École polytechnique — X2013 — INF412 Fondements de l'informatique

Machines de Turing

Sujet proposé par Bruno Salvy
(corrigé)

Les machines de Turing sont un de ces sujets qui paraissent simples et ennuyeux à la lecture, mais qui s'avèrent subtils et intéressants si l'on essaye de les construire par soi-même. L'objectif de ce sujet est de faire appréhender par la construction "manuelle" la puissance d'expressivité de ce modèle.

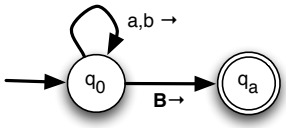
Définition. Le point de départ sera la définition du polycopié, rappelée ici :
Une machine de Turing est un octuplet $M = (Q, \Sigma, \Gamma, \mathbf{B}, \delta, q_0, q_a, q_r)$ *où*

- Q *est l'ensemble fini des états ;*
- Σ *est un alphabet fini ;*
- $\Gamma \supset \Sigma$ *est l'alphabet de travail fini ;*
- $\mathbf{B} \in \Gamma \setminus \Sigma$ *est le caractère blanc ;*
- q_0, q_a, q_r *sont des éléments de* Q *appelés état initial, état d'acceptation, état de refus ;*
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, |, \rightarrow\}$ *est la fonction de transition.*

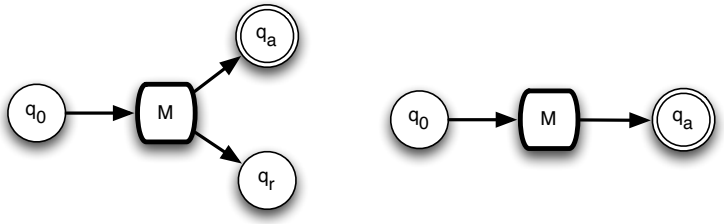
Notations. Une machine de Turing sera définie graphiquement par un graphe dont les sommets sont les états, les arcs représentant les transitions. Une transition $\delta(q_1, \ell) = (q_2, m, d)$, où q_1, q_2 sont dans Q, les lettres ℓ, m dans Γ et d est une direction sera représentée par une étiquette ℓ/md sur l'arc reliant le sommet q_1 au sommet q_2. Pour alléger les figures, on ajoute trois conventions :

- dans le cas où $m = \ell$, on se contentera de l'étiquette ℓd ;
- lorsqu'existent plusieurs transitions du type précédent entre deux mêmes états, on étiquettera l'arc $\ell_1, \ell_2, \ldots d$;
- les arcs pointant vers l'état de refus ne seront pas représentés.

Par exemple, voici une représentation de la machine de Turing acceptant les mots n'utilisant que les lettres a et b de Σ :



Sous-programmes. On s'efforcera de programmer les machines de Turing de façon *modulaire*, en réutilisant des machines déjà écrites. Pour cela, une machine sera symbolisée par l'un des deux schémas



selon que la transition vers l'état de refus est réutilisée pour pointer vers un autre état ou non.

L'utilisation de machines comme sous-programmes impose une spécification très précise de leurs droits. Ainsi, sauf indication contraire, les machines de Turing n'écriront jamais à gauche

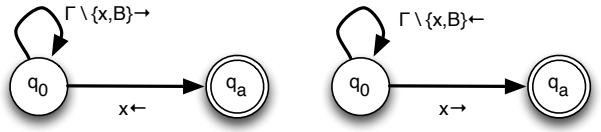
de la position initiale de leur tête de lecture, le mot donné en entrée sera toujours borné par un **B** à gauche et à droite, et (à partir de la question 4.3) lorsqu'une machine de Turing s'arrêtera sur son état d'acceptation, sa tête de lecture sera d'abord revenue à sa position initiale.

1 Déplacements sur le ruban

La programmation de machines de Turing pousse à déplacer fréquemment la tête de lecture sur le ruban, par exemple jusqu'aux extrémités. Les deux sous-routines suivantes seront donc très utiles.

