Algorithmes de Markov

Vincent Laviron

4 avril 2007

Table des matières

Introduction

Les algorithmes de Markov permettent de transformer un mot, donné dans un certain alphabet, en un autre mot, par une série de règles élémentaires. On peut en donner plusieurs définitions, mais on va prouver que les deux définitions que l'on va donner ici sont équivalentes, ce qui permet de choisir la plus facile à manipuler.

Le but est ici d'étudier les possibilités de simulation offertes par les algorithmes de Markov, en particulier par rapport à d'autres systèmes de calcul plus connus comme les fonctions primitives récursives et les machines de Turing.

1 Définitions

Définition 1.1 *Algorithme de Markov : Soit Σ un alphabet. Un algorithme de Markov sur l'alphabet Σ est une suite d'instructons de la forme*

$$i : x_i / y_i : l_i$$

avec i le numéro de l'instruction, x_i et y_i des mots de Σ^* et l_i est un numéro d'instruction, plus une dèrnière instruction de la forme

$$i : STOP$$

Un algorithme de Markov prend en entrée un mot de Σ^*, et pour chaque instruction i á partir de la première, il lit le mot qu'il a en partant de la gauche jusqu'à ce qu'il trouve un sous-mot x_i et le remplace par y_i, puis passe à l'instruction l_i. Si il n'y a pas de sous-mot x_i, il passe á l'instruction $i + 1$. Quand il arrive à l'instruction $STOP$, il s'arrête et renvoie le mot qu'il a.

Exemple 1.2 *L'algorithme de Markov défini sur l'alphabet $\{0, 1, D, T, E, F\}$ par :*

$$1 : D/DT : 2$$
$$2 : F/EF : 3$$
$$3 : T0/0T : 6$$
$$4 : T1/1T : 7$$
$$5 : TE/\varepsilon : 8$$
$$6 : F/0F : 3$$
$$7 : F/1F : 3$$
$$8 : STOP$$

renvoie, pour un mot u de $\{0, 1\}^*$ donné, et sur l'entrée DuF, le résultat $DuuF$.

Définition 1.3 *Variante : On peut aussi définir l'algorithme de Markov sans les étiquettes ; dans ce cas, les instructions sont de la forme*

$$x_i \to y_i$$

ou de la forme

$$x_i \to_{(t)} y_i$$

pour indiquer qu'elles sont terminales.

Dans cette variante, l'algorithme essaye toutes les instructions à partir de la première, et dès qu'il en trouve une qui marche il l'applique. Si cette instruction est terminale il s'arrête, sinon il reprend avec le nouveau mot à la première insêtruction. On peut montrer que cette variante des algorithmes de Markov permet de simuler les machines de Turing.

Exemple 1.4 *L'algorithme de Markov défini sur l'alphabet* $\{a, b, \alpha, \beta\}$ *par :*

$$\alpha\alpha \to \beta$$

$$\beta a \to a\beta$$

$$\beta b \to b\beta$$

$$\beta\alpha \to \beta$$

$$\beta \to_{(t)} \varepsilon$$

$$\alpha aa \to a\alpha a$$

$$\alpha ab \to b\alpha a$$

$$\alpha ba \to a\alpha b$$

$$\alpha bb \to b\alpha b$$

$$\varepsilon \to \alpha$$

prend en entrée un mot de $\{a, b\}^*$ *et renvoie son « miroir ».*

Proposition 1.5 *Les deux définitions proposées sont équivalentes.*

Preuve. D'abord, un algorithme de Markov de la deuxième forme peut être transformé en un algorithme de Markov de la première forme qui fait la même chose en étiquetant les instructions dans l'ordre, et en renvoyant à la première instruction pour les instructions non terminales, et en renvoyant à une instruction supplémentaire $STOP$ placée à la fin pour les instructions terminales.

Pour simuler en sens inverse, il faut introduire des marqueurs : ce sont des lettres qui n'apparaissent pas dans l'algorithme. On pourra par exemple les appeler α_i et β_i. Si k est le nombre total d'instructions (y compris l'instruction $STOP$), à chaque instruction d'étiquette $i \leqslant k$, on va donc associer un marqueur α_i et un marqueur β_i. Ensuite, pour toutes les instructions d'étiquette $i < k$, on remplace le mot x_i par le mot $\alpha_i x_i$ et le mot y_i par le mot $\beta_{l_i} y_i$ (pour l'instruction $STOP$, d'étiquette k, on la remplace par l'instruction $\beta_k \to_{(t)} \varepsilon$). Enfin, on rajoute à la fin les instructions :

$$\sigma\beta_i \to \beta_i\sigma$$

pour tout $\sigma \in \Sigma$ et pour tout $i < k$, puis les instructions :

$$\beta_i \to \alpha_i$$

pour tout $i < k$, et encore les instructions :

$$\alpha_i\sigma \to \sigma\alpha_i$$

pour tout $\sigma \in \Sigma$ et pour tout $i < k$, et enfin les instructions :

$$\alpha_i \to \beta_{i+1}$$

pour tout $i < k$ et l'instruction d'initialisation $\varepsilon \to \alpha_1$.

