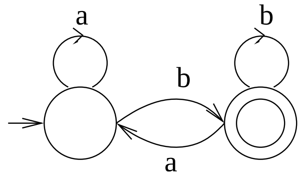

FIG. 14 – Un automate de Büchi déterministe acceptant $(a^*b)^\omega$

tout état q' apparaissant dans $\delta(q,a)$, $q' \leq q$. Les automates correspondant aux formules LTL sont faibles en ce sens : il suffit de choisir l'ordre sous-formule.

Exercice 13 Montrer que, pour un automate de Büchi déterministe et faible, il suffit d'échanger les états finaux et non-finaux pour obtenir un automate qui accepte le complémentaire.

Les automates de Büchi alternants faibles ne sont cependant pas assez expressifs pour que n'importe quel langage Büchi-reconnaissable soit reconnu par un tel automate. Il nous faut donc changer la condition pour accepter (c'est pourquoi on introduit une condition de parité) puis montrer que tout langage accepté par un automate de Büchi est accepté par un automate alternant à parité faible, puis montrer la complémentation des automates à parité faible et enfin retourner aux automates de Büchi. C'est ce que nous faisons dans les paragraphes qui suivent.

6.2 Automates alternants faibles à condition de parité

On suppose qu'à chaque état q est associé un entier $r(q)$ appelé *rang*. Un automate avec condition de parité est un automate fini comme précédemment, mais la condition pour accepter est remplacée par :

> Il existe un calcul ρ de $\mathcal{A}$ sur w tel que le minimum des rangs des états de ρ apparaissant infiniment souvent est pair.

Un automate alternant accepte un mot avec la condition de parité ssi il existe un calcul ρ de l'automate tel que tout chemin infini satisfait la condition de parité.

Un automate (alternant ou non) avec condition de parité est *faible*, si pour toute transition $\delta(q,a)=\phi$ et tout état $q'\in\phi$, $r(q')\leq r(q)$.

Exercice 14 Les indices dans la notation des états indiquent leur rang. L'état initial est q_3^1.

$$\begin{array}{lcl}
q_3^1 & \xrightarrow{a} & (q_2^1 \vee q_3^1) \wedge q_1^1 \\
q_3^1 & \xrightarrow{b} & (q_2^2 \vee q_3^1) \wedge q_0^1 \\
q_2^1 & \xrightarrow{b} & \mathbf{true} \\
q_2^2 & \xrightarrow{a} & \mathbf{true} \\
q_1^1 & \xrightarrow{b} & q_1^1 \\
q_1^1 & \xrightarrow{a} & q_1^1 \vee q_0^1 \\
q_0^1 & \xrightarrow{a} & q_0^1
\end{array}$$

les transitions non indiquées vont vers **false**.

L'automate est-il faible ?

Donner un calcul de l'automate sur le mot aba^ω.

Donner un calcul sur le mot $aaba^\omega$.

Décrire l'ensemble des mots acceptés.

Exercice 15 Calculer un automate alternant faible avec condition de parité pour le langage $(ab)^\omega$.

Exercice 16 Montrer que, pour toute formule LTL, on peut calculer en temps linéaire un automate alternant faible avec condition de parité qui accepte les modèles de ϕ.

Donner un automate alternant faible avec condition de parité qui n'accepte les modèles d'aucune formule de LTL (la preuve n'est pas demandée).

6.3 Passage des automates de Büchi aux automates alternants faibles avec condition de parité

Lemme 22 *Etant donné un automate de Büchi quelconque $\mathcal{A}$, on peut construire en temps polynômial un automate alternant faible avec condition de parité $\mathcal{A}'$ tel que $L(\mathcal{A}) = L(\mathcal{A}')$.*

Preuve: Soit $\mathcal{A}=(Q,Q_f,q_0,\Delta)$ un automate de Büchi. On construit $\mathcal{A}'$ de la manière suivante :

- l'ensemble des états est $Q\times\{0,\ldots,2\times n\}$ où n est le nombre d'éléments de Q (si bien que $\mathcal{A}'$ contient $O(n^2)$ états).
- Le rang est donné par la deuxième composante des états.
- L'état initial est $(q_0, 2n)$.
- La fonction de transition est définie par :
  - pour tout $q\in Q$, $(q,0)\xrightarrow{a}\bigvee_{(q,a,q')\in\Delta}(q',0)$. Autrement dit, sur la composante 0, $\mathcal{A}'$ se comporte comme $\mathcal{A}$.

– Si $i > 0$ est pair, $(q,i) \xrightarrow{a} \bigvee_{(q,a,q')\in\Delta}(q,i) \wedge (q,i-1)$. La conjonction force alors à rencontrer de nouveau un état pair, ce qui ne sera possible que si l'on rencontre un état final.
– Si i est impair et $q \notin Q_f$, alors $(q,i) \xrightarrow{a} \bigvee_{(q,a,q')\in\Delta}(q',i)$. Autrement dit, on reste sur le rang impair tant qu'on a pas rencontré un état final.
– Si i est impair et $q \in Q_f$, alors $(q,i) \xrightarrow{a} \bigvee_{(q,a,q')\in\Delta}(q',i-1)$.

Supposons tout d'abord que ρ est un calcul réussi de $\mathcal{A}$ sur w. On construit ρ', calcul de $\mathcal{A}'$ sur w, par récurrence sur la longueur de la position p de ρ', tel que $\rho'(p) = (\rho(|p|), i)$ pour un certain i :

– La racine est étiquetée par $(q_0, 2n)$.
– Si p est une position de ρ' étiquetée par (q,i) et $\rho(|p|) = q$, $w(|p|) = a$ et $\rho(|p|+1) = q'$, on ajoute un ou deux fils à p dans ρ', en respectant la relation de transition ci-dessus définie et de telle manière que chacun des fils soit étiqueté par un état de la forme (q',j).

Tous les chemins de ρ' sont alors infinis et contiennent une infinité d'états de la forme (q_f,i) où $q_f \in Q_f$. Comme l'automate est faible, sur chaque chemin le rang se stabilise sur une certaine valeur. Cette valeur ne peut pas être impaire puisque, lorsque le rang est impair, en passant par un état final, le rang décroit. Il en résulte que les rangs se stabilisent sur chaque branche sur des valeurs paires et donc le calcul est réussi.

Réciproquement, supposons que ρ' est un calcul acceptant de $\mathcal{A}'$ sur w. On définit une *tranche* de ρ' comme étant l'ensemble de tous les états étiquetant des nœuds de même profondeur. Remarquons que, si une tranche contient $(q, 2i)$, alors la tranche suivante contient un état $(q', 2i-1)$. Si le calcul est acceptant le nœud étiqueté $(q', 2i-1)$ doit avoir un descendant étiqueté par $(q'', 2i-2)$. Donc, par récurrence et puisque l'état initial est $(q_0, 2n)$, à partir d'une certaine profondeur, toutes les tranches d'un calcul acceptant contiennent, pour chaque $i \in \{0,\dots,n\}$ un état $(q_i, 2i)$. de plus comme, il n'y a qu'un nombre fini de tranches possibles, il existe une tranche S répétée infiniment souvent et telle que S contient au moins un état (q,i) pour chaque rang pair i. De plus, on peut supposer sans perdre de généralité (comme vu ci-dessus) que, de chaque état $(q, 2i)$ de S, il existe un chemin dans ρ' vers un état $(q', 2i-2) \in S$. On choisit alors de numéroter les états de Q de manière à ce qu'il existe un chemin dans ρ de $(q_i, 2i)$ vers $(q_{i-1}, 2i-2)$ pour $i > 0$. Par définition de $\mathcal{A}'$, ce chemin passe par un état $(q_i^f, 2i-1)$ tel que $q_i^f \in Q_f$.

Maintenant, au moins deux des états $q_0,\dots,q_n$ sont identiques. Disons $q_i = q_{i-k}$. On construit alors un calcul de $\mathcal{A}$ comme suit ; On commence par un chemin quelconque π_0 qui conduit à une position $(q_i, 2i)$ à une profondeur dont la tranche est S. On construit ensuite π_1 conduisant de $(q_i, 2i)$ à $(q_{i-k}, 2i-2k)$ (passant par $k+1$ tranches S). Si l'on oublie les deuxièmes composantes du mot $\pi_0\pi_1^\omega$, on obtient un calcul réussi de $\mathcal{A}$ sur w : comme $q_i = q_{i-k}$, il s'agit bien d'un calcul de $\mathcal{A}$ et, comme π_1 contient nécessairement une paire (q_f, j) avec $q_f \in Q_f$, il s'agit d'un calcul réussi.

Exercice 17 Construire un automate alternant faible avec condition de parité qui accepte le langage $(a+b)^*a^\omega$

Exercice 18 Soit $\mathcal{A}$ un automate de Büchi déterministe et complètement spécifié à n états. Construire un automate alternant faible avec condition de parité qui accepte le même langage et qui comporte $2n$ états.

Exercice 19 Construire un automate alternant à parité faible pour chacun des langages définis dans les figures 13 et 14.

Exercice 20 Montrer que, pour tout entier k, il existe un automate de Büchi $\mathcal{A}_k$ à n états tel que tout automate alternant faible à condition de parité qui accepte le même langage que $\mathcal{A}_k$ comporte au moins $k \times n$ états.

6.4 Jeux à parité

Pour démontrer le résultat de dualité des automates alternants faibles à condition de parité, il est commode de voir le problème en termes de jeux, ce qui d'ailleurs permettra plus loin de classifier certains problèmes de vérification en termes de stratégies gagnantes, puis de se ramener à des problèmes algorithmiques sur des graphes finis.

Un *jeu à parité* est un tuple $G = (Q_1, Q_2, \delta_1, \delta_2, r, q_0)$ où Q_1, Q_2 sont des ensembles d'états (pas nécessairement finis) disjoints, δ_1 est une application de $Q_1 \times \mathbb{N}$ dans 2^{Q_2} et δ_2 est une application de $Q_2 \times \mathbb{N}$ dans 2^{Q_1} telles que $\delta_k(q,i)$ n'est jamais vide. r est une fonction de *rang* de $Q_1 \cup Q_2$ dans $\mathbb{N}$. Enfin, $q_0 \in Q_1$ est l'*état initial*.

Une *configuration* du jeu G est un élément de $Q_1 \times \mathbb{N} \cup Q_2 \times \mathbb{N}$. Une *partie* est un mot infini P sur $Q_1 \cup Q_2$ tel que $w(0) = q_0$ et, si $w(i) = q \in Q_j$, alors $w(i+1) \in \delta_j(q,i)$. Le joueur 1 *gagne la partie* P si le minimum des rangs des états apparaissant infiniment souvent dans P est pair. Sinon, c'est le joueur 2 qui gagne.

Une *stratégie* (sans mémoire) s pour le joueur 1 (resp. le joueur 2) est une application de $Q_1 \times \mathbb{N}$ (resp. $Q_2 \times \mathbb{N}$) dans Q_2 (resp. Q_1) telle que $s(q,i) \in \delta_1(q,i)$ (resp. $\in \delta_2(q,i)$) pour tous q, i.

Un joueur B a une *stratégie gagnante* s'il gagne la partie en appliquant sa stratégie, quels que soient les coups de son adversaire.

Exercice 21 Considérer le jeu G décrit dans la figure 15. Les deux tours de jeu sont figurés respectivement par un carré et un rond dans lesquels sont indiqués les noms des configurations (pour faciliter l'expression des stratégies). Le rang de chaque configuration est figuré à l'extérieur de chaque configuration et les déplacements autorisés sont les arêtes du graphe.

Montrer que le joueur rond a une stratégie gagnante (sans mémoire). Que se passe-t-il si l'on remplace $r(7)$ par 2 ?

Un jeu à parité est *faible* si $r(q') \leq r(q)$ dès que $q' \in \delta_i(q,j)$ pour un certain j. Pour ces jeux particuliers on a alors :

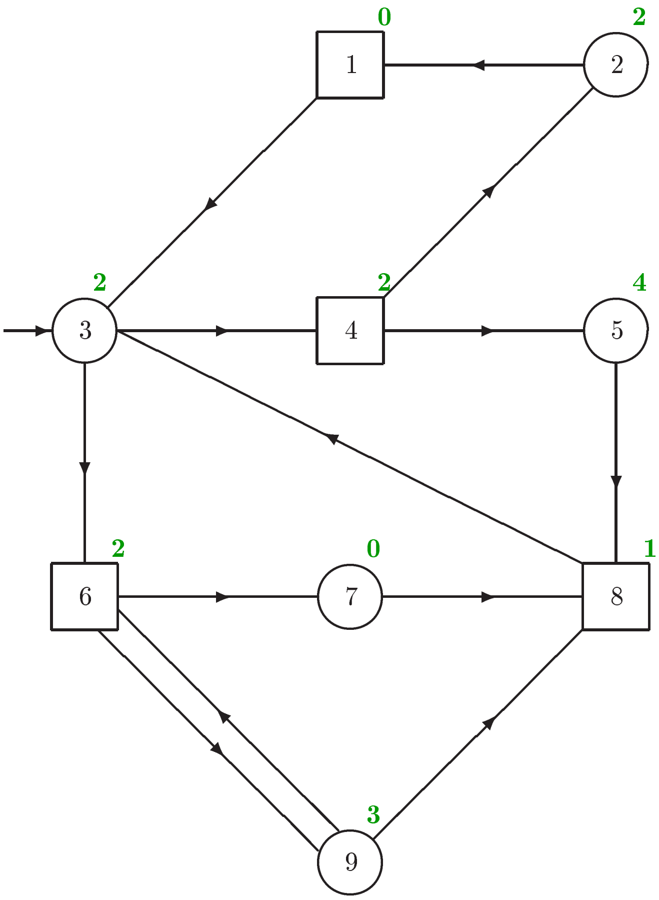

Fig. 15 – Un jeu à parité

Lemme 23 (déterminisme des jeux à parité faibles) *Pour tout jeu à parité faible, l'un des deux joueurs a une stratégie gagnante sans mémoire.*

Preuve: Soit A_i^j et B_i^j les ensembles définis par récurrence comme suit (l'intention est de définir ainsi les configurations gagnantes pour A et B respectivement en jouant au plus j coups à un rang supérieur à i) :

- $A_0^0 = \{(q,k) \in Q_1 \times \mathbb{N} \mid r(q) = 0\}$
- $B_0^0 = \emptyset$
- $$\begin{array}{rcl} A_i^{j+1} & = & A_i^j \\ & & \cup\{(q,k) \in Q_1 \times \mathbb{N} \mid A_i^j \cap \delta_1(q,k) \neq \emptyset\} \\ & & \cup\{(q,k) \in Q_2 \times \mathbb{N} \mid \delta_2(q,k) \subseteq A_i^j\} \end{array}$$
- $$\begin{array}{rcl} B_i^{j+1} & = & B_i^j \\ & & \cup\{(q,k) \in Q_1 \times \mathbb{N} \mid \delta_1(q,k) \subseteq B_i^j\} \\ & & \cup\{(q,k) \in Q_2 \times \mathbb{N} \mid \delta_2(q,k) \cap B_i^j \neq \emptyset\} \end{array}$$
- Si i est pair, alors $B_{i+1}^0 = \{(q,k) \notin \bigcup_{j=0}^{i} \bigcup_{m=0}^{+\infty} A_j^m \mid r(q) = i+1\}$. Si i est impair, $B_{i+1}^0 = \emptyset$
- Si i is est pair, alors $A_{i+1}^0 = \{(q,k) \notin \bigcup_{j=0}^{i} \bigcup_{m=0}^{+\infty} B_j^m \mid r(q) = i+1\}$. Si i est pair, alors $A_{i+1}^0 = \emptyset$.