Question 1.1. *Écrire une machine de Turing* F_x *(pour* Forward*) qui avance la tête de lecture sur le ruban jusqu'à la première occurrence de la lettre* $x \in \Gamma$ *et recule alors sur le caractère précédent. Cette machine refuse les mots qui ne contiennent pas* x*. Écrire de même la fonction symétrique* B_x *(pour* Backward*) qui recule et termine sur le premier caractère suivant l'occurrence de* x *précédente.*

Solution : La machine F_x avance sa tête de lecture tant qu'elle ne lit pas de x, et va dans l'état d'acceptation dès qu'elle en rencontre un. La transition vers l'état de refus qui n'est pas indiquée se produit lorsque le mot ne contient pas la lettre x, ce qui est détecté en lecture d'un blanc en q_0, sauf si $x = \mathbf{B}$. Il y a une petite difficulté puisque x est supposé appartenir à Γ et non Σ, ce qui entraine un traitement spécial pour **B** afin d'éviter que la machine ne boucle au lieu de refuser. La machine B_x est obtenue comme miroir de F_x en renversant les flèches.



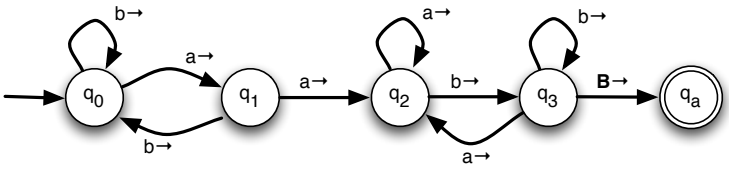
□

2 Langages rationnels

Les machines de Turing restreintes dont les transitions sont toutes de la forme $\delta(q_1, \ell) = (q_2, \ell, \rightarrow)$ sont classiquement appelées automates finis déterministes. Elles reconnaissent des langages appelés rationnels, dont l'exercice suivant donne un exemple typique.

Question 2.1. *Écrire une machine de Turing acceptant les mots sur l'alphabet* $\Sigma = \{a, b\}$ *qui contiennent le sous-mot aab et se terminent par un b, et refusant tous les autres mots sur* Σ*.*

Solution : La machine suivante effectue ce travail :



La lecture s'effectue depuis l'état initial q_0 jusqu'à l'état d'acceptation q_a. Tant que la machine ne rencontre pas le premier a, elle avance sa lecture et reste sur son état initial. Dès qu'un premier a est rencontré, elle passe dans l'état q_1, qu'elle quitte vers q_2 si un deuxième a suit le premier, alors qu'elle doit retourner en q_0 sinon. Une fois le sous-mot aa rencontré, la machine reste dans l'état q_2 tant qu'elle ne lit pas un b, qui lui permet d'arriver en q_3. À ce stade, il ne reste plus qu'à attendre le b final, c'est-à-dire le sous-mot $b\mathbf{B}$, ce qui se traduit par des aller-retours entre q_2 et q_3 en attendant la fin du mot. Les transitions vers l'état de refus ne sont pas

indiquées. Elles ont lieu lorsqu'une lettre ne correspondant pas à une transition indiquée est lue, c'est-à-dire pour cette machine si la lettre **B** est lue dans l'un des états q_0, q_1, q_2. □

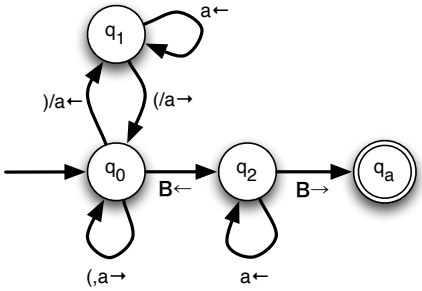
Les langages rationnels servent à exprimer les "expressions régulières" qui sont utilisés dans de nombreux éditeurs de texte pour effectuer de la recherche avancée. Ne pouvant ni écrire, ni revenir en arrière, ces automates sont incapables de reconnaître des langages dont les mots ont une structure de dépendance à longue portée, mais permettent des recherches très rapides et avec peu de mémoire dans de très grand textes comme le génome.