On peut alors montrer facilement qu'il y a toujours un et un seul marqueur à la fois, et donc qu'on est bien dans un seul état à la fois. De même, on peut montrer que toutes les propriétés sont conservées : terminaison, correction ...

On a donc « montré »que les deux définitions sont éuivalentes.

Dans la suite, on utilisera la première définition, plus intuitive pour donner des algorithmes faisant une tâche donnée.

2 Calculs avec les algorithmes de Markov

On peut utiliser les algorithmes de Markov pour calculer sur les entiers en les représentant en unaire ou en binaire. Voici par exemple un algorithme d'addition en binaire :

Exemple 2.1 *Addition :*

$$1 : +/+a : 2$$
$$2 : a0/0a : 2$$
$$3 : a1/1a : 2$$
$$4 : 0+/+ : 7$$
$$5 : 1+/+ : 9$$
$$6 : +/\varepsilon : 14$$
$$7 : 0a/a0 : 4$$
$$8 : 1a/a1 : 4$$
$$9 : 0a/a1 : 4$$
$$10 : 1a/ba0 : 11$$
$$11 : 1b/b0 : 11$$
$$12 : 0b/1 : 4$$
$$13 : +b/+1 : 4$$
$$14 : STOP$$

On peut de même définir la multiplication, et d'autres opérations usuelles, grâce à ces algorithmes de Markov.

La composition de deux algorithmes de Markov se fait en mettant le deuxième à la suite du premier (en enlevant l'instruction $STOP$ du premier).

On peut également traiter la récurrence de la même façon : au lieu de renvoyer vers STOP, on renvoie vers le début de l'algorithme, et on introduit une ou plusieurs instructions de contrôle pour les cas de base.

Proposition 2.2 *Simulation des fonctions primitives récursives par des algorithmes de Markov*

Toute fonction primitive récursive peut être simulée par un algorithme de Markov.

Preuve. On notera par $*$ la séparation entre deux entiers pour les couples.

Les fonctions constantes peuvent être simulées par des algorithmes de la forme :

$$1 : 0/\varepsilon : 1$$

$$2 : 1/\varepsilon : 1$$

$$3 : */\varepsilon : 1$$

$$4 : \varepsilon/n : 5$$

$$5 : STOP$$

qui simule la fonction constante de valeur n.

Pour le successeur, on peut adapter sans problème l'algorithme d'addition présenté en exemple 2.1.

Enfin, pour les projecteurs, par exemple $\mathbf{proj}_{3,2}$, on a l'algorithme :

$$1 : 0/\varepsilon : 1$$

$$2 : 1/\varepsilon : 1$$

$$3 : */\varepsilon : 4$$

$$4 : */a : 5$$

$$5 : a0/a : 5$$

$$6 : a1/a : 5$$

$$7 : a*/a : 5$$

$$8 : STOP$$

En fait, pour $\mathbf{proj}_{p,i}$, les trois premières lignes sont répétées $p - i$ fois.

On a vu comment faire une composition classique $(f \circ g)$, pour le cas général il faut faire plus attention : une solution est d'utiliser un algorithme tel celui de l'exemple 1.2 pour dupliquer le code de départ suffisamment de fois, en rajoutant des séparateurs, puis appliquer sur chaque partie du code un premier algorithme qui marque la partie du code à traiter, par exemple en ajoutant devant tous les symboles à traiter un symbole non utilisé par ailleurs, ensuite modifier les algorithmes des fonctions à appliquer sur le mot de base pour qu'ils n'opèrent que sur le code marqué, puis enlever les marques en prévision de la fonction suivante à appliquer. Après avoir appliqué toutes les fonctions à partir desquelles on compose, on transforme les séparateurs en symboles $*$ puis on applique l'algorithme pour la dernière fonction.

Il reste à traiter le cas de la récurrence ; pour cela, on suppose que l'on a un algorithme G qui calcule la fonction associée à la fonction g du cas $n = 0$, et un algorithme H qui calcule la fonction de récurrence. On suppose qu'on peut appliquer les algorihmes sur des parties du mot en introduisant des marqueurs. De même, on suppose qu'on peut recopier certaines parties du mot lorsqu'on en a besoin. Il suffit alors de garder en « mémoire »les arguments, et d'appliquer

d'abord G, puis H en recopiant les arguments dont on a besoin, en testant à chaque itération si on a atteint la valeur de n sur laquelle on a appelé l'algorithme.