Soit B_i (resp. A_i) l'ensemble $\bigcup_{k\leq i} \bigcup_{j=0}^{+\infty} B_k^j$ (resp. $\bigcup_{k\leq i} \bigcup_{j=0}^{+\infty} A_k^j$).

On montre par récurrence sur i que

- $A_i \cap B_i = \emptyset$,
- $A_i \cup B_i$ contient toutes les configurations de rang inférieur ou égal à i,
- si $(q,k) \in A_i$ (resp. $(q,k) \in B_i$), il existe une stratégie gagnante sans mémoire pour le joueur 1 (resp. pour le joueur 2) pour les parties commençant dans la configuration (q,k).

Pour cela, remarquons d'abord que, pour i pair, B_i^j est vide et que, pour i impair, A_i^j est vide (par récurrence sur j). Si $i = 0$, $B_0 = \emptyset$ et le résultat est prouvé : la stratégie du joueur 1 sur $(q,0)$ est arbitraire. Soit maintenant $i > 0$. Supposons que i est pair (si i est impair, il suffit d'échanger les A et les B). s est la stratégie gagnante pour le joueur A à partir d'une configuration de A_{i-1}. On étend s à A_i^j par récurrence sur j :

- Si $j = 0$, $q \in Q_1$, il exite un successeur de (q,k) qui n'est pas dans B_{i-1}, par définition de A_i^0. On choisit $s(q,k) \notin B_{i-1}$ (si $s(q,k)$ n'est pas déjà défini).
- Si $j > 0$, $q \in Q_1$ et $(q,k) \in A_i^j \setminus A_i^{j-1}$, alors, par définition, $\delta_1(q,k) \cap A_i^{j-1}$ est non vide. On choisit $s(q,k)$ dans cet ensemble (s'il n'est pas déjà défini).

Montrons par récurrence sur j que toute partie commençant dans une configuration de l'un des ensembles A_i^j, et suivant la stratégie s, est gagnante pour le joueur 1. Seul le cas de base $j = 0$ est non trivial. Si une partie commence en $(q,k) \in A_i^0$, avec $q \in Q_1$ alors, par définition de s, la configuration suivante $s(q,k)$ n'est pas dans B_{i-1}. Par faiblesse du jeu, $r(s(q,k)) \leq i$. Donc, ou bien $r(s(q,k)) = i$ et $s(q,k) \in A_i^0$, ou bien $r(s(q,k)) < i$ et, par hypothèse de récurrence, $s(q,k) \in A_{i-1}$ Si maintenant $q \in Q_2$, montrons qu'aucun successeur de (q,k) n'est dans B_{i-1}. On concluera alors comme ci-dessus que la configuration suivante d'une partie commençant par (q,k) est dans $A_i^0 \cup A_{i-1}$. Pour cela, raisonnons par l'absurde : si $(q', k+1) \in B_{i-1}$, alors, il existe $l < i, m$ tels que $(q', k+1) \in B_l^m$. Mais, par définition de B_l^{m+1}, cela implique que $(q,k) \in B_l^{m+1}$, donc $(q,k) \in B_{i-1}$, ce qui est contraire à l'hypothèse.

Nous avons donc montré que, dans tous les cas, dans une partie commençant par $(q,k) \in A_i^0$ et suivant la stratégie s, la configuration suivante est dans $A_i^0 \cup A_{i-1}$. Alors, ou bien toutes les configurations de la partie sont dans A_i^0 et, le rang de toutes les configurations de A_i^0 étant le nombre pair i, la partie est gagnée pour le joueur 1. Ou bien, il existe une configuration de la partie qui est dans A_{i-1}. Dans ce cas, par hypothèse de récurrence, la partie est aussi gagnée pour le joueur 1.

$A_i \cap B_i = \emptyset$ car il ne peut exister de stratégie gagnante pour les deux joueurs. Par ailleurs, $A_i \cup B_{i-1}$ contient, par définition de A_i^0 et par hypothèse de récurrence, toutes les configurations de rang inférieur ou égal à i.

Maintenant, $W_A = \bigcup_i A_i$ et $W_B = \bigcup_i B_i$ forment une partition de l'ensemble des configurations du jeu et, comme nous l'avons vu, si la configuration initiale est dans W_A (resp. dans W_B), c'est à dire dans l'un des A_i (resp. B_i), alors il existe une stratégie gagnante pour le joueur 1 (resp. le joueur 2). $\square$

Ce résultat se généralise aux jeux à parité, pas nécessairement faibles. Mais la preuve est beaucoup plus difficile (voir par exemple [Tho97] où en fait est montré un résultat plus général : au lieu de damiers "linéaires", les damiers sont arborescents),

6.5 Complémentation des automates alternants avec condition de parité.

Soit $\mathcal{A} = (Q, q^0, \delta, r)$ un automate alternant avec condition de parité.

Lemme 24 (complète spécification) *Si $\mathcal{A}$ est un automate alternant à parité, il existe un automate $\mathcal{A}'$ acceptant le même langage et tel que, pour tout état q et toute lettre d'entrée a, $\delta'(q,a)$ est satisfaisable. $\mathcal{A}'$ se calcule en temps linéaire.*

Preuve: On peut supposer sans perdre de généralité (i.e. au prix d'une translation du rang) que le minimum des rangs est 2. $\mathcal{A}'$ est alors l'automate $\mathcal{A}$ augmenté d'un état poubelle $q_\perp$ de rang 1 et des transitions :

- $\delta'(q_\perp, a) = q_\perp$ pour toute lettre a
- $\delta'(q,a) = q_\perp$ si $\delta(q,a)$ est non spécifié ou insatisfaisable.

Comme dès l'instant qu'on atteint l'état poubelle dans un calcul, celui-ci ne peut être réussi, l'ensemble des calculs réussis de $\mathcal{A}$ est identique à l'ensemble des calculs réussis de $\mathcal{A}'$. $\square$

On supposera donc désormais que $\delta(q,a)$ est toujours satisfaisable.

On associe à $\mathcal{A}$ et à $w \in A^\omega$ un jeu $G(\mathcal{A},w)$ avec condition de parité comme suit : $Q_1 = Q$, $Q_2 = 2^Q$, $\delta_1(q,i)$ est l'ensemble des ensembles d'états S qui satisfont $\delta(q,w(i/2))$ et $\delta_2(S,i)$ est l'ensemble S lui-même. Enfin, le rang du jeu associe à chaque état q $r(q)$ et aux ensembles d'états $+\infty$. En fait, pour simplifier dans la suite, on ne considérera que les coups du joueur 1 (évitant la division par 2) et la partie consistera en une séquence d'états.

Lemme 25 *Soit $\mathcal{A}$ un automate à parité quelconque. $\mathcal{A}$ accepte w si et seulement si le joueur 1 a une stratégie gagnante sans mémoire pour $G(\mathcal{A},w)$.*

Preuve: Supposons tout d'abord que $\mathcal{A}$ accepte w : il existe un calcul réussi ρ de $\mathcal{A}$ sur w. Chaque nœud de ce calcul est caractérisé par son étiquette $q \in Q$ et sa profondeur i. La stratégie du joueur 1 consiste alors, depuis (q,i) à choisir l'ensemble des étiquettes des fils de (q,i) dans l'arbre ρ. La stratégie n'est pas définie pour les paires (q,i) qui ne correspondent pas à des nœuds de ρ. En appliquant cette stratégie, le joueur 1 est assuré que la partie sera la séquence des étiquettes d'un chemin infini de ρ : le joueur 2 ne peut que choisir, à partir d'un nœud, le fils qui continuera la partie. Mais, comme pour tous les chemins infinis de ρ, le minimum des rangs des états apparaissant infiniment souvent est pair, le joueur 1 gagne nécessairement la partie.

Réciproquement, si s est une stratégie gagnante pour le joueur 1, on construit un calcul ρ à partir de s : si q étiquette le calcul à une profondeur i, on choisit $s(q,i)$ pour construire les successeurs de q dans ρ. $\square$

Si $\mathcal{A}$ est un automate alternant, on note $\widetilde{\mathcal{A}}$ l'*automate dual* défini comme suit : les états de $\widetilde{\mathcal{A}}$ sont les mêmes que ceux de $\mathcal{A}$, l'état initial est le même, les transitions de $\widetilde{\mathcal{A}}$ sont obtenues à partir de celles de $\mathcal{A}$ en remplaçant les conjonctions par des disjonctions et les disjonctions par des conjonctions. Enfin, la fonction de rang $\tilde{r}$ est définie par $\tilde{r}(q) = r(q) + 1$.

$\widetilde{\mathcal{A}}$ est un automate à parité faible dès que $\mathcal{A}$ est un automate à parité faible. Nous allons montrer qu'il accepte le complémentaire du langage accepté par $\mathcal{A}$ dans ce cas.

Lemme 26 *Supposons que $\mathcal{A}$ est un automate alternant avec condition de parité. Si le joueur 1 a une stratégie gagnante sans mémoire sur $G(\mathcal{A},w)$, alors le joueur 2 une stratégie gagnante sans mémoire sur $G(\widetilde{\mathcal{A}},w)$.*

Preuve: Soit s la stratégie gagnante du joueur 1. Soit $s(q,i) = S$ et $\tilde{\delta}(q,w(i)) = \phi$. Prouvons tout d'abord que, pour tout modèle R de ϕ, il existe un état $q_{i,R} \in S$ tel que $q_{i,R} \in R$. Supposons par exemple que ϕ est en forme normale conjonctive. Alors S est l'une des disjonctions d'états. Comme tout modèle de ϕ contient au moins un état par disjonction, $R \cap S \neq \emptyset$.

On définit maintenant la stratégie du joueur 2 comme suit : à partir d'une paire (R,i), le joueur 2 choisit l'état $q_{i,R}$. Avec cette stratégie, le joueur 2 s'assure que chaque configuration d'une partie sur $G(\widetilde{\mathcal{A}}, w)$ est aussi une configuration d'une partie jouée sur $G(\mathcal{A}, w)$, en suivant la stratégie s.

Alors, s étant une stratégie gagnante sur $G(\mathcal{A}, w)$, pour toute partie P sur ce jeu suivant s, le minimum m des $r(q)$ pour les états q rencontrés infiniment souvent dans P est pair. Comme toute partie P sur $G(\widetilde{\mathcal{A}}, w)$, suivant la stratégie $\tilde{s}$ du joueur 2 définie ci-dessus est aussi une partie sur $G(\mathcal{A}, w)$ suivant la stratégie du joueur 1, le minimum des $\tilde{r}(q)$ pour les états rencontrés infiniment souvent dans P est $m + 1$ qui est impair. Il s'en suit que $\tilde{s}$ est une stratégie gagnante pour le joueur 2. $\square$

Lemme 27 *Pour tout automate à parité $\mathcal{A}$ et $\widetilde{\widetilde{\mathcal{A}}}$ acceptent le même langage.*

Theorème 28 *Si $\mathcal{A}$ est un automate alternant à parité, l'automate dual $\widetilde{\mathcal{A}}$ accepte le complémentaire du langage accepté par $\mathcal{A}$.*

Preuve: D'après le lemme 25, l'automate $\mathcal{A}$ accepte w si et seulement si le joueur 1 a une stratégie gagnante sans mémoire sur $G(\mathcal{A}, w)$. D'apres le lemme 26, si le joueur 1 a une stratégie gagnante sans mémoire sur ce jeu alors le joueur 2 a une stratégie gagnante sans mémoire sur le jeu dual. D'après le lemme 25, cela implique que w n'est pas accepté par $\widetilde{\mathcal{A}}$.

Réciproquement, si w n'est pas accepté par $\widetilde{\mathcal{A}}$, comme, d'après le lemme 23, l'un des deux joueurs a une stratégie gagnante sans mémoire et que, d'après le lemme 25 ce ne peut pas être le joueur 1, c'est que le joueur 2 a une stratégie gagnante sans mémoire sur w dans le jeu dual. D'après le lemme 26, ceci implique que le joueur 1 a une stratégie gagnante dans le double dual. D'après le lemme 25, ceci entraine que w est accepté par le double dual de $\mathcal{A}$, donc par $\mathcal{A}$ d'après le lemme 27. $\square$

Exercice 22 En calculant des automates à parité (pas nécessairement faibles) pour les exemples des figures 12, 13, 14, donner dans chaque cas un automate à parité acceptant le complémentaire. Les automates obtenus sont ils faibles ?

6.6 Complémentation des automates de Büchi

Lemme 29 *Si $\mathcal{A}$ est un automate alternant faible avec condition de parité, on peut calculer en temps exponentiel un automate de Büchi qui accepte le même langage.*

Preuve: Elle est semblable à la preuve du lemme 16 à laquelle il faut se reporter pour plus de détails. Si $\mathcal{A} = (Q, q^0, \delta, r)$, l'automate de Büchi $\mathcal{A}'$ a pour ensemble d'états les paires (S, S') où S et S' sont des sous-ensembles de Q. La fonction de transition est définie par :

Si $S_1 \neq \emptyset$, $(S_1, S_1') \xrightarrow{a} (S_2, S_2')$ si :

> Pour tout $q \in S_1 \cup S_1'$, il existe un ensemble $S_q \subseteq Q$ tel que $S_q \models \delta(q, a)$ et

– $S_2 = (\bigcup_{q \in S_1} S_q) \setminus Q_0$ où Q_0 est l'ensemble des états de rang pair.
– $S'_2 = (\bigcup_{q \in S'_1} S_q) \cup ((\bigcup_{q \in S_1} S_q) \cap Q_0)$

Si $S_1 = \emptyset$, $(S_1, S'_1) \xrightarrow{a} (S_2, S'_2)$ si $S'_2 = \emptyset$ et $S_2 = \bigcup_{q \in S'_1} S_q$ où S_q est un ensemble d'états qui satisfait $\delta(a, q)$.