3 Langages context-free

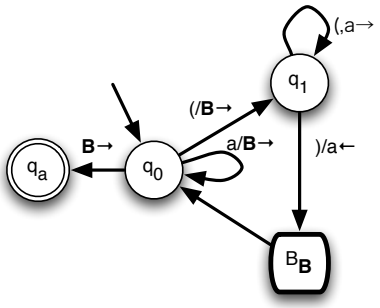
Les langages de programmation ne sont pas des langages rationnels, mais sont souvent "context-free", c'est-à-dire une généralisation des langages rationnels, qui permet de reconnaître des mots bien parenthésés (penser à des *begin*...*end* ou *if*...*fi*). Pour les reconnaître, il est crucial de pouvoir écrire sur le ruban.

Question 3.1. *Écrire une machine de Turing acceptant les mots bien parenthésés sur l'alphabet* $\{a,), (\}$. *Ces mots ont le même nombre de parenthèses fermantes que de parenthèses ouvrantes, et aucun préfixe ne possède plus de parenthèses fermantes que d'ouvrantes. (Indication : on peut effacer le mot initialement présent sur le ruban).*

Solution : Il y a (au moins) deux variantes correspondant à des algorithmes de natures différentes. Dans la première variante ci-dessous, la machine efface des paires de parenthèses qui se correspondent dans le mot. La tête de lecture cherche la première parenthèse fermante à partir du début du mot, la remplace par un a et repart en arrière à la recherche de la parenthèse ouvrante correspondante, qu'elle remplace à son tour par un a, avant de revenir à l'état initial. Lorsque la fin du mot est atteinte, pour vérifier que le mot ne comportait pas un excédent de parenthèses ouvrantes, le mot est balayé en reculant pour s'assurer qu'il n'y reste que des a avant d'accepter. Les refus se produisent soit en lecture d'une parenthèse ouvrante en q_2 (parenthèse qui n'est pas refermée), soit en lecture d'une parenthèse fermante en q_0 (parenthèse qui n'en referme pas d'autre).



Dans la seconde variante ci-dessous, la machine se contente de maintenir le nombre de parenthèse ouvrantes supérieur ou égal au nombre de parenthèses fermantes dans les facteurs gauches. La tête de lecture efface tous les caractères jusqu'à la première parenthèse ouvrante incluse en les remplaçant par des blancs, puis va chercher la première parenthèse fermante et la remplace par un a avant de revenir au début de ce qui reste du mot. Lorsque le mot est complètement effacé, la machine accepte. Les refus se produisent soit en lecture d'une parenthèse fermante en q_0 (parenthèse qui n'en referme pas d'autre), soit en lecture d'un blanc en q_1 (fin de mot prématurée).

