

Devoir maison

1 Automates à parité et automates de Buchi

1.1 Exemple

- Seul l'état 2 peut être répété infiniment souvent. En effet, il n'est pas possible de revenir à l'état 1 depuis le 2, et 3 est impair.

On commence donc par b^*a pour arriver à l'état 2 depuis l'état 1. Ensuite la boucle $\{2, 3\}$ permet de reproduire ce schéma un nombre fini de fois, pour que l'état 2 soit répété infiniment souvent, il est nécessaire de finir sur b^ω .

Le langage reconnu par jeu parité est donc $b^*ab^*a\{b^+a\}^+b^\omega$.

En logique LTL, b^*a correspond à bUa , il doit être doublé (b^*ab^*a) ce qui donne : $bU(a \wedge X(bUa))$.

Il faut maintenant le faire suivre par $\{b^+a\}^+b^\omega$ soit $(b \wedge X(bUa))U(FGb)$ ce qui donne lorsqu'on le met à la suite de notre expression précédente : $bU(a \wedge X(bU((a \wedge X(b \wedge X(bUa))U(FGb))))$.

- Pour construire l'automate du Buchi équivalent, il n'est pas possible de mettre simplement le sommet 2 dans F car cela permettrait d'avoir $3 \in Inf(\rho)$ ce qui ne serait pas reconnu dans l'automate à parité.

Une solution consiste à rendre non déterministe l'automate, en rajoutant un état $2'$, relié à 2 par une arête $2 \rightarrow 2'$ étiqueté b. Une boucle en b permet de rester dans l'état $2'$. De plus, $F = \{2'\}$ ce qui permet de bien finir sur Z^ω .

On obtient donc l'automate présenté à la figure 1.1.

1.2 Parité et Buchi

- (a) Pour transformer un automate de Buchi en automate à parité, on garde l'automate et on numérote les sommets de la manière suivante : $\forall s \notin F, \chi(s) = 1, \forall s \in F, \chi(s) = 2$.

En effet, quand un mot est reconnu par l'automate de Buchi on a au moins un sommet visité infiniment souvent dans F . Dans ce cas, dans l'automate à parité, ce sommet est étiqueté 2. Comme 2 est l'étiquette maximale, ce mot sera accepté par l'automate à parité. Réciproquement, quand un mot est accepté par l'automate à parité, cela signifie que un sommet étiqueté 2 est visité infiniment souvent. Ce sommet appartenant à F dans l'automate de Buchi, le mot sera accepté par ce dernier.

- (b) Il est possible d'obtenir un automate de Buchi indéterministe de la manière suivante :

Soit n l'étiquette maximale de l'automate à parité. Pour chaque étiquette paire p , on considère l'automate A_p sous-automate induit par les sommet portant des étiquettes $\leq p$, on note F_p l'ensemble des sommets d'étiquette maximale dans A_p .

On considère maintenant l'union disjointe des différents A_p et de l'automate à parité initiale (que l'on note A). L'ensemble F correspond à l'union des différents F_p . Soit p_0 une étiquette paire, on associe à chaque sommet de F_{p_0} une epsilon-transition venant de son sommet équivalent dans A (il est possible de se passer d'epsilon-transition en doublant les transitions entrante du sommet dans A et en redirigeant la nouvelle vers le sommet correspondant dans A_{p_0}). Le sommet initial est le sommet correspondant dans A .

L'automate obtenu est alors un automate de Buchi équivalent à l'automate de parité. En effet, dès qu'un mot est reconnu par l'automate à parité, avec p comme étiquette infinie maximale, il existe une exécution infini telle que $Inf(\rho) \subset A_p$ et $Inf(\rho) \cap F_p \neq \text{emptyset}$ (car le mot est reconnu par l'automate à parité), donc

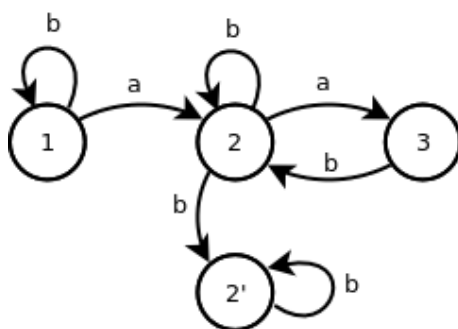


Figure 1: Automate de Buchi équivalent à l'automate A

$Inf(\rho) \cap F \neq \emptyset$ car $F_p \subset F$.