Les états finaux sont ceux tels que $S = \emptyset$. L'état initial est $(\{q^0\}, \emptyset)$ □

Exercice 23 Montrer par un exemple que la construction du lemme ci-dessus n'est pas correcte dans le cas où l'automate alternant à parité n'est pas faible. (On pourra considérer un automate à deux états acceptant les mots sur l'alphabet $\{0, 1\}$ qui ne contiennent qu'un nombre fini de 1).

Theorème 30 *La classe des langages reconnus par un automate de Büchi est effectivement close par complémentaire.*

Preuve: C'est une conséquence des lemmes 22, 29 et du théorème 28. □

6.7 Recherche de stratégies gagnantes sur les jeux finis

Un jeu est dit *fini* si les relations de transitions des deux joueurs (δ_1 et δ_2) ne dépendent pas du second argument.

Exercice 24 Démontrer le lemme de détermination des jeux sans la condition de faiblesse, dans le cas d'un jeu fini.

On considère dans ce paragraphe des jeux finis et on s'intéresse à la complexité de la recherche de stratégies gagnantes. Ce problème est lié à la vérification de modèles (finis) et au problème d'accessibilité dans les graphes ET/OU.

Un graphe (dirigé) ET/OU est un ensemble de sommets S, un ensemble d'arêtes $R \subseteq S \times S$, ainsi qu'une application e de S dans $\{\wedge, \vee\}$. Etant donné un graphe ET/OU G, on note $\xrightarrow[G]{*}$ la relation *d'accessibilité* définie par : $\xrightarrow[G]{*}$ est le plus petit sous-ensemble de $S \times 2^S$ tel que :

– si $s^0 \in S_1$, alors $s^0 \xrightarrow[G]{*} S_1$
– si $e(s^0) = \vee$ et il existe un sommet s^1 tel que $(s^0, s^1) \in R$ et $s^1 \xrightarrow[G]{*} S_1$, alors $s^0 \xrightarrow[G]{*} S_1$
– si $e(s^0) = \wedge$ et pour tout sommet s^1 tel que $(s^0, s^1) \in R$, $s^1 \xrightarrow[G]{*} S_1$, alors $s^0 \xrightarrow[G]{*} S_1$.

Theoreme 31 *L'accessibilité dans un graphe ET/OU est décidable en temps linéaire.*

Preuve: Etant donné un ensemble de sommets S_1, on calcule le plus petit point fixe de l'application qui à S associe l'ensemble des sommets desquels on peut accéder à S en au plus une étape. Plus précisément :

1. Initialement, $S = S_1$, $T = S_1$
2. Tant que $T \neq \emptyset$:
   - On calcule l'ensemble T des sommets s qui ne sont pas dans S et tels que
     - Ou bien $e(s) = \vee$ et l'un des successeurs de s est dans S.
     - Ou bien $e(s) = \wedge$ et tous les successeurs de s sont dans S
   - On remplace T par $T \cup S$
3. $s^0 \xrightarrow[G]{*} S_1$ si et seulement si $s^0 \in S$

Remarquons que cet algorithme naïf de calcul de point fixe est bien polynômial (quadratique). Mais on peut trouver un algorithme linéaire en étant un peu plus rusé, que nous expliquons ci-dessous.

On peut tout d'abord se ramener, en temps linéaire, au cas où chaque sommet du graphe a un degré sortant au plus deux.

A chaque sommet s du graphe est attaché un quadruplet (e(s),val(s),flag(s),ancestors(s)). $e(s) \in \{\wedge, \vee\}$, $val(s), flag(s) \in \{\textbf{true}, \textbf{false}\}$ et ancestors(s) est la liste des sommets tels qu'il existe une arête (s', s) dans le graphe. Initialement, flag(s) =**false**pour tous les sommets ayant deux successeurs et flag(s)=**false**pour les sommets ayant 0 ou 1 successeur. val(s) =**true**si et seulement si $s \in S_1$. Cette construction peut être effectuée en temps linéaire.

On maintient ensuite une piles P. Initialement, P contient les sommets de S_1. L'algorithme procède comme suit :

Tant que la pile P est non vide :

1. retirer le sommet de pile s.
2. pour chacun des sommets t de ancestors(s) faire :
   (a) Si $val(t) = \textbf{true}$, ne rien faire
   (b) Si $e(t) = \vee$ et $val(t) = \textbf{false}$, alors mettre à jour $val(t)$ à **true** et empiler t sur P
   (c) Si $e(t) = \wedge$ et $flags(t) = \textbf{true}$, alors mettre à jour $val(t)$ à **true** et empiler t sur P
   (d) Si $e(t) = \wedge$ et $flags(t) = \textbf{false}$, alors mettre à jour $flags(t)$ à **true**.

Si la pile P est vide, les sommets s desquels on peut accéder à S_1 sont ceux tels que $val(s) = \textbf{true}$ et l'algorithme s'arrête

On vérifie que l'algorithme est linéaire : chaque sommet est empilé au plus une fois et, pour chaque sommet dépilé s, le nombre de mises à jour est proportionnel au nombre d'ancêtres de s ; au total, le nombre de mises à jour est proportionnel au nombre d'arêtes.

Remarquons que, en cas d'accessibilité, on peut effectivement calculer un "chemin" en temps linéaire. Un tel chemin c associe à chaque sommet t tel que $e(t) = \vee$ un successeur à partir duquel S_1 est accessible. Dans la présentation ci-dessus, il suffit dans le cas 2b de garder $c(t) = s$.

□

En fait, le problème est complet pour PTIME, mais ce résultat est admis ici.

Theorème 32 *Supposant l'ensemble d'états fini, on peut décider quel joueur a une stratégie gagnante dans un jeu à parité faible en temps linéaire (et calculer cette stratégie).*

Preuve: On montre par récurrence sur le rang maximal d'une configuration que l'on peut calculer en temps linéaire, pour chaque configuration q, quel est le joueur qui a une stratégie gagnante à partir de q et aussi donner effectivement cette stratégie.

Si tous les états sont de même rang, le joueur 1 a une stratégie gagnante si et seulement si ce rang est pair. Dans ce cas, la stratégie gagnante est arbitraire.

Supposons maintenant que M est le rang maximal. On décompose les positions du jeu en $Q_0 \uplus \ldots \uplus Q_R$, Q_i étant l'ensemble des configurations de rang i. Le jeu étant faible la restriction des transitions aux configurations de $Q_0 \uplus \ldots \uplus Q_{M-1}$ est encore un jeu à parité faible lorsque la configuration initiale est choisie parmi ces états. Par hypothèse de récurrence, on peut supposer associé à chacun des états $q \in Q_0 \uplus \ldots \uplus Q_{M-1}$ le joueur gagnant $J(q)$ à partir de cet état et sa stratégie $S(q)$.

Considérons alors le graphe ET/OU dont les sommets sont les configurations Q_M plus deux sommets particuliers 1 et 2. Les arêtes sont données par les relations de transition : pour $s, s' \in Q_M$, $(s, s') \in R$ si et seulement si $s' \in \delta_i(s)$. De plus, si $s' \in \delta_i(s)$ et $r(s') < M$, alors on ajoute l'arête $(s, J(s'))$. $e(s)$ est $\vee$ si s est une configuration dans laquelle c'est au joueur 1 de jouer. $e(s)$ est $\wedge$ sinon. Si 1 est accessible à partir de q, alors le joueur 1 a une stratégie gagnante à partir de q. Si 2 est accessible à partir de q, dans le graphe dual (où les $\wedge$ et les $\vee$ sont échangés) alors le joueur 2 a une stratégie gagnante à partir de q. Si ni 1 ni 2 ne sont accessibles à partir de q dans G et le graphe dual respectivement, alors, ou bien M est pair et le joueur 1 a une stratégie gagnante, ou bien M est impair et le joueur 2 a une stratégie gagnante.

Comme, d'après le théorème 31, l'ensemble des sommets desquels 1 est accessible (resp. 2) peut être calculé en temps linéaire, on peut calculer $J(q)$ pour $q \in Q_M$ en temps linéaire.

De plus, la stratégie du joueur 1 (par exemple) est fournie par le chemin qui permet d'accéder à 1, dans le cas où 1 est accessible. Dans l'autre cas (1 n'est pas accessible dans le graphe, 2 n'est pas accessible dans le graphe dual et M est pair), il suffit d'associer à chaque sommet $\wedge$ du graphe dual un des sommets successeurs qui n'est pas étiqueté par **true**. □

Comme conséquence du théorème 32 et du lemme 25 :

Corollaire 33 *Le problème du vide pour les automates alternants à parité faibles sur un alphabet à une lettre peut être résolu en temps linéaire.*

Theorème 34 *Supposant l'ensemble d'états fini, déterminer si le joueur 1 a une stratégie gagnante dans un jeu à parité est dans NP $\cap$ co-NP.*

Preuve: On devine la stratégie ainsi que l'ensemble S des états de rang pair minimal rencontrés infiniment souvent. Vérifier que la stratégie est gagnante revient à décider que l'on peut atteindre S dans un graphe ET/OU, puis que l'on peut atteindre S à partir d'un état de S, dans le même graphe. Comme l'accessibilité dans un graphe ET/OU est polynômial (théorème 31), cette vérification se fait en temps polynômial. Il en résulte que le problème est dans NP. Par détermination des jeux à parité, le joueur 1 n'a pas de stratégie gagnante si et seulement si le joueur 2 a une stratégie gagnante, ce qui est dans NP à nouveau. Donc le problème est dans NP $\cap$ co-NP. $\square$

Comme conséquence du théorème 32 et du lemme 25 :

Corollaire 35 *Le problème du vide pour les automates alternants avec condition de parité sur un alphabet à une lettre est dans NP $\cap$ co-NP.*

7 Logique monadique de second ordre à k successeurs SkS

Ce paragraphe n'est qu'un rapide survol.

Syntaxe
- **true**, **false**
- $t \in X$
- $t = \epsilon$, $t = S_i(t')$, pour $i = 1, ..., k$, $t = t'$, $t \geq t'$
- $\phi \vee \psi$, $\phi \wedge \psi$, $\neg \phi$
- $\forall t.\phi$, $\exists t.\phi$
- $\exists^2 X.\phi$, $\forall^2 X.\phi$

Sémantique Domaine d'interprétation des variables du premier ordre : $\{1, ..., k\}^*$ avec les opérations : concaténation à droite par $1, ..., k$, mot vide, ordre préfixe.

Domaine d'interprétation des variables de second ordre : sous-ensembles de $\{1, ..., k\}^*$.

Exercice 25 Exprimer en SkS les prédicats suivants :

1. Pair(X) qui est satisfait par l'ensemble des positions de longueur paire et cet ensemble seulement.
2. Chemin(X) qui est satsifait par les ensembles qui représentent des chemins infinis
3. Infini(X) satisfait par les ensembles infinis de positions
4. Origine(x, X) satisfait par les chemins X dont l'orgine est x
5. Succ(X, Y) satisfait par les chemins X et Y tels que Y est X privé de son origine

Exercice 26 Codage d'un arbre quelconque à branchement fini : si un nœud possède k fils : 21, ... $2^k 1$. Plus précisément, $c(i \cdot w) = 2^i \cdot 1 \cdot c(w)$. On obtient ainsi une application dans l'arbre binaire, pour laquelle l'image du codage est expressible dans $S2S$ et le codage injectif. Si bien que $S_1, ..., S_n \models \phi$ ssi $c(S_1), ..., c(S_n) \models c(\phi)$.

De même que pour S1S, si $X_1, \ldots, X_n$ sont les variables libres (de second ordre) d'une formule ϕ de SkS, ϕ définit un langage (ensemble d'arbres) sur l'alphabet $\{0,1\}^n$: un arbre (infini) de branchement k et étiqueté par $\{0,1\}^n$ est une application de $\{1, ..., k\}^*$ dans $\{0,1\}^n$. Un arbre T satisfait ϕ si les ensembles $S_1, ...S_n$ définis par : S_i est l'ensemble des positions de T dont l'étiquette a pour ième composante 1, satisfont ϕ. On note L_ϕ le langage ainsi défini.

Exemple 36 Soit ϕ la formule

$$\begin{array}{rl} 0 \in X \wedge \forall x. & (x \in X \Rightarrow x \notin Y \wedge x \cdot 1 \in X \wedge x \cdot 2 \notin X) \\ \wedge & (x \in Y \Rightarrow x \notin X \wedge x \cdot 1 \notin Y \wedge x \cdot 2 \notin Y) \end{array}$$

Les arbres de la figure 16 satisfont ϕ.

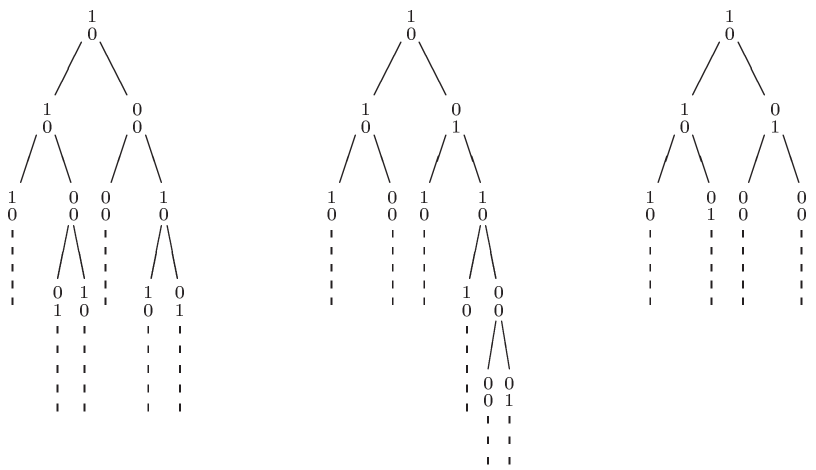

FIG. 16 – Quelques arbres binaires sur l'alphabet $\{0,1\}^2$

7.1 Automates d'arbres

Un automate d'arbres est un tuple (Q, q_0, A, δ, F) où Q est un ensemble fini d' états, $q_0 \in Q$ est l'état initial, A est un ensemble fini de symboles d'entrée, F est une condition d'acceptance (nous serons plus précis ultérieurement) et δ est un ensemble de règles de la forme :

$$q(f(x_1, \ldots, x_k)) \rightarrow f(q_1(x_1), \ldots, q_k(x_k))$$

Autrement dit :

$$\delta \subseteq Q \times F \times Q^k$$

Un *calcul* ρ de l'automate sur un arbre infini T (de branchement constant k) étiqueté par A est un arbre étiqueté par Q (i.e. une application de $\{1, \ldots, k\}^*$ dans Q tel que $\rho(\epsilon) = q_0$ et :

$$\left.\begin{array}{r} \rho(t) = q \\ \forall i = 1..k,\ \rho(t \cdot i) = q_i \\ T(t) = f \end{array}\right\} \Rightarrow \exists q(f(x_1, \ldots, x_k)) \rightarrow f(q_1(x_1), \ldots, q_k(x_k)) \in \delta$$

Exemple 37 Soit l'automate défini par : $Q = \{q, q'\}, q_0 = q$, $A = \{a, b\}$, et δ est donné par :

$$\begin{array}{rcl} q(a(x,y)) & \rightarrow & a(q(x), q(y)) \\ q'(a(x,y)) & \rightarrow & a(q(x), q(y)) \\ q(b(x,y)) & \rightarrow & b(q'(x), q'(y)) \\ q'(b(x,y)) & \rightarrow & b(q'(x), q'(y)) \end{array}$$

Un calcul de l'automate est représenté sur la figure 17.