On peut prendre pour cela l'algorithme de test dégalité suivant :

$$1 : \varepsilon/\alpha : 2$$
$$2 : */ * \beta : 3$$
$$3 : \alpha 0/0\alpha : 6$$
$$4 : \alpha 1/1\alpha : 8$$
$$5 : \alpha * /* : 10$$
$$6 : \beta 0/0\beta : 3$$
$$7 : \varepsilon/\gamma : 12$$
$$8 : \beta 1/1\beta : 3$$
$$9 : \varepsilon/\gamma : 12$$
$$10 : \beta 0/\gamma : 12$$
$$11 : \beta 1/\gamma : 12$$
$$12 : \beta/\varepsilon : 12$$
$$13 : 1/\varepsilon : 12$$
$$14 : 0/\varepsilon : 12$$
$$15 : */\varepsilon : 12$$
$$16 : \alpha/\varepsilon : 12$$
$$17 : \gamma/1 : 19$$
$$18 : \varepsilon/0 : 19$$
$$19 : STOP$$

On a prouvé que les algorithmes de Markov permettaient de calculer toutes les fonctions primitives récursives. En fait, on peut même prouver que les algorithmes de Markov sont équivalents aux machines de Turing.

3 Algorithmes de Markov et machines de Turing

Proposition 3.1 *Tout algorithme de Markov peut être simulé par une machine de Turing.*

Preuve. On ne va pas simuler chaque instruction par un état mais par un ensemble d'états :

- Un automate pour reconnaître le motif x_i dans le mot ;
- Un ensemble d'états pour effacer le mot reconnu et écrire le mot y_i â la place, ce qui implique dans le cas général de décaler tout le reste du mot, puis qui renvoie vers l'ensemble d'états correspondant â l_i ;
- Un ensemble d'états pour passer â l'état suivant si x_i n'est pas reconnu.

Pour le cas de l'instruction $STOP$, il suffit d'un état final.

On a donc prouvé que les machines de Turing pouvaient simuler les algorithmes de Markov. Voyons maintenant la réciproque.

Proposition 3.2 *Toute machine de Turing peut être simulée par un algorithme de Markov.*

Preuve. On va le prouver pour des machines de Turing déterministes, et on utilisera ensuite le fait que toute machine de Turing est équivalente à une machine de Turing déterministe.

Pour simuler une machine de Turing $\mathcal{M}$, on va introduire un marqueur α qui va représenter la position de la tête de lecture de la machine de Turing ; on suppose que α n'est pas dans l'alphabet de bande Γ de la machine de Turing. La première instruction de l'algorithme de Markov sera d'initialiser le marqueur α : on aura donc l'instruction $1 : \varepsilon/\alpha : i$ où i est la première instruction de l'ensemble d'instructions correspondant à l'état q_0. Ensuite, pour chaque état q, on associe un ensemble d'instructions de la forme :

- $i : \alpha a/b\alpha : j$ avec j la première instruction de l'ensemble d'instructions correspondant à l'état q' si $q, a \to q', b, R$ est une transition de $\mathcal{M}$ et q' n'est pas final ;
- $i : \alpha a/b : f$ où f est l'étiquette de l'instruction $STOP$ si $q, a \to q', b, R$ est une transition de $\mathcal{M}$ et q' est final ;
- $i : \gamma\alpha a/\alpha\gamma b : j$ avec j la première instruction de l'ensemble d'instructions correspondant à l'état q' si $q, a \to q', b, L$ est une transition de $\mathcal{M}$ et q' n'est pas final, et ce pour tout $\gamma \in \Gamma$.
- $i : \gamma\alpha a/\gamma b : f$ où f est l'étiquette de l'instruction $STOP$ si $q, a \to q', b, L$ est une transition de $\mathcal{M}$ et q' est final, et ce pour tout $\gamma \in \Gamma$.

Comme on a supposé que $\mathcal{M}$ était déterministe, on n'a aucun conflit sur l'ordre des instructions. Enfin, on rajoute à cet ensemble d'instructions une instruction du type $i : \alpha/\varepsilon : f$, où f est l'étiquette de l'instruction $STOP$, pour traiter les cas où la machine se bloque.

Il suffit alors de recoller tous ces blocs dans un ordre quelconque pour obtenir (avec l'instruction initiale et l'instruction finale $STOP$) un algorithme de Markov qui simule la machine de Turing $\mathcal{M}$. La vérification de cette assertion se fait sans trop de difficulté, mais elle est relativement laborieuse et utilise abondamment le fait que $\mathcal{M}$ est déterministe.

On a donc prouvé que les algorithmes de Markov sont aussi puissants en termes de calcul que les machines de Turing. En particulier, ils permettent de calculer toute fonction récursive (et pas seulement primitive récursive), ce qu'une preuve directe aurait pu nous montrer aussi.