Réciproquement, si un mot est reconnu par l'automate de Buchi, une exécution infini s'enferme dans un des A_p et visite infiniment souvent un des sommet de F_p . Ce mot serait aussi accepté par l'automate à parité puisque il existe un sommet pair (celui visité infiniment souvent et appartenant à F_p) qui est visité infiniment souvent et dont l'étiquette est maximale parmi celle visitées infiniment souvent.

Cet automate de Buchi est donc bien équivalent à l'automate à parité initial.

2 Jeux de parité faible

2.1 Transformation en jeu à parties infinies

Pour éviter d'avoir des parties finies, il est nécessaire que tout sommet ait un degré sortant ≥ 1 . Pour cela il est possible d'ajouter un état puit qui permet d'éviter un blocage. Toutefois, pour conserver la condition de perte lorsqu'on arrive sur un sommet bloquant, il est nécessaire d'avoir 2 états puits, un qui fait gagner J_0 (on le note p_0) et un qui fait gagner J_1 (on le note p_1).

Soit n l'étiquette maximale du graphe, on considère n_0 le plus petit entier pair supérieur à n et n_1 le plus petit entier impair supérieur à n . p_0 porte n_0 pour étiquette et p_1 porte n_1 .

Les deux puits ont une seule arête sortante qui est une boucle sur eux même. Le joueur auquel ils appartiennent n'a pas d'importance.

Vu que n_0 et n_1 sont toutes les 2 plus grandes que le reste des étiquettes du graphes, le fait de visiter un puit (et de s'y emprisonner) déterminera le gagnant de la partie.

Pour tout sommet bloquant appartenant à J_0 , on ajoute une arête sortante vers p_1 et pour tout sommet bloquant appartenant à J_1 on ajoute une arête sortante vers p_0 . Ainsi toute partie est infinie et les conditions de victoires sont conservées.

Au niveau des zones gagnantes, p_0 et p_1 sont des zones gagnantes pour respectivement J_0 et J_1 . Il est possible de déterminer les régions gagnantes pour J_0 et J_1 , à l'exceptions des puits, ces zones seront les même dans l'arène initiale.

2.2 Exemple

Considérons tout d'abord Win_1 puisque l'étiquette maximale, celle de d est impaire.

On a donc $\{b, d\} \subseteq Win_1$ puisque b appartient à J_1 et une arête relie b à d .

Il n'est pas possible d'ajouter pour l'instant d'autres sommets à Win_1 puisque les voisins de b et de d sont tous à J_0 et ce dernier à d'autres possibilité que d'aller en b ou d .

Calculons maintenant l'attracteur issu de $\{c, g\}$ qui sont les 2 sommets d'étiquette maximale dans $G \setminus \{b, d\}$.

$f \in Win_0$ car il existe une arête allant de f à g et f appartient à J_0 . Il n'est pas possible d'ajouter d'autres sommets dont $\{c, g\}$ seraient attracteurs.

Dans $G \setminus \{d, b, c, g, f\}$ l'étiquette maximale est celle de a et e et elle est paire (0). Donc ces 2 sommets sont dans Win_0 puisqu'ils n'appartiennent à aucun autre attracteur et son d'étiquette paire. En effet, si le jeu démarre sur e , J_1 est obligé d'avancer en a , et une fois sur a , J_0 peut faire une boucle infinie sur a .

On a donc $Win_0 = \{a, c, e, f, g\}$.

2.3 Positions gagnantes et μ -calcul

- Win_0 est le plus grand point fixe d'une fonction devant traiter deux cas, soit le sommet appartient J_0 , dans ce cas, un de ces successeurs doit appartenir à Win_0 , soit le sommet appartient à J_1 et tous les successeurs doivent appartenir à Win_0 . De plus, tout chemin restant infiniment dans des sommets d'étiquettes 0 est gagnant, si un sommet composant ce chemin appartient à J_1 tous les successeurs de ce sommets doivent être appartenir à l'ensemble solution.