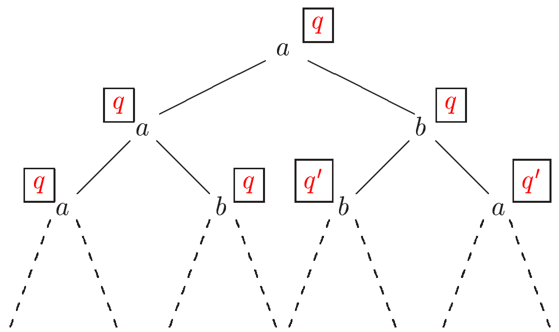

FIG. 17 – Un calcul d'automate d'arbres

La *condition d'acceptance de Büchi* est donnée par $F \subseteq Q$; un calcul de l'automate est réussi si toutes ses branches contiennent une infinité d'éléments de F. Un arbre T est accepté par l'automate $\mathcal{A}$ s'il existe au moins un calcul réussi sur T.

Exemple 38 Si l'automate de l'exemple 37 est complété par la condition d'acceptance de Büchi $F = \{q\}$, l'automate accepte les arbres dont toutes les branches contiennent une infinité de a.

Exercice 27 Donner un automate de Büchi d'arbres qui accepte les arbres binaires sur l'alphabet $\{a, b\}$ tels qu'au moins une branche contient une infinité de a.

Par contre, le complémentaire du langage de cet exemple n'est pas accepté par un automate d'arbres de Büchi : avec cette condition d'acceptance, il n'y a pas stablité par complémentaire.

La *condition d'acceptance de Rabin* est donnée par un ensemble fini de paires de sous ensembles de Q : $F = \{(L_1, U_1), \ldots, (L_n, U_n)\}$. Un calcul ρ sur T est *réussi* si, pour tout chemin π de ρ, il existe un indice i tel que π ne contienne qu'un nombre fini d'éléments de L_i et un nombre infini d'éléments de U_i.

Exemple 39 Le complémentaire du langage de l'exercice précédent, c'est-à-dire l'ensemble des arbres dont tous les chemins ne contiennent qu'un nombre fini de a est accepté par l'automate de l'exemple 37 avec la condition d'acceptance : $F = \{(\{q\}, Q)\}$.

La *condition de parité* est donnée par une application r de l'ensemble des états dans les entiers. Un arbre t est accepté s'il existe un calcul ρ sur t tel que, pour tout chemin infini π de ρ, le minimum des $r(q)$ pour les états q apparaissant infiniment souvent dans π, est pair.

Exercice 28 Construire un automate à parité acceptant le complémentaire du langage de l'exercice 27.

7.2 Les résultats de décision

Les résultats obtenus pour S1S se généralisent à SkS (au prix d'une complexité supplémentaire) :

Theorème 40 *Le vide des automates d'arbres avec condition de parité (resp. de Rabin) est dans co-NP. (En fait co-NP-complet).*

Preuve: On ramène le problème du non-vide des automates d'arbres avec condition de parité au problème du non vide des automates alternants de mots, avec condition de parité, sur un alphabet à une lettre : l'automate de mots a pour états $Q \times A$, le rang est défini par $r'(q, f) = r(q)$, les états initiaux sont (q^0, f) et la relation de transition est définie par :

$$\delta'(q, f) = \bigvee_{(q,f,(q_1,\ldots,q_k))\in\delta} \; \bigvee_{g\in A} \; \bigwedge_{i=1}^{k} (q_i, g)$$

Il existe un arbre t sur lequel l'automate d'arbres a un calcul réussi si et seulement si l'automate de mots a un calcul réussi : ce calcul donne en même temps l'arbre t qui est deviné à chaque étape.

On utilise alors le théorème 35. $\square$

Theorème 41 (Rabin, 1969) *: Un ensemble d'arbres infinis est définissable en SkS si et seulement si il est reconnu par un automate de Rabin.*

Nous ne donnons pas la preuve du théorème ici (voir par exemple l'excellent article de synthèse [Tho97]). Il faut d'abord montrer les propriétés de cloture des automates à parité. Le point le plus délicat est la clôture par complémentaire qui s'obtient, comme dans le chapitre précédent, grâce à un résultat de détermination des jeux, dans le cas général des jeux à parité.

Comme le vide est décidable pour les automates de Rabin, on obtient en corollaire :

Corollaire 42 *SkS est décidable.*

8 Logiques du temps arborescent (CTL*et CTL)

Syntaxe

formules d'états ϕ_s, ψ_s

- **true**, **false**
- P pour $P \in \mathcal{P}$
- $\phi_s \wedge \psi_s$, $\phi_s \vee \psi_s$, $\neg \phi_s$
- $A\phi_p$, $E\phi_p$

formules de chemins ϕ_p, ψ_p

- ϕ_s
- $\phi_p \wedge \psi_p$, $\phi_p \vee \psi_p$, $\neg \phi_p$
- $\mathbf{X}\phi_p$
- $\phi_p \, \mathcal{U} \psi_p$

On utilise aussi comme en LTL les abréviations $\Diamond\phi \stackrel{\text{def}}{=} \mathbf{true}\, \mathcal{U}\phi$ et $\Box\phi \stackrel{\text{def}}{=} \neg(\mathbf{true}\, \mathcal{U}\neg\phi)$.

Sémantique Soit $\mathcal{K}$ une structure de Kripke. $T_{\mathcal{K}}$ est le dépliage de $\mathcal{K}$: c'est un arbre infini étiqueté par les positions de $\mathcal{K}$. Un *chemin* d'origine s de $T_{\mathcal{K}}$ est une suite de positions de $\mathcal{K}$: c'est une éxecution de $\mathcal{K}$ à partir de s.

$\mathcal{K} \models \phi_s$ ssi $T, \epsilon \models \phi_s$.

formules d'états ϕ_s, ψ_s

- $T, t \models$ **true**, et $T, t \not\models$ **false**
- $T, t \models P$ ssi $P \in T(t)$
- $T, t \models \phi_s \wedge \psi_s$ ssi $T, t \models \phi_s$ et $T, t \models \psi_s$, etc...
- $T, t \models A\phi_p$, ssi pour tout chemin π de T d'origine t, $\pi, 0 \models_t \phi_p$
- $T, t \models E\phi_p$ ssi il existe un chemin π de T d'origine t tel que $\pi, 0 \models_t \phi_p$

formules de chemins ϕ_p, ψ_p

- $\pi, n \models_{t_0} \phi_s$ ssi $T, t \models \phi_s$ où $t = t_0 \cdot t_1$ et t_1 est le préfixe de π de longueur n.
- $\pi, t \models_{t_0} \phi_p \wedge \psi_p$ ssi $\pi, t \models_{t_0} \phi_p$ et $\pi, t \models_{t_0} \psi_p$, etc ...
- $\pi, t \models_{t_0} \mathbf{X}\phi_p$ ssi $\pi, t+1 \models_{t_0} \phi_p$
- $\pi, t \models_{t_0} \phi_p \, \mathcal{U} \psi_p$ ssi il existe $t' \geq t$ tel que
  - $\pi, t' \models_{t_0} \psi_p$
  - pour tout $t'', t \leq t' < t''$, $\pi, t'' \models_{t_0} \phi_p$

Exercice 29 Soit $\mathcal{K}$ la structure de Kripke donnée dans la figure 18. Représenter visuellement son dépliage et dire (en le justifiant) si $\mathcal{K}$ vérifie la formule $A\Diamond A\Box P$.

CTL*est au moins aussi expressive que LTL qu'elle contient (ce sont les formules de chemins) mais il y a nombre de propriétés qui sont expressibles en CTL*et pas en LTL. Par exemple les propriétés de vivacité comme

$$A\Box(P \Rightarrow E\Diamond Q)$$

"Quand P est vrai, il est possible d'obtenir Q ensuite".

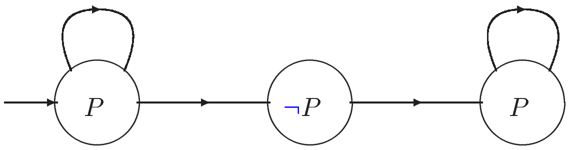

FIG. 18 – Une structure de Kripke

8.1 CTL

CTL est le sous-ensemble de CTL*dans lequel les formules de chemins $\mathbf{X}\phi_p$ et $\phi_p\,\mathcal{U}\psi_p$ sont immédiatement précédées d'un quantificateur sur les chemins A ou E.

Exercice 30 Montrer que $\Box\Diamond P$ n'est pas expessible en CTL.

8.2 Traduction en SkS

On considère seulement ici les structures de Kripke à branchement borné par un nombre fixe k. Nous verrons (lemme 49) que ce n'est pas une restriction. Par abus de notation, si K est une structure de Kripke, on note encore $t_\mathcal{K}$ le dépliage de K dans lequel les étiquettes $s \in S$ sont remplacées par $f(s) \in 2^\mathcal{P}$. On ordonne de plus arbitrairement les variables propositionnelles : $\mathcal{P} = \{P_1, \ldots, P_n\}$.

Theorème 43 *Pour toute structure de Kripke de branchement au plus k et toute formule ϕ de* CTL**, il existe une formule $\widetilde{\phi}(t, X_1, \ldots, X_n)$ de SkS (effectivement calculable en temps linéaire) telle que $\mathcal{K} \models \phi$ si et seulement si $\epsilon, t_\mathcal{K} \models \widetilde{\phi}(t, X_1, \ldots, X_n)$*

Preuve: On peut toujours coder un nombre arbitraire (fini) de successeurs à l'aide de deux successeurs, l'image étant définissable en S2S. On se limite ainsi à deux successeurs.

On code les formules d'états comme des formules de S2S ayant une variable libre de premier ordre t et les formules de chemins comme des formules ayant une variable libre de second ordre C. ($X_1, \ldots, X_n$ qui sont libres dans toutes les formules sont sous-entendus).

On utilise aussi les prédicats Origine,Suc et Chaîne, qui sont définissables en S2S, comme nous l'avons vu en exercice.

formules d'états ϕ_s, ψ_s

- $\widetilde{\mathbf{true}}(t) \stackrel{\text{def}}{=} \mathbf{true}$, $\widetilde{\mathbf{false}}(t) \stackrel{\text{def}}{=} \mathbf{false}$
- $\widetilde{P_i}(t) \stackrel{\text{def}}{=} t \in X_i$
- $\widetilde{\phi_s \wedge \psi_s}(t) \stackrel{\text{def}}{=} \widetilde{\phi_s}(t) \wedge \widetilde{\psi_s}(t)$, etc...
- $\widetilde{A\phi_p}(t) \stackrel{\text{def}}{=} \forall C.\text{Chaîne}(C, t) \Rightarrow \widetilde{\phi_p}(C)$, etc...

formules de chemins ϕ_p, ψ_p

- $\widetilde{\phi_s}(C) \stackrel{\text{def}}{=} \exists t.\text{Origine}(t, C) \wedge \widetilde{\phi_s}(t)$
- $\widetilde{\phi_p \wedge \psi_p}(C) \stackrel{\text{def}}{=} \widetilde{\phi_p}(C) \wedge \widetilde{\psi_p}(C)$, etc ...
- $\widetilde{\mathbf{X}\phi_p}(C) \stackrel{\text{def}}{=} \exists C'.\text{Suc}(C, C') \wedge \widetilde{\phi_p}(C')$
- $\widetilde{\phi_p \mathcal{U} \psi_p}(C) \stackrel{\text{def}}{=} \exists t. t \in C \wedge (\exists C'.\text{Origine}(t, C') \wedge \widetilde{\psi_p}(C') \wedge \forall t' \in C, \exists C''. t' \geq t \vee (\text{Origine}(t', C'') \wedge \widetilde{\phi_p}(C'')))$

La vérification de l'équivalence est alors routinière. $\square$

9 Automates d'arbres alternants

La notion clef cette fois ci est celle des automates d'arbres alternants (ou plus exactement des sous-classes de celle-ci). Nous introduisons et étudions les automates d'arbres alternants faibles (WAA) et leur algorithmique. Nous présentons ensuite tout ou partie des algorithmes optimaux de model-checking qui en découle :

– - CTL (polynomial)
– - mu-calcul sans alternation (polynomial)
– - CTL* (PSPACE-complet)
– - mu-calcul (NP inter co-NP)

Pourquoi la complexité pourrait-elle être réduite par rapport au problème de vérification en LTL, alors que les automates sont plus complexes ? En bref, c'est parce qu'on teste l'appartenance et non l'inclusion.

9.1 Automates alternants d'arbres ; différentes conditions d'acceptance

Les arbres considérés ici sont infinis et leur branchement est variable (éventuellement infini), l'ordre des fils n'étant pas significatif. Pour être plus précis, on considère un ensemble D (par exemple l'ensemble des entiers naturels). Un arbre T (sous-entendu à branchements dans D) étiqueté par A est une application d'un ensemble $Pos(T) \subseteq D^*$ clos par préfixe (appelé *ensemble des positions de* T) dans A. Si $t \in Pos(t)$, on note $s_T(t)$ l'ensemble des successeurs de t dans T : $s_T(t) \stackrel{\text{def}}{=} \{d \mid d \in D, t \cdot d \in Pos(T)\}$.

Un automate d'arbres alternant $\mathcal{A}$ est un tuple (A, Q, δ, q_0, F) où Q est ensemble fini d'états, $q_0 \in Q$ est l'état initial, A est l'alphabet d'entrée, supposé fini. $\delta : Q \times A \to \mathcal{B}^+(\{\Box, \Diamond\} \times Q)$.

