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 6 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 $(\{0\}, \{0\})$.

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 $a \in A$.

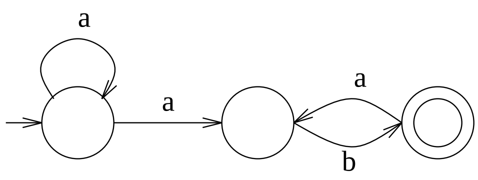

FIG. 3 – Automate de Büchi acceptant $a^*(ab)^\omega$

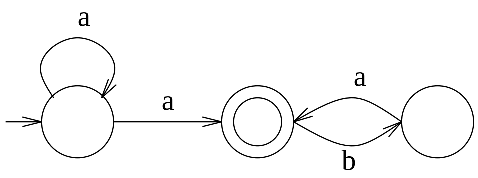

FIG. 4 – Un autre automate acceptant $a^*(ab)^\omega$

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 $ababababab\ldots$. 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 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

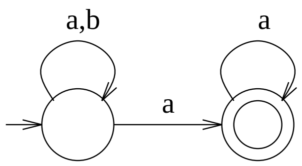

FIG. 5 – Un autre automate acceptant $(a+b)^*a^\omega$

(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 7 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.

$\square$

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

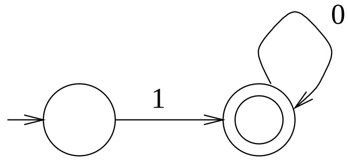

FIG. 6 – Automate acceptant le langage défini par $X = 0$

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 :

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

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) :

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

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, 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

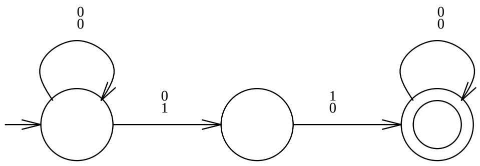

FIG. 7 – Automate acceptant le langage défini par $X = S(Y)$

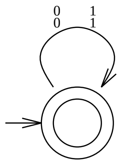

FIG. 8 – Automate acceptant le langage défini par $X = Y$

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 $\neg\exists X,\ \neg\phi$.

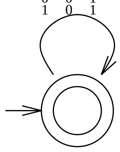

FIG. 9 – Automate acceptant le langage défini par $X \subseteq Y$

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

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

2. Calculer directement un automate de Büchi pour la formule

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

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 9 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.