Toutefois, je n'arrive pas à utiliser les termes de μ -calcul avec l'alternance des joueurs. Il est peut être possible d'utiliser le terme $X \wedge c_1$ pour désigner les sommets de l'ensemble appartenant à J_1 toutefois, je ne vois pas comment s'assurer de l'appartenance de l'ensemble des successeur de ces sommets à l'ensemble des sommets étiquetés 0.

-
-

2.4 Jeux de parités et jeux de Buchi

On reprend plus ou moins le même principe que dans la question 1, c'est à dire l'union disjointe de plusieurs graphes correspondant aux sous graphes induits par G telles que les étiquettes d'un sous graphe G_i soient $\leq i$.

Soit i une des couleurs de G , on considère G_i , à chaque sommet de G_i d'étiquette i , on ajoute une epsilon transition vers le sommet équivalent dans G_{i+1} .

Ensuite pour la définition de F , pour tout i pair, $V(G_i) \subseteq F$. Pour i impair, $V(G_i) \cap F = \emptyset$.

Lors de la "lecture" d'un mot, l'exécution va aller s'enfermer dans le sous graphe d'étiquette maximale rencontré. Si cette étiquette est paire, le mot sera accepté (puisque tous les sommets du sous graphes dans lequel s'enferme l'exécution sont finaux), si elle est impair le mot est refusé.

En considérant que les sommets maximaux de chaque G_i sont ceux de G , on a bien le fait qu'un sommet est gagnant dans G' si et seulement si il est gagnant dans G . En effet, tous les sommets maximaux d'un G_i pour i pair sont gagnants (sommet d'étiquette paire dans G) et tous les sommets maximaux d'un G_i pour i impair sont perdants.

Le nombre de sommets du nouveau graphe est en $O(n^2)$ et la construction de ce graphe ne demande aucun calcul particulier. Cette dernière est donc réalisable en temps polynomiale. Toutefois, le graphe ainsi obtenu est non-déterministe et je ne suis pas sur que cela soit possible pour un jeu.

3 Jeux de parité et algorithmes

La première étape de l'algorithme consiste à déterminer Win_0 . Pour cela, on calcule Win_0 et Win_1 à l'aide d'attracteur. On considère tout d'abord les sommets d'étiquettes maximales, si cette dernière est paire (resp. impaire), on calcule l'attracteur pour J_0 (resp. J_1) à partir de ces sommets. Puis on retire l'ensemble de sommets obtenu du graphe et on l'ajoute à Win_0 (resp. Win_1) et on calcule l'attracteur suivant. Chaque sommet étant traité au plus n fois (majoration large), cet étape admet une complexité en n^2 .

Ensuite, l'algorithme vérifie que v appartient bien à Win_0 . Si ce n'est pas le cas, φ ne peut pas être une stratégie gagnante puisque v est une position gagnante pour J_1 .

Si c'est le cas, l'algorithme doit vérifier le reste de la stratégie. Pour cela, on associe à chaque sommet de Win_0 un booléen initialement à faux qui indique si le sommet a déjà été vérifié ou pas.

On utilise ensuite une pile qui va permettre de vérifier la stratégie pour les sommets accessibles depuis v . Deux cas sont possible lors de cette vérification, soit le sommet appartient à V_0 , dans ce cas, la stratégie est consulté pour connaître le sommet suivant choisi. Si ce dernier n'appartient pas à Win_0 la stratégie est incorrecte, sinon, si le sommet n'a pas encore été traité, on l'ajoute à la pile. Dans l'autre cas, c'est à dire si

le sommet appartient à V_1 , on ajoute à la pile les successeurs n'ayant pas encore été vérifiés.

Un sommet n'étant vérifié que une fois, et le graphe étant fini, cet algorithme termine bien.