La taille d'un automate alternant est (comme pour les mots) égale au nombre d'états + taille de l'alphabet + taille du système de transition. Cette dernière est définie comme la somme, pour chaque transition qui ne va pas à **false**de la taille de la formule résultat.

Un *calcul* de l'automate sur un arbre T est un arbre T_r dont la racine est étiquetée par q_0 et tous les autres nœuds sont étiquetés par un élément de $Q \times Pos(T)$. De plus, si $T_r(u) = (q, t), T(t) = a$, alors $\delta(q, a)$ est défini et, il existe un ensemble (éventuellement vide) $S = \{(\Box, q_1), \ldots, (\Box, q_k), (\Diamond, q'_1), \ldots, (\Diamond, q'_m)\}$ tel que S satisfait $\delta(q, a)$ et

– pour tout $d \in s_T(t)$, pour tout $i = 1, ...k$, il existe un $j \in s_{T_r}(u)$ tel que $T_r(u \cdot j) = (q_i, t \cdot d)$
– pour tout $i = 1, ..., m$, il exite un $d_i \in s_T(t)$, il existe un $j \in s_{T_r}(u)$ tel que $T_r(u \cdot j) = (q'_i, t \cdot d_i)$

Un calcul est *réussi* s'il satisfait les conditions définies par F (ce peut être des conditions de Büchi ou de Rabin par exemple).

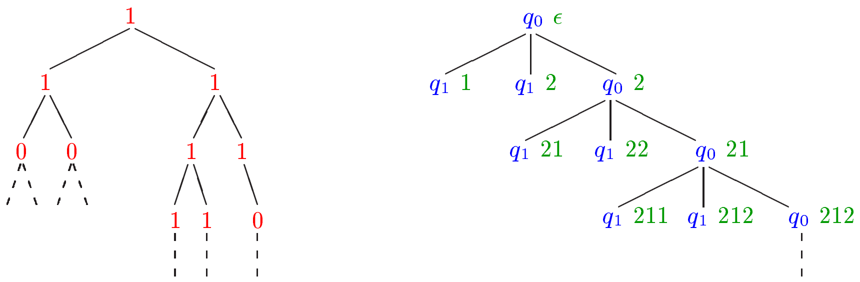

FIG. 19 – Un exemple de calcul d'un automate alternant d'arbres

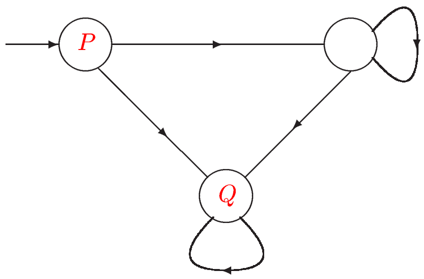

FIG. 20 – Une structure de Kripke

Exemple 44 Soit δ la relation de transition définie par la table :

δ	0	1
q_1	**false**	**true**
q_0	**true**	$(\Box, q_1) \wedge (\Diamond, q_0)$

L'état initial étant q_0, un arbre et un calcul de l'automate sont représentés sur la figure 19.

Exercice 31 Soit l'automate d'arbres alternants dont les transitions sont données par la table :

	$\begin{matrix}0\\0\end{matrix}$	$\begin{matrix}0\\1\end{matrix}$	$\begin{matrix}1\\0\end{matrix}$	$\begin{matrix}1\\1\end{matrix}$
q_0	**true**	**true**	$((\Box, q_2) \vee (\Diamond, q_1)) \wedge (\Box, q_0)$	**true**
q_1	$(\Box, q_2)$	$(\Box, q_2)$	$(\Box, q_2) \vee (\Diamond, q_1)$	**true**
q_2	**false**	**true**	**false**	**true**

L'état initial est q_0. Donner un calcul de l'automate sur le dépliage de la structure de Kripke représentée sur la figure 20.

La *condition d'acceptance de Büchi* est donnée par un sous-ensemble Q_f de Q. Un calcul ρ est alors réussi si tout chemin infini de ρ contient une infinité d'éléments de Q_f.

La *condition d'acceptance de Rabin* est donnée par un ensemble de paires de sous-ensembles de Q : $F = \{(L_1, U_1), \ldots, (L_n, U_n)\}$. Un calcul ρ est alors réussi si, pour tout chemin infini π de ρ, il existe une paire $(L_\pi, U_\pi) \in F$ telle que $\pi \cap L_\pi$ est fini et $\pi \cap U_\pi$ est infini.

9.1.1 Automates alternants faibles

Un automate alternant d'arbres est *faible* s'il a une condition d'acceptance de Büchi $Q_f \subseteq Q$ et, de plus, il existe une partition de Q en ensembles $Q_1, \ldots, Q_n$ telle que, pour tout i, ou bien $Q_i \subseteq Q_f$ (et dans ce cas Q_i est un ensemble acceptant), ou bien $Q_i \cap Q_f = \emptyset$ (et dans ce cas c'est un ensemble de rejet). De plus, il existe un ordre partiel sur les ensembles Q_i tel que, si $q \in Q_i$ et $q' \in Q_j$ et q' apparait dans une des formules $\delta(q, a, k)$, alors $Q_j \leq Q_i$.

Exemple 45 Les automates des exemples 15, 18, 44 sont tous faibles. L'automate des exemples 37 et 38 n'est pas faible.

9.2 CTL et automates d'arbres alternants

Theorème 46 *Etant donnés une formule ϕ de CTL, on peut construire en temps linéaire un automate alternant faible qui accepte exactement l'ensemble des arbres qui satisfont ϕ.*

Preuve: On suppose tout d'abord, sans perte de généralité, que les négations sont poussées dans ϕ à l'intérieur des formules suivant les règles :

$$\begin{array}{rcl}
\neg\neg\psi & \to & \psi \\
\neg(\psi_1 \vee \psi_2) & \to & (\neg\psi_1) \wedge (\neg\psi_2) \\
\neg(\psi_1 \wedge \psi_2) & \to & (\neg\psi_1) \vee (\neg\psi_2) \\
\neg(A\mathbf{X}\psi) & \to & E\mathbf{X}(\neg\psi) \\
\neg(E\mathbf{X}\psi) & \to & A\mathbf{X}(\neg\psi) \\
\neg\mathbf{true} & \to & \mathbf{false} \\
\neg\mathbf{false} & \to & \mathbf{true}
\end{array}$$

Cette normalisation peut être effectuée en temps linéaire.

Si bien que les négations n'apparaissent que devant les variables propositionnelles et devant $A\,\mathcal{U}$ et $E\,\mathcal{U}$.

Les états de l'automate $\mathcal{A}_\phi$ sont alors les sous-formules de ϕ ainsi que les négations des sous-formules de la forme $A\ \mathcal{U}$ ou $E\ \mathcal{U}$. le nombre d'états est linéaire en la taille de ϕ. L'état initial est ϕ, les états finaux sont les formules de la forme $\neg A(\phi_1\ \mathcal{U}\phi_2)$ ou $\neg E(\phi_1\ \mathcal{U}\phi_2)$. La fonction de transition est définie par :

$$
\begin{array}{rcll}
\delta(P,a) & \stackrel{\mathrm{def}}{=} & \textbf{true} & \text{Si } P \in a \\
 & & \textbf{false} & \text{Sinon} \\
\delta(\neg\phi,a) & \stackrel{\mathrm{def}}{=} & \overline{\neg\delta(\phi,a)} & \\
\delta(\phi_1 \wedge \phi_2,a) & \stackrel{\mathrm{def}}{=} & \delta(\phi_1,a)\wedge\delta(\phi_2,a) & \\
\delta(\phi_1 \vee \phi_2,a) & \stackrel{\mathrm{def}}{=} & \delta(\phi_1,a)\vee\delta(\phi_2,a) & \\
\delta(A\mathbf{X}\phi_1,a) & \stackrel{\mathrm{def}}{=} & (\Box,\phi_1) & \\
\delta(E\mathbf{X}\phi_1,a) & \stackrel{\mathrm{def}}{=} & (\Diamond,\phi_1) & \\
\delta(A\phi_1\,\mathcal{U}\phi_2,a) & \stackrel{\mathrm{def}}{=} & \delta(\phi_2,a)\vee(\delta(\phi_1,a)\wedge(\Box,A\phi_1\,\mathcal{U}\phi_2)) & \\
\delta(E\phi_1\,\mathcal{U}\phi_2,a) & \stackrel{\mathrm{def}}{=} & \delta(\phi_2,a)\vee(\delta(\phi_1,a)\wedge(\Diamond,E\phi_1\,\mathcal{U}\phi_2)) &
\end{array}
$$

où $\overline{\phi}$ est la normalisation en poussant les négations : on utilise le système ci-dessus ainsi que les règles

$$
\begin{array}{rcl}
\neg(\Box,\psi) & \rightarrow & (\Diamond,\psi) \\
\neg(\Diamond,\psi) & \rightarrow & (\Box,\psi)
\end{array}
$$

La preuve que cet automate accepte bien les arbres qui satisfont ϕ est laissée au lecteur ; il s'agit d'une double récurrence, comme dans la preuve du théorème 17. L'automate est faible : il suffit de choisr pour partition l'ensemble des singletons $\{q\}$ et pour ordre $\phi_1 \geq \phi_2$ si ϕ_2 est une sous-formule de ϕ_1 ou bien la négation d'une sous-formule stricte de ϕ_1. On peut vérifier qu'il s'agit bien d'une relation d'ordre sur les états. $\Box$

Exemple 47 Soit ϕ la formule $\neg A(P\,\mathcal{U}(A\mathbf{X}\neg P))$. La fonction de transition associée est donnée par la table :

δ	0	1
ϕ	$(\Diamond,P)$	$((\Diamond,P)) \wedge ((\Diamond,\phi))$
P	**false**	**true**

Exemple 48 La relation de transition de l'exercice 31 correspondent à la formule $A[(E(P\,\mathcal{U}A\mathbf{X}Q))\,\mathcal{U}\neg P]$.

9.3 Satisfaisabilité de formules CTL

On peut tout d'abord se ramener aux structures à branchement borné :

Lemme 49 ([ES84]) *Soit d le nombre d'occurrences de E dans la formule ϕ. ϕ est satisfaisable ssi il existe une structure de Kripke $\mathcal{K}$ dont le branchement maximal est d et telle que $\mathcal{K} \models \phi$.*

Preuve: Supposons que $\mathcal{K} = (S,R,s^0,f,\mathcal{P})$ est une structure de Kripke arbitraire qui satisfait ϕ. Soient $\phi_1,\ldots,\phi_d$ les sous-formules de ϕ de la forme $E\psi$

(pour tout i, $\phi_i = E\psi_i$). On construit une structure de Kripke de branchement d qui satisfait ϕ comme suit : pour chaque $s \in S$, soient $\phi_{1,s}, \ldots, \phi_{k,s}$ les formules existentielles qui sont satisfaites par $(S, R, s, f, \mathcal{P})$. Pour chacune de ces formules $\phi_{s,i}$, il existe au moins un sommet s_i tel que $(S, R, s_i, f, \mathcal{P}) \models \psi_i$. Par définition, $(s, s') \in R'$ si et seulement si s' est l'un des s_i. On définit alors $\mathcal{K}' = (S, R', s^0, f, \mathcal{P})$.

Par construction $\mathcal{K}'$ est de branchement au plus d. De plus, par récurrence sur la taille de ψ, sous-formule de ϕ, pour tout $s \in S$, $(S, R', s, f, \mathcal{P}) \models \psi$ si et seulement si $(S, R, s, f, \mathcal{P}) \models \psi$. □

Ainsi, grâce au théorème 46,

> ϕ est satisfaisable ssi l'ensemble des abrres à branchement au plus d reconnus par l'automate $\mathcal{A}_\phi$ est non vide.

Il en résulte que la satisfaisabilité se ramène, en temps linéaire, au vide d'un automate alternant d'arbres faible sur un domaine d'arbres de branchements $D = [1..d]$.

On procède alors de manière analogue à LTL :

Lemme 50 *Pour tout automate alternant d'arbres $\mathcal{A}$ avec condition de Büchi, on peut calculer en temps déterministe exponentiel un automate de Büchi d'arbres $\mathcal{A}'$ tel que $L(\mathcal{A}) = L(\mathcal{A}')$.*

Preuve: Elle est analogue à la preuve du lemme 16.

Remarquons que ce lemme n'utilise pas la faiblesse de l'automate.

Lemme 51 *Le vide des automates de Büchi d'arbres à branchement borné est PTIME-complet.*

Preuve: Comme dans la preuve du théorème 40, on ramène en temps plolynomial le problème du non-vide au problème du non-vide vide pour les automates alternants de mots sur un alphabet à une lettre, avec condition d'acceptance de Büchi. Ce problème se ramène à son tour à déterminer s'il existe une stratégie gagnante pour le joueur 1 dans un graphe fini G, avec condition de victoire de Büchi.

Pour résoudre ce dernier problème, on calcule le plus grand sous-ensemble S de Q_f tel que de toute configuration de S on peut atteindre S dans le graphe G. Ce plus grand point fixe est calculé par raffinements successifs :

1. Initialement, $S_0 = \emptyset$ et $S_1 = Q_f$
2. Tant que $S_1 \neq S_0$
   (a) Remplacer S_0 par S_1
   (b) calculer le sous-ensemble S_1 de S_0 des configurations à partir desquelles on peut atteindre S_0 dans G
3. S est la valeur commune de S_0 et S_1

Le joueur 1 a une stratégie gagnante si et seulement si S est accessible dans G à partir de l'état initial.

La boucle est parcourue au plus $|Q_f|$ fois et chaque étape utilise au plus $O(|Q_f|)$ calculs d'accessibilité dans un graphe ET/OU, ce qui se fait en temps linéaire d'après le théoreme 31. On obtient ainsi en fin de compte un algorithme cubique (on peut faire mieux en utilisant une méthode un peu moins naïve). $\square$

Pour une autre preuve de ce lemme ainsi que des références, voir [Tho90].

Theorème 52 *La satisfaisabilité de formules de CTL est un problème DEXPTIME-complet.*

Preuve: Nous admettons ici la difficulté pour DEXPTIME. Concernant l'appartenance à DEXPTIME, il suffit d'enchainer les résultats qui précèdent : lemme 49, théorème 46, lemme 50, lemme 51. Noter que le branchement borné par $O(|\phi|)$ est utilisé pour construire le graphe fini dans la preuve du lemme 51. $\square$

9.4 Principes du model-checking en CTL

Etant donnée une structure de Kripke finie $\mathcal{K}$, on lui associe son dépliage $T_{\mathcal{K}}$ dont les nœuds, au lieu d'être étiquetés par des positions de $\mathcal{K}$, sont étiquetés par les propositions qui sont vraies à ces positions (on remplace s par $f(s)$). Grâce au théorème 46, on obtient l'équivalence :

$\mathcal{K} \models \phi$ si et seulement si $T_{\mathcal{K}} \in L(\mathcal{A})$

Autrement dit, le model-checking se ramène à tester l'appartenance d'un arbre infini à un langage défini par un automate d'arbres alternant faible. Ceci doit être mis en parallèle avec la vérification de modèles pour les formules de LTL : il s'agissait alors de tester l'inclusion de l'ensemble des executions de $\mathcal{K}$ dans le langage défini par $\mathcal{A}_\phi$. Or l'appartenance est un problème a priori plus simple que l'inclusion. Et, en effet, si le model-checking de LTL est PSPACE-complet, le model-checking de CTL est dans PTIME, comme nous allons le voir.

Pour tester l'appartenance de $T_{\mathcal{K}}$ à L, on procède de la manière suivante :

1. construire un automate qui accepte un et un seul arbre : $T_{\mathcal{K}}$
2. construire un automate qui accepte l'intersection de l'automate précédent et de l'automate $\mathcal{A}_\phi$.
3. Tester le vide de l'automate intersection.

Remarquons que l'automate intersection accepte 0 ou 1 arbre : il n'accepte aucun arbre si $\mathcal{K}$ ne satisfait pas ϕ, et il en accepte exactement un sinon.

Le calcul de l'intersection de deux automates est peu complexe (complexité= produit des complexités de chacun des deux automates). Pour obtenir le résultat voulu, il suffit donc de savoir effectuer la dernière opération (test du vide) en temps linéaire. Arrivé à ce point, nous utiliserons trois propriétés de l'automate intersection :

– c'est un automate faible (ce qu'il faudra prouver)
– L'alphabet peut être ignoré, puisqu'on ne s'intéresse qu'à l'existence d'un arbre accepté, sans s'occuper de l'étiquetage des nœuds
– Lorsque l'alphabet est réduit à une lettre, automates d'arbres alternants et automates de mots alternants sont équivalents : les copies de l'arbres envoyées aux états ne sont pas pertinentes. On peut donc considérer qu'il s'agit d'un automate de mots.

Et nous avons vu (corollaire 33) que ce dernier problème peut être résolu en temps linéaire.

9.5 L'automate intersection

Soit $\mathcal{K} = (S, R, s^0, f, \mathcal{P})$ une structure de Kripke, d son branchement maximal, ϕ une formule de CTL sur les variables propositionnelles $\mathcal{P}$. On définit comme suit l'automate alternant $\mathcal{A}_{\mathcal{K},\phi}$.

Soit $\mathcal{A}_\phi = (2^{\mathcal{P}}, Q, \delta_\phi, q_0, Q_f)$ l'automate du théorème 46. $\mathcal{A}_{\mathcal{K},\phi}$ est un automate alternant de mots sur un alphabet à une lettre (lettre désormais omise) dont les états sont $Q \times S$, l'état initial est $< q_0, s^0 >$, la relation de transition $\delta_{\mathcal{K},\phi}$ est définie par :

$$\delta_{\mathcal{K},\phi}(< q, s >) = \delta_\phi(q, f(s))\{(\Box, q_i) \mapsto \bigwedge_{j=1}^{d(s)} < q_i, s_j >,\ (\Diamond, q_i) \mapsto \bigvee_{j=1}^{d(s)} < q_i, s_j >\}$$

$d(s)$ étant le branchement à la position s : on suppose que $\{s' \mid sRs'\} = \{s_1, \ldots, s_{d(s)}\}$ c'est à dire que les successeurs de s dans $\mathcal{K}$ sont numérotés.

La condition d'acceptance de $\mathcal{A}_{\mathcal{K},\phi}$ est une condition de parité telle que l'automate soit faible ; le rang est défini par :

$$r(< \phi, s >) = \begin{cases} 2 \times d(\phi) \text{ si } \phi = \neg(E\phi_1\, \mathcal{U}\phi_2) \text{ ou } \neg(A\phi_1\, \mathcal{U}\phi_2) \\ 2 \times d(\phi) + 1 \text{ Sinon} \end{cases}$$

où $d(\phi)$ est le nombre d'imbrications de négations de ϕ. (On peut aussi prendre la profondeur ou la taille de ϕ). La faiblesse de l'automate est assurée par la faiblesse de $\mathcal{A}_\phi$.

Exemple 53 Reprenons l'exemple 47, avec la structure de Kripke donnée dans la figure 21.

Les transitions (pertinentes) de l'automate intersection sont données par :

δ	
$< \phi, s^0 >$	$(< P, s^1 > \vee < P, s^2 >) \wedge (< \phi, s^1 > \vee < \phi, s^2 >)$
$< \phi, s^1 >$	$< P, s^1 > \wedge < \phi, s^1 >$
$< \phi, s^2 >$	$< P, s^1 > \vee < P, s^2 >$
$< P, s^1 >$	**true**
$< P, s^2 >$	**false**

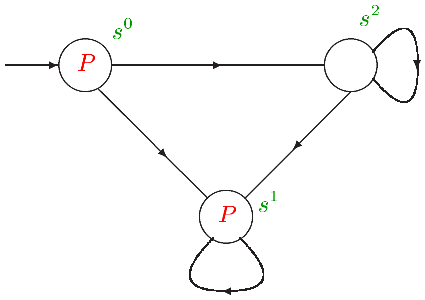

FIG. 21 – Une structure de Kripke

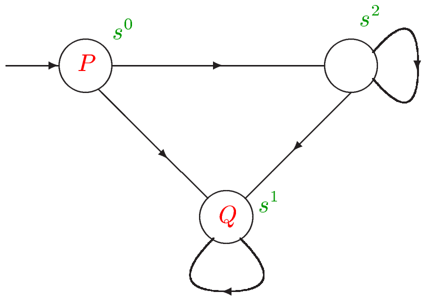

FIG. 22 – Une structure de Kripke

Exercice 32 Calculer l'automate intersection en considérant las tructure de Kripke donnée figure 22 et la formule de l'exemple 48.

Theorème 54 $\mathcal{K} \models \phi$ *si et seulement si* $L(\mathcal{A}_{\mathcal{K},\phi}) \neq \emptyset$.

Preuve: Supposons tout d'abord que $T_{\mathcal{K}} \in L(\mathcal{A}_\phi)$. Il existe donc un calcul réussi T_r de l'automate sur $T_{\mathcal{K}}$. On construit un calcul ρ de $\mathcal{A}_{\mathcal{K},\phi}$ sur 1^ω (1 est la seule lettre de l'alphabet) de la manière suivante : ρ a même ensemble de positions que T_r et, si $T_r(p) = (q, w)$, alors $\rho(p) =< q, s_w >$, s_w étant défini par récurrence sur la longueur de w par : $s_\epsilon = s^0$, $s_{w\cdot i}$ est le $i$ème successeur de s_w dans $\mathcal{K}$.

Comme T_r est un calcul sur $T_{\mathcal{K}}$, si $T_r(p) = (q, w)$, $T_{\mathcal{K}}(w) = a(= f(s_w))$, alors il existe un ensemble $S = \{(\Box, q_1), \ldots, (\Box, q_n), (\Diamond, q'_1), \ldots, (\Diamond, q'_m)\}$ tel que $S \models \delta_\phi(q, a)$. L'ensemble S' obtenu en remplaçant chaque $(\Box, q_i)$ par $< q_i, s_1 >, \ldots, < q_i, s_{d(s)} >$ et chaque $(\Diamond, q_i)$ par un $< q_i, s_{j_i} >$ tel que $T_r(p\cdot l) = (q_i, w\cdot j_i)$, satisfait $\delta_{\mathcal{K},\phi}(< q, s_w >)$ par définition. ρ est donc un calcul de $\mathcal{A}_{\mathcal{K},\phi}$ sur 1^ω. De plus, comme T_r est un calcul réussi (et que $\mathcal{A}_\phi$ est faible), pour tout chemin infini γ de T_r, il existe une formule ψ_γ de la forme $\neg(A\phi_1 \mathcal{U} \phi_2)$ ou $\neg(E\phi_1 \mathcal{U} \phi_2)$ telle que, pour n suffisamment grand, $\gamma(n) = (\psi_\gamma, w_n)$. Il en résulte que, pour tout chemin infini γ' de ρ, pour n suffisamment grand, $r(\gamma'(n)) = 2 \times d(\psi_\gamma)$ est pair. Donc ρ est un calcul réussi, et donc $L(\mathcal{A}_{\mathcal{K},\phi}) \neq \emptyset$.

Réciproquement, supposons que $L(\mathcal{A}_{\mathcal{K},\phi})$ est non vide : soit ρ un calcul réussi. On construit un calcul T_r sur $T_{\mathcal{K}}$ comme suit. Les positions de T_r sont les mêmes que celles de ρ. $T_r(\epsilon) = (q_0, \epsilon)$ et, si $\rho(p\cdot i) =< q, s >$ et $T_r(p) = (q', w)$, alors $T_r(p \cdot i) = (q, w \cdot j)$ si s est le $j$ème successeur de s_w dans $\mathcal{K}$.

Pour toute position p de T_r, si $T_r(p) = (q, w)$, alors, par récurrence sur $|p|$, w est une position de $T_{\mathcal{K}}$. Soit $a = T_{\mathcal{K}}(w) = f(s_w)$. Par définition, $\rho(p) =< q, s_w >$ et, si $p\cdot 1, \ldots p\cdot n$ sont les successeurs de la position p dans ρ et $\rho(p\cdot i) =< q_i, s_{w\cdot j_i} >$ avec $j_i \in \{1..k\}$ où k est le nombre de successeurs de s_w, alors $\{< q_1, s_{w\cdot j_1} >, \ldots, < q_n, s_{w\cdot j_n} >\}$ satisfait $\delta_{\mathcal{K},\phi}(< q, s_w >)$ (par définition d'un calcul). Maintenant, par définition de $\delta_{\mathcal{K},\phi}$,

$$S = \begin{array}{ll} & \{(\Box, q) \mid \forall i \in \{1, ...k\}, \exists j_i \in \{1, ..., n\}, q = q_{i_j}\} \\ \cup & \{(\Diamond, q) \mid \exists i \in \{1, ..., k\}, \exists j_i \in \{1, ..., n\}, q = q_{i_j}\} \end{array}$$

satisfait $\delta_\phi(q, a, k)$, ce qui montre que T_r est un calcul. Il est réussi car ρ est réussi. $\Box$

9.6 Conséquence pour la vérification en CTL

Theorème 55 *Etant donnés* $\mathcal{K}$ *et* $\phi \in CTL$, *on peut décider en temps* $O(|\mathcal{K}| \times |\phi|)$ *si* $\mathcal{K} \models \phi$.

Preuve: $\mathcal{K} \models \phi$ se ramène au vide de $\mathcal{A}_{\mathcal{K},\phi}$ qui est construit en temps $O(|\mathcal{K}| \times |\phi|)$. Cet automate est un automate alternant à parité, faible, sur un alphabet à une lettre. Donc, d'après le corollaire 33, le vide peut en être décidé en temps linéaire, d'où un algorithme de vérification linéaire à la fois en la taille de la spécification et en la taille du modèle. □

Exemple 56 Poursuivons l'exemple 53 ; on obtient la fonction de transition puis sa simplification après marquage :

δ	
q_0	$q_{0,1} \wedge q_{0,2}$
$q_{0,1}$	$q_4 \vee q_5$
$q_{0,2}$	$q_2 \vee q_3$
q_2	$q_2 \wedge q_4$
q_3	$q_4 \vee q_5$
q_4	**true**
q_5	**false**

q_0	**true**
$q_{0,1}$	**true**
$q_{0,2}$	**true**
q_2	q_2
q_3	**true**
q_4	**true**
q_5	**false**

L'état initial q_0 étant déjà marqué, on peut conclure sans aller plus avant. On peut aussi remplacer q_2 par **true** puisqu'il est final.

Exercice 33 Poursuivre l'exercice 32 en montrant comment l'algorithme de décision de la satisfaction fonctionne sur cet exemple.

10 Vérification de modèles pour le μ-calcul

Nous ne présentons ici qu'une version simple du μ-calcul propositionnel introduit par D. Kozen [Koz83]. On pourra consulter par exemple l'ouvrage de synthèse [AN01] pour une introduction plus systématique. Nous adoptons ici les idées et la présentation de [Wil01].

10.1 Syntaxe

On suppose donné un ensemble de variables propositionnelles $\mathcal{P}$. Les négations sont supposées poussées jusqu'aux propositions. L'ensemble des formules du μ-calcul est le plus petit ensemble tel que, si ϕ, ϕ_1, ϕ_2 sont des formules et $P \in \mathcal{P}$ alors

$\mathbf{true}, \mathbf{false}, P, \neg P$

y (variable)

$\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$

$AX\phi, EX\phi$

$\mu y.\phi, \nu y.\phi$

sont des formules.

10.2 Sémantique

$\mathcal{K}$ est une structure de Kripke d'ensemble S de configurations, ϕ est une formule comportant n variables libres $y_1, \ldots, y_n$ σ est une substitution $\{y_i \mapsto S_i \mid i \in \{1, \ldots, n\},\ S_i \subseteq S\}$.

$[\![\phi]\!]_\sigma^\mathcal{K}$ est défini par :

$[\![\mathbf{true}]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} S$

$[\![\mathbf{false}]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} \emptyset$

$[\![y]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} \sigma(y)$

$[\![\phi \wedge \psi]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} [\![\phi]\!]_\sigma^\mathcal{K} \cap [\![\phi]\!]_\sigma^\mathcal{K}$

$[\![\phi \vee \psi]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} [\![\phi]\!]_\sigma^\mathcal{K} \cup [\![\phi]\!]_\sigma^\mathcal{K}$

$[\![AX\phi]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} \{s \in S \mid \forall s \to s',\ s' \in [\![\phi]\!]_\sigma^\mathcal{K}\}$

$[\![EX\phi]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} \{s \in S \mid \exists s \to s',\ s' \in [\![\phi]\!]_\sigma^\mathcal{K}\}$

$$[\![\mu y.\phi]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} \bigcap_{\substack{S_1 \subseteq S \\ [\![\phi]\!]_{\sigma \cup \{y \mapsto S_1\}}^\mathcal{K} \subseteq S_1}} S_1$$

$$[\![\nu y.\phi]\!]_\sigma^\mathcal{K} \stackrel{\text{def}}{=} \bigcup_{\substack{S_1 \subseteq S \\ [\![\phi]\!]_{\sigma \cup \{y \mapsto S_1\}}^\mathcal{K} \supseteq S_1}} S_1$$

Si ϕ est une formule close, alors $\mathcal{K} \models \phi$ ssi $s^0 \in [\![\phi]\!]_\emptyset^{\mathcal{K}}$.