Algorithm 1 L'algorithme de vérification d'une stratégie

```

Calcul de  $Win_0$ 
verif est un tableau de  $|Win_0|$  booléens indexé par les sommets de  $Win_0$ 
 $P := empiler(P, v)$ 
while P n'est pas vide do
5:    $s = depiler(P)$ 
      if  $s \in V_0$  then
           $suivant := \varphi(s)$ 
          if  $suivant \notin Win_0$  then
              Renvoyer Faux
10:   end if
          if  $\neg verif[suivant]$  then
               $verif[suivant] := Vrai$ 
               $P := empiler(P, suivant)$ 
          end if
15:   else
          for all successeur  $suc$  de  $s$  do
              if  $\neg verif[suc]$  then
                   $verif[suc] := Vrai$ 
                   $P := empiler(P, suc)$ 
20:   end if
          end for
      end if
end while
Renvoyer Vrai

```

La complexité de cette 2 étape est en $O(n^2)$, en effet, chaque sommet est vérifié au plus une fois, et si il s'agit d'un sommet de J_1 , l'ensemble de ses successeurs sont vérifiés (si besoin). Cela fait donc une complexité en n^2 . La complexité globale de l'algorithme est donc en $O(n^2)$.

3.1 Jeux de parité sur des arènes sans grand cycle

Notre algorithme va initialement trier les sommets du graphe afin d'obtenir la liste des sommets triés par degré sortant sans prendre en compte les boucles. Ainsi, on obtient facilement tous les sommets puits.

Il est simple de déterminer pour qui sont gagnants ces sommets puits, puisque cela dépend uniquement de la parité de l'étiquette.

Une fois que le joueur gagnant est déterminé pour un puit, on indique à son prédécesseur que le puit est déterminé. Un sommet qui n'a que des successeurs déterminés peut l'être à son tour. Si le sommet appartient à J_0 , il est gagnant ssi au moins un de ses successeurs est gagnant. Si le sommet appartient à J_1 , il est gagnant ssi tous ses successeurs sont gagnants. Lorsque le sommet admet une boucle, il peut échapper à cette règle, si il est pair et appartient à J_0 , il est gagnant, si il est impair et appartient à J_1 , il est perdant.

Le graphe étant acyclique, il existe toujours un sommet pouvant être traité (puits ou sommet dont tous les successeurs ont été traités). Pour obtenir facilement la liste des sommets à traiter, il est possible d'informer tous les prédécesseurs lors du traitement d'un sommet et de décrémenter une variable indiquant le nombre de successeurs non traités. Lorsque cette variable atteint 0, on ajoute le sommet à la liste des sommets pouvant être traités.

L'algorithme est donc le suivant :

Tous les sommets sont traités puisqu'on épluche le DAG à partir des feuilles en remontant jusqu'aux racines. On peut mettre en place une fonction hauteur, égale à 0 pour les racines, et égale au max + 1 de la hauteur des prédécesseurs. Comme le graphe est sans cycle, cette fonction est définie et finie pour chaque sommet du graphe.

En considérant un dessin par étage, ou chaque sommet est dessiné dans l'étage correspondant à sa hauteur (étage 0 tout en haut), on a que tous les successeurs d'un sommet sont dessinés en dessous de lui. L'algorithme proposé remonte progressivement l'arbre à partir du dernier étage, étage par étage (enfin, cela dépend de l'ordre de traitement des sommets dans la pile, mais on peut toujours se ramener à une exécution vérifiant cette propriété en réalisant des réorganisations de la pile en cours d'exécution).

Ce dessin permet d'être sûr que tous les successeurs d'un sommet ont été traités lorsqu'il arrive son tour et qu'il est donc possible d'utiliser Win_0 et Win_1 qui sont donc à jour. De plus, lorsqu'on finit de traiter un étage, tous les sommets de l'étage suivant sont dans la liste, puisque tous leurs successeurs ont été traités. On en déduit que tous les sommets sont bien traités une fois. On vérifie tous les successeurs de chaque sommet à chaque étape, l'algorithme est donc en n^2 .

Algorithm 3 Suite de l'algorithme

```
    else
        ▷ On traite maintenant le cas  $s \in J_1$ 
        if  $s$  admet une boucle et son étiquette est impaire then
             $Win_1 = Win_1 \cup s$ 
40:      $defaite := Faux$ 
            for all successeur  $suc$  de  $s$  do
                if  $suc \in Win_1$  then
                     $defaite = Vrai$ 
                end if
45:     end for
            ▷ Si au moins un des successeurs est perdant, le sommet est perdant
            if  $defaite$  then
                 $Win_1 = Win_1 \cup s$ 
            else
50:      $Win_0 = Win_0 \cup s$ 
            end if
        end if
    end if
end while
55: Retourner  $Win_0$  et  $Win_1$ 
```