Exercice 34 Montrer que les définitions sémantiques de $\mu y.\phi$ et $\nu u.\phi$ correspondent bien (respectivement) au plus petit et au plus grand point fixe :

$$\begin{array}{ll} [\![\mu y.\phi]\!]_\sigma^{\mathcal{K}} & = [\![\phi\{y \mapsto \mu y.\phi\}]\!]_\sigma^{\mathcal{K}} \\ [\![\nu y.\phi]\!]_\sigma^{\mathcal{K}} & = [\![\phi\{y \mapsto \nu y.\phi\}]\!]_\sigma^{\mathcal{K}} \end{array}$$

10.3 Approximations des points fixes

Les plus petits et plus grands points fixes peuvent être obtenus par approximations successives, ce qui donne un moyen effectif simple de les calculer dans le cas d'une structure de Kripke finie. Dans le cas général, il faut définir des approximants ordinaux : si α est un ordinal, ϕ est une formule dont les variables libres sont contenues dans $\{x, x_1, \ldots, x_k\}$ et σ est une affectation de $x_1, \ldots, x_k$ à des ensembles $S_1, \ldots, S_k$ de positions, on définit S_α et T_α par récurrence bien fondée

- $S_0 = \emptyset$, $T_0 = S$ (ensemble de toutes les positions de $\mathcal{K}$)
- $S_{\alpha+1} = [\![\phi]\!]_{\sigma\cup\{x\mapsto S_\alpha\}}^{\mathcal{K}}$, $T_{\alpha+1} = [\![\phi]\!]_{\sigma\cup\{x\mapsto T_\alpha\}}^{\mathcal{K}}$,
- $S_\alpha = \bigcup_{\alpha'<\alpha} S_{\alpha'}$, $T_\alpha = \bigcap_{\alpha'<\alpha} T_{\alpha'}$ si α est un ordinal limite.

Il existe alors un ordinal λ tel que $S_\lambda = [\![\mu x.\phi]\!]_\sigma^{\mathcal{K}}$ (resp. $T_\lambda = [\![\nu x.\phi]\!]_\sigma^{\mathcal{K}}$). En effet, il est facile de voir que $S_\alpha \subseteq S_{\alpha'}$ (resp. $T_{\alpha'} \subseteq T_\alpha$) si $\alpha \leq \alpha'$. Comme ces ensembles sont des sous-ensembles de S, ensemble des positions de $\mathcal{K}$, il existe un ordinal λ tel que $S_\lambda = S_{\lambda+1}$ (cf. les fondements de la théorie des ensembles, par exemple [Sho77] et les références qui y sont données).

Exercice 35 Soit $\mathcal{K}$ une structure de Kripke et S_1 un sous-ensemble des configurations de $\mathcal{K}$. Montrer que

$$S_1 \subseteq [\![\phi]\!]_{\sigma\cup\{x\mapsto S_1\}}^{\mathcal{K}} \quad \Rightarrow \quad S_1 \subseteq [\![\nu x.\phi]\!]_\sigma^{\mathcal{K}}$$

Montrer par un contre-exemple que l'on n'a pas nécessairement $S_1 \subseteq [\![\mu x.\phi]\!]_\sigma^{kripke}$.

Exemple 57 Soit ϕ la formule $\mu y.(Q\vee(P\wedge EXy))$ et $\mathcal{K}$ la structure de Kripke donnée dans la figure 23.

$[\![\phi_1]\!]^{\mathcal{K}}$ est calculé en dépliant les définitions de point fixe : calculer tout d'abord $V_1 \stackrel{\text{def}}{=} [\![Q\vee(P\wedge EXy)]\!]_{y\mapsto\emptyset}^{\mathcal{K}}$, puis $V_2 \stackrel{\text{def}}{=} [\![Q\vee(P\wedge EXy)]\!]_{y\mapsto V_1}^{\mathcal{K}}$, ...

En partant de l'ensemble vide, on obtient d'abord $\{s^1, s^3\}$, puis $\{s^0, s^1, s^3\}$ et ensuite on atteint un point fixe : $[\![\phi_1]\!]^{\mathcal{K}} = \{s^0, s^1, s^3\}$.

Soit ϕ_2 la formule $\nu y.(Q\wedge(P\vee AXy))$. Pour calculer l'interprétation de ϕ_2 sur la structure donnée dans la figure 23, on procède de la même manière que précédemment, excepté qu'on commence par l'affectation de y à S, parce qu'il s'agit d'un plus grand point fixe.

On trouve finalement $[\![\phi_2]\!]_\sigma^{\mathcal{K}} = \{s^1, s^3\}$.

Exercice 36 Soient $\phi_1 \stackrel{\text{def}}{=} \mu x.\nu y.((P\wedge EXy)\vee EXx)$ et $\phi_2 \stackrel{\text{def}}{=} \nu x.\mu y.((P\wedge EXy)\vee EXx)$. Calculer $[\![\phi_1]\!]^{\mathcal{K}}$ et $[\![\phi_2]\!]^{\mathcal{K}}$ sur la structure de Kripke $\mathcal{K}$ donnée dans la figure 24.

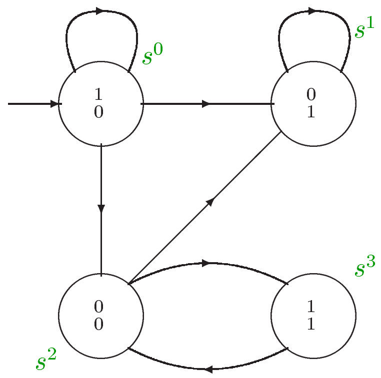

FIG. 23 – Une structure de Kripke

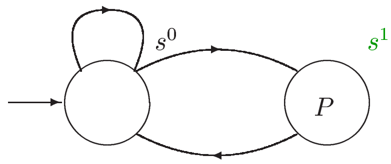

FIG. 24 – Une structure de Kripke

Exercice 37 A-t-on, pour toute formule ϕ et toute structure de Kripke $\mathcal{K}$, $[\![\mu x.\nu y.\phi]\!]^{\mathcal{K}} \subseteq [\![\nu y.\mu x.\phi]\!]^{\mathcal{K}}$? Justifier.

10.4 Expressivité

Remarquons que le μ-calcul est plus expressif que CTL*. Par exemple $EP\,\mathcal{U}Q$ s'exprime par $\mu y.Q \vee (EXy \wedge P)$. Dans l'autre sens, la propriété "P peut être atteint après un nombre pair de transitions" s'exprime en μ-calcul par $\mu y.P \vee EXEXy$ et n'a pas d'équivalent en CTL*.

En fait le μ-calcul est presque aussi expressif que SkS : si l'on ne s'autorise pas à distinguer entre les différents successeurs (i.e. on oublie leur numérotation), alors on obtient le théorème [**?**] : *les formules de SkS qui sont invariantes par bisimilarité sont expressibles en SkS.* Montrer ce résultat est en dehors de la portée de ces notes, mais nous pouvons préciser son énoncé et donner une traduction du μ-calcul en SkS.

10.5 Profondeur d'alternance

On suppose, par renommage, que chaque variable est liée au plus une fois dans la formule. $M(\phi)$ est alors l'ensemble des variables liées pas un μ et $N(\phi)$ est l'ensemble des variables liées par un ν. La *profondeur d'alternance* $a(\phi)$ d'une formule du μ-calcul est définie récursivement par :
- $a(\phi) = A(\phi, \emptyset, -1)$
- $A(\textbf{true}, E, i) = A(\textbf{false}, E, i) = A(P, E, i) = A(\neg P, E, i) = 0$
- $A(x, E, i) = E(x)$ si $E(x)$ est défini, $A(x, E, i) = 0$ sinon.
- $A(\phi_1 \wedge \phi_2, E, i) = A(\phi_1 \vee \phi_2, E, i) = \max(A(\phi_1, E, i), A(\phi_2, E, i))$
- $A(EX\phi_1, E, i) = A(AX\phi_1, E, i) = A(\phi_1, E, i)$
- $A(\mu x.\phi, E, i) = A(\phi, E \cup \{x \mapsto 0\}, 0)$ si $i = 0, -1$
- $A(\mu x.\phi, E, 1) = A(\phi, E', 0)$ où $E'(y) = E(y) + 1$ si $y \neq x$ et $E'(x) = 0$.
- $A(\nu x.\phi, E, i) = A(\phi, E \cup \{x \mapsto 0\}, 1)$ si $i = 0, -1$
- $A(\nu x.\phi, E, 0) = A(\phi, E', 1)$ où $E'(y) = E(y) + 1$ si $y \neq x$ et $E'(x) = 0$.

Dans cette définition, ϕ peut le cas échéant contenir des variables libres.

Le μ-calcul *sans alternance* est le fragment du μ-calcul pour lequel $a(\phi) = 0$.

Exercice 38 Donner la profondeur d'alternance des formules suivantes :

1. $\mu y.(Q \vee (P \wedge EXy))$
2. $\mu x.\nu y.((P \wedge EXy) \vee EXx)$
3. $\mu x.EXx \wedge (\mu z.P \vee AXz \vee (\nu y.AXy \vee Q))$
4. $\nu y.\mu x.(P \wedge y) \vee EXx$
5. $\nu x.\mu y.(Q \vee (P \wedge EXy))$

10.6 Traduction du μ-calcul en SmS

Soit $\phi(y_1, \ldots, y_n)$ une formule du μ-calcul de variables libres $y_1, \ldots, y_n$ et $\mathcal{K} = (s^0, S, \to, f)$ une structure de Kripke. Comme d'habitude la structure de Kripke est identifiée à son dépliage $t_{\mathcal{K}}$: $x_1, \ldots, x_k$ sont les ensembles de positions auxquelles $P_1, \ldots, P_k$ sont satisfaits respectivement. Z est l'ensemble des positions de $t_{\mathcal{K}}$. On suppose que le branchement de $\mathcal{K}$ est borné par m.

On définit une formule $\hat{\phi}(y_1, \ldots, y_n, y)$ telle que $[\![\phi]\!]_\sigma^{\mathcal{K}} = S$ si et seulement si $\sigma, S \models_{SmS} \hat{\phi}$, avec l'interprétation adéquate de $X_1, \ldots, X_n, Z$.

$$\widehat{\mathbf{true}}(y) \stackrel{\text{def}}{=} \mathbf{true},\ \widehat{\mathbf{false}}(y) \stackrel{\text{def}}{=} \mathbf{false},$$

$$\widehat{P}(y) \stackrel{\text{def}}{=} y = x_P,\ \widehat{\neg P}(y) \stackrel{\text{def}}{=} \forall x.(x \in y \Leftrightarrow x \notin x_P)$$

$$\widehat{y_1}(y) \stackrel{\text{def}}{=} y = y_1$$

$$\widehat{\phi \wedge \psi}(y_1, \ldots, y_n, y) \stackrel{\text{def}}{=} \exists z_1, z_2.\hat{\phi}(y_1, \ldots, y_n, z_1) \wedge \hat{\phi}(y_1, \ldots, y_n, z_2) \wedge y = z_1 \cap z_2$$

$$\widehat{\phi \vee \psi}(y_1, \ldots, y_n, y) \stackrel{\text{def}}{=} \exists z_1, z_2.\hat{\phi}(y_1, \ldots, y_n, z_1) \wedge \hat{\phi}(y_1, \ldots, y_n, z_2) \wedge y = z_1 \cup z_2$$

$$\widehat{EX\phi}(y_1, \ldots, y_n, y) \stackrel{\text{def}}{=} \exists z.\hat{\phi}(y_1, \ldots, y_n, z) \wedge \forall x(x \in y \Leftrightarrow (x \in Z \wedge \bigvee_{i=1}^{m} x \cdot i \in z))$$

$$\widehat{AX\phi}(y_1, \ldots, y_n, y) \stackrel{\text{def}}{=} \exists z.\hat{\phi}(y_1, \ldots, y_n, z) \wedge \forall x(x \in y \Leftrightarrow (x \in Z \wedge \bigwedge_{i=1}^{m} (x \cdot i \in Z \Rightarrow x \cdot i \in z)))$$

$$\widehat{\mu x.\phi}(y_1, \ldots, y_n, y) \stackrel{\text{def}}{=} \hat{\phi}(y_1, \ldots, y_n, y, y) \wedge y \subseteq Z \wedge \forall z.(z \subseteq Z \wedge \hat{\phi}(y_1, \ldots, y_n, z, z) \Rightarrow y \subseteq z)$$

$$\widehat{\nu x.\phi}(y_1, \ldots, y_n, y) \stackrel{\text{def}}{=} \hat{\phi}(y_1, \ldots, y_n, y, y) \wedge y \subseteq Z \wedge \forall z.(z \subseteq Z \wedge \hat{\phi}(y_1, \ldots, y_n, z, z) \Rightarrow z \subseteq y)$$

10.7 μ-calcul et automates alternants

Tout d'abord, on étend légèrement la définition des automates alternants en autorisant les ϵ-transitions : en plus des transitions habituelles, on peut désormais avoir des transitions $\delta(q, \epsilon) \in \mathcal{B}^+(Q)$. Dans le cas où il existe une ϵ-transition à partir de q, on fait l'hypothèse qu'il n'existe aucune autre transition à partir de q.

La définition d'un calcul de l'automate est alors légèrement modifiée également : si $T_r(p) = (q, w)$ et que l'automate effectue une ϵ-transition à partir de q, alors il existe un ensemble $S = \{q_1, \ldots, q_n\}$ d'états tels que $S \models \delta(q, \epsilon)$ et p a $|S|$ successeurs $p \cdot 1, \ldots, p \cdot n$ dans T_r, et $T_r(p \cdot i) = (q_i, w)$ (le même w : on ne progresse pas dans la lecture de l'arbre).

On peut éviter cette modification de définition, mais au prix d'une part d'une réduction aux formules gardées du μ-calcul et d'autre part d'une étape supplémentaire dans la construction des automates.

Theorème 58 *Etant donnée une formule ϕ du μ-calcul, on peut calculer en temps $O(|\phi|)$ un automate alternant d'arbres avec condition de parité $\mathcal{A}_\phi$ qui accepte les modèles de ϕ.*

Preuve:

Les états de l'automate sont les sous-formules de ϕ. Pour distinger formules et états de l'automate, ces derniers seront écrits $< \phi >$.

On suppose, sans perdre de généralité, que chaque variable est liée exactement une fois dans ϕ : pour tout x apparaissant liée dans ϕ, on note ϕ_x la sous-formule de ϕ de la forme $\mu x.\phi_1$ ou $\nu x.\phi_1$.

La relation de transition est définie par :

$$\delta(< \mathbf{true} >, a) \stackrel{\text{def}}{=} \mathbf{true},\ \delta(< \mathbf{false} >, a) \stackrel{\text{def}}{=} \mathbf{false}$$

$$\delta(< P >, a) \stackrel{\text{def}}{=} \begin{cases} \mathbf{true} & \text{Si } P \in a \\ \mathbf{false} & \text{Sinon} \end{cases}$$

$$\delta(< \neg P >, a) \stackrel{\text{def}}{=} \begin{cases} \mathbf{true} & \text{Si } P \notin a \\ \mathbf{false} & \text{Sinon} \end{cases}$$

$$\delta(< x >, \epsilon) \stackrel{\text{def}}{=} < \phi_x >$$

$$\delta(< \phi_1 \wedge \phi_2 >, \epsilon) \stackrel{\text{def}}{=} < \phi_1 > \wedge < \phi_2 >$$

$$\delta(< \phi_1 \vee \phi_2 >, \epsilon) \stackrel{\text{def}}{=} < \phi_1 > \vee < \phi_2 >$$

$$\delta(< AX\phi >, a) \stackrel{\text{def}}{=} (\Box, \phi)$$

$$\delta(< EX\phi >, a) \stackrel{\text{def}}{=} (\Diamond, \phi)$$

$$\delta(< \mu y.\phi >, \epsilon) \stackrel{\text{def}}{=} < \phi >$$

$$\delta(< \nu y.\phi >, \epsilon) \stackrel{\text{def}}{=} < \phi >$$

On définit maintenant le *rang* d'un état de la façon suivante : (intuitivement, le rang correspond à la profondeur d'alternation dans ϕ) :

$$\begin{array}{rcll} r(< \mu x.\psi >) & \stackrel{\text{def}}{=} & 2 \times (a(\phi) - a(\mu x.\psi)) + 1 & \\ r(< \psi >) & \stackrel{\text{def}}{=} & 2 \times (a(\phi) - a(\psi)) & \text{Pour toutes les autres formules} \end{array}$$

La condition d'acceptance est la condition de parité (sur toute branche d'un calcul le rang minimal d'un état apparaissant infiniment souvent est pair).

Montrons que l'automate $\mathcal{A}_\phi$ accepte bien les modèles de ϕ.

Supposons d'abord que ρ est un calcul réussi de $\mathcal{A}_\phi$ sur $t_{\mathcal{K}}$.

Soit , pour toute sous-formule ψ de ϕ, $S(\psi, \rho)$ l'ensemble des $s \in S$ telles qu'il existe une position p de s dans $t_{\mathcal{K}}$ et une position de ρ étiquetée par (ψ, p).

On considère alors la substitution σ_ρ telle que $\sigma_\rho(x) = S(\phi_x, \rho)$, pour toute variable x.

On montre, par récurrence sur la taille de ψ que

$$S(\psi, \rho) \subseteq [\![\psi]\!]^{\mathcal{K}}_{\sigma_\rho}$$

Lorsque $\psi \in \{P, \neg P, \mathbf{true}, \mathbf{false}\}$, la propriété découle des définitions.

Si $\psi = \psi_1 \wedge \psi_2$, par construction de δ, pour toute position p telle que $\rho(p) = (< \psi_1 \wedge \psi_2 >, w)$, il existe deux successeurs de p dans ρ étiquetés par $(< \psi_1 >, w)$ et $(< \psi_2 >, w)$ respectivement. Donc, si $s \in S(\psi, \rho)$, $s = t_{\mathcal{K}}\ (w)$, alors $s \in S(\psi_1, \rho) \cap S(\psi_2, \rho)$ et, par hypothèse de récurrence $S(\psi_i, \rho) \subseteq [\![\psi_i]\!]_{\sigma_\rho}$. Comme $[\![\psi_1 \wedge \psi_2]\!]^{\mathcal{K}}_{\sigma_{\rho'}}$ est, par définition, $[\![\psi_1]\!]^{\mathcal{K}}_{\sigma_{\rho'}} \cap [\![\psi_2]\!]^{\mathcal{K}}_{\sigma_{\rho'}}$, on obtient l'inclusion voulue.

Si $\psi = \psi_1 \vee \psi_2$, de la même manière que ci-dessus,

$$S(\psi_1 \vee \psi_2, \rho) \subseteq S(\psi_1, \rho) \cup S(\psi_2, \rho)$$

et, par hypothèse de récurrence $S(\psi_i, \rho) \subseteq [\![\psi_i]\!]_{\sigma_{\rho'}}$. Comme $[\![\psi_1 \vee \phi_2]\!]^{\mathcal{K}}_{\sigma_\rho}$ est, par définition, $[\![\psi_1]\!]^{\mathcal{K}}_{\sigma_\rho} \cup [\![\psi_2]\!]^{\mathcal{K}}_{\sigma_\rho}$, on obtient l'inclusion voulue.

Si $\psi = AX\psi_1$ et $p \in S(\psi, \rho)$, $\rho(p) = (< \psi >, w)$, alors, par définition, pour tout successeur $s_i = t_{\mathcal{K}}\ (w \cdot i)$ de $s = t_{\mathcal{K}}\ (w)$ dans $\mathcal{K}$, il existe un successeur p_{j_i} de p dans ρ tel que $\rho(p \cdot j_i) = (< \psi_1 >, w \cdot i)$. Par hypothèse de récurrence $S(\psi_1, \rho) \subseteq [\![\psi_1]\!]^{\mathcal{K}}_{\sigma_\rho}$ et donc, pour tout i, $s_i \in [\![\psi_1]\!]^{\mathcal{K}}_{\sigma_\rho}$. Alors, par définition de $[\![AX\psi_1]\!]^{\mathcal{K}}_i\ on \sigma_\rho$, $s \in [\![AX\psi_1]\!]^{\mathcal{K}}_{\sigma_\rho}$, ce qui permet de conclure.

Si $\psi = EX\psi_1$, le raisonnement est analogue au précédent.

Si ψ est une variable x, alors $[\![x]\!]^{\mathcal{K}}_{\sigma_\rho} = S(\phi_x, \rho)$ par définition. Or, par construction de l'automate, $S(x, \rho) \subseteq S(\phi_x, \rho)$.

Si $\psi = \mu x.\phi$ et $s \in S(\psi, \rho)$, $\rho(p) = (< \psi >, w)$, $t_{\mathcal{K}}\ (w) = s$, considérons alors l'arbre $\rho|_p$ dans lequel les branches sont coupées aux variables libres de ψ. Formellement, soit E le plus grand sous-ensemble des positions de $\rho|_p$ qui soit clos par préfixe et tel que, pour tout $\pi \in E$, il existe une sous-formule θ de ψ et un mot w' tels que $\rho(p \cdot \pi) = (< \theta >, w')$. Soit t le sous-arbre de $\rho|_p$ dont les positions sont l'ensemble E. Par construction, les feuilles de t sont étiquetées par des variables z qui sont libres dans ψ.

Le calcul ρ étant réussi et le rang de ψ étant impair et minimal dans les étiquettes de t, t ne contient pas de chemin infini.

Soit encore E_ψ l'ensemble des noeuds de t étiquetés par $(< \psi >, w')$ pour un certain w'. On définit alors E_α, où α est un ordinal, par récurrence bien fondée :

- $E_0 = \emptyset$
- $E_{\alpha+1}$ est l'ensemble des positions π de E_ψ dont tous les descendants ou bien sont dans $\bigcup_{\alpha' \leq \alpha} E_{\alpha'}$ ou bien n'appartiennent pas à E_ψ
- $E_\alpha = \bigcup_{\alpha' < \alpha} E_{\alpha'}$ si α est un ordinal limite.

Montrons que p est dans l'un des ensembles E_α (pour un certain ordinal α). ** to be completed **

On montre maintenant le résultat par récurrence sur l'ordinal α : si σ est la restriction de σ_ρ aux variables libres de ψ, par récurrence sur α, p est dans l'ensemble S_α défini dans le paragraphe 10.3.

Si $\psi = \nu x.\theta$, comme, par hypothèse de récurrence, $S(\theta, \rho) \subseteq [\![\theta]\!]^{\mathcal{K}}_{\sigma_\rho}$ et que, par définition, $[\![\psi]\!]^{\mathcal{K}}_{\sigma_\rho}$ contient tous les ensembles S_1 tels que $S_1 \subseteq [\![\theta]\!]^{\mathcal{K}}_{\sigma' \cup \{x \mapsto S_1}$, $S(\theta, \rho) \subseteq [\![\psi]\!]^{\mathcal{K}}_{\sigma_\rho}$ et l'inclusion voulue découle de la définition des calculs.

En conclusion, on a toujours $S(\psi,\rho) \subseteq [\![\psi]\!]^{\mathcal{K}}_{\sigma_\rho}$. En particulier, si on considère la formule ϕ elle-même, $s^0 \in S(\phi,\rho) \subseteq [\![\phi]\!]^{\mathcal{K}}$ et donc $\mathcal{K} \models \phi$.

Réciproquement, supposons que $\mathcal{K} \models \phi$.

Nous allons construire un calcul réussi de l'automate, en gardant l'invariant suivant : si $\rho(p) = (<\psi>, w)$, alors $\mathcal{K}, t_{\mathcal{K}}(w), \sigma_\phi \models \psi$. ** to be completed ** $\square$

Exemple 59 Reprenons l'exemple 57 : ϕ est la formule $\mu y.(Q \vee (P \wedge EXy))$. On note encore ψ la sous-formule $Q \vee (P \wedge EXy)$ L'automate associé est défini par :

$$
\begin{array}{llllll}
<\phi> & \rightarrow & <\psi> & <\psi> & \rightarrow & <Q> \vee (<P> \wedge <EXy>) \\
<Q> & \xrightarrow{\genfrac{}{}{0pt}{}{0}{1},\genfrac{}{}{0pt}{}{1}{1}} & \textbf{true} & <Q> & \xrightarrow{\genfrac{}{}{0pt}{}{1}{0},\genfrac{}{}{0pt}{}{0}{0}} & \textbf{false} \\
<P> & \xrightarrow{\genfrac{}{}{0pt}{}{1}{0},\genfrac{}{}{0pt}{}{1}{1}} & \textbf{true} & <P> & \xrightarrow{\genfrac{}{}{0pt}{}{0}{1},\genfrac{}{}{0pt}{}{0}{0}} & \textbf{false} \\
<EXy> & \xrightarrow{\genfrac{}{}{0pt}{}{0}{0},\genfrac{}{}{0pt}{}{0}{1},\genfrac{}{}{0pt}{}{1}{0},\genfrac{}{}{0pt}{}{1}{1}} & (\Diamond, <y>) & <y> & \rightarrow & <\psi>
\end{array}
$$

Le rang de toutes les formules sauf ϕ est 0. $r(\phi) = 1$.

Exercice 39 Soit $\mathcal{K}$ la structure de Kripke donnée dans la figure 23. Construire deux calculs de l'automate de l'exemple ci-dessus sur $t_{\mathcal{K}}$, l'un réussi, l'autre non.

Exercice 40 Calculer l'automate correspondant à la formule $\mu x.x$. Quels sont les arbres acceptés par cet automate ?

Exercice 41 Soit $\mathcal{K}$ la structure de Kripke dont les positions sont $\{s^0\} \cup \{t_i, i \in \mathbb{N}\}$ et les transitions : $s^0 \rightarrow t_i$ pour tout i et $t_{i+1} \rightarrow t_i$ pour tout i. Soit enfin f définie par : $f(s^0) = \emptyset$, $f(t_0) = \{P, Q\}$ et $f(t_i) = \{Q\}$ pour $i \geq 1$.

Calculer $[\![\mu x.((P \vee EXx) \wedge (Q \vee AXx))]\!]^{\mathcal{K}}$. S'il existe un calcul réussi de l'automate $\mathcal{A}_\phi$ sur $\mathcal{K}$, l'exhiber.

10.8 Vérification de modèles en μ-calcul

Theorème 60 *Etant donnés $\mathcal{K}$ et ϕ, décider si $\mathcal{K} \models \phi$ est dans NP$\cap$co-NP.*

Preuve: On peut tout d'abord calculer l'automate $\mathcal{A}_\phi$ en temps linéaire (théorème 58).

On construit ensuite l'automate produit comme suit :

- l'ensemble des états est $Q \times S$
- l'état initial est $<q_0, s^0>$
- le rang est défini par $r(<q,s>) = r_\phi(q)$
- $\delta(<q,s>) = \delta_\phi\{q' \mapsto <q', s>\}$ si δ_ϕ effectue une ϵ-transition à partir de q
- $\delta(<q,s>) = \delta_\phi(q, f(s))\{(\square, q') \mapsto \bigwedge_{s \rightarrow s'} <q', s'>, \; (\Diamond, q') \mapsto \bigvee_{s \rightarrow s'} <q', s'>\}$

$\mathcal{K} \models \phi$ si et seulement si l'automate produit (un automate alternant de mots sur un alphabet à une lettre) accepte au moins un mot.

Or, d'après le corollaire 35, ce problème est dans NP$\cap$ co-NP. (C'est le problème de déterminer si le joueur 1 a une stratégie gagnante dans un jeu à parité fini).

Exercice 42 Soit ϕ une formule du μ-calcul sans alternance, montrer par un exemple que l'automate à parité construit comme dans le théorème 58 n'est pas nécessairement un automate faible.

Montrer qu'on peut associer à toute formule ϕ du μ-calcul sans alternance un automate à parité *faible* qui accepte les dépliages des structures de Kripke qui satisfont ϕ.

En déduire que la vérification de modèles pour le μ-calcul sans alternance est PTIME-complète.

10.9 Satisfaisabilité du μ-calcul

La satisfaisabilité des formules du μ-calcul se ramène au problème du vide des automates alternants d'arbres (avec condition de parité), d'après le théorème 58.

A nouveau, ce problème se ramène à la recherche du joueur ayant une stratégie gagnante dans un jeu à parité. Un tel jeu peut être construit en temps exponentiel et comporte $O(2^{P(|A|)})$ positions où P est un polynôme et le rang maximal d'une position est $Q(|A|)$ où Q est un polynôme (voir par exemple [Wil01]). Il reste à regarder plus finement les algorithmes de décision de l'existence de stratégies gagnantes dans le cas des jeux à parité finis. On peut en fait décider qui a une stratégie gagnante en temps $O(|G|^p)$ où p est le rang maximal d'une position (voir par exemple [Jur00] pour des bornes plus fines). Il en résulte un algorithme de décision en $O(2^{P(|A|)\times Q(|A|)})$:

Theorème 61 *La satisfaisabilité du μ-calcul est dans DEXPTIME.*