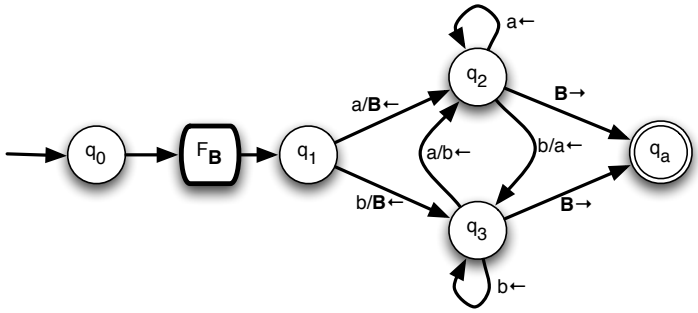


□

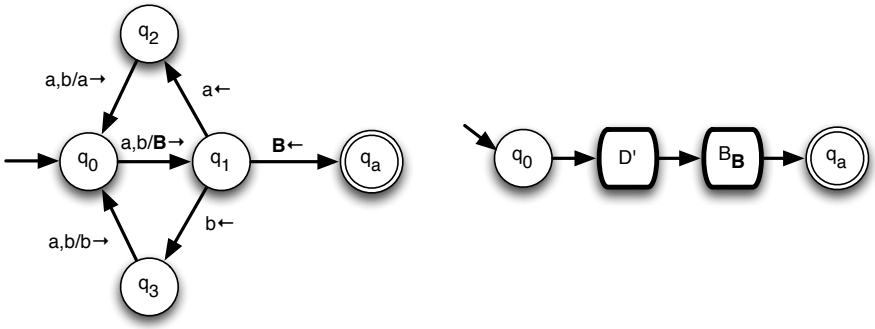
4 Copies et effacements

Question 4.1. *Écrire deux machines* D *et* D′ *(pour* delete*) qui prennent en entrée un mot ℓw où $\ell \in \Sigma = \{a, b\}$ et w est un mot sur Σ, et s'arrêtent avec le mot w sur leur ruban. Il faut donc décaler d'une lettre toutes les lettres de w. La machine* D *termine avec sa tête de lecture sur le premier caractère de w. La machine* D′ *termine avec sa tête de lecture sur le premier caractère suivant w, mais ne devra pas supposer que le premier caractère à gauche de ℓ est un blanc (cela sera utile pour des utilisations comme sous-programme plus loin).*

Solution : Là encore, il y a plusieurs façons de procéder. La première machine D ci-dessous commence par aller à la fin du mot (par $\mathsf{F}_{\mathbf{B}}$), efface le dernier caractère en le mémorisant par son choix de branche (vers q_2 si c'est un a, vers q_3 si c'est un b), puis décale le mot d'un caractère en passant de q_2 à q_3 selon la lettre précédente. Elle termine en atteignant le blanc précédant le début du mot.



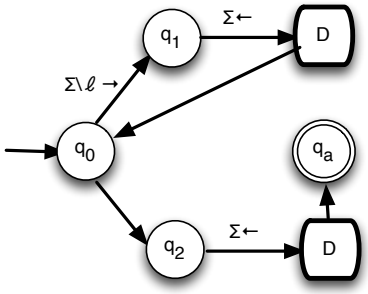
La seconde variante fonctionne pour D et D′. La tête de lecture saute le premier caractère, lit le suivant, puis recule pour l'écrire à la place du précédent, et recommence jusqu'à rencontrer la fin du mot. À ce stade, soit elle s'arrête en acceptant, ce qui donne D′ (à gauche). Soit elle revient au début du mot, ce qui donne D (à droite).



□

Question 4.2. *Écrire une machine* D_ℓ *qui prend en entrée un mot sur Σ et en efface tous les caractères jusqu'au premier $\ell \in \Sigma$ inclus.*

Solution : Il suffit d'utiliser la machine précédente.



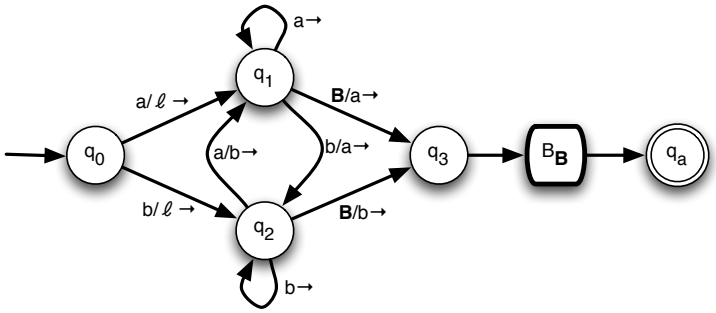
□

Question 4.3. *À l'inverse, écrire trois machines* I_ℓ, I'_ℓ *et* I''_ℓ *(pour* insert*) qui prennent en entrée un mot* w *sur l'alphabet* $\{a, b\}$*, et s'arrêtent avec respectivement les mots* ℓw*,* $w\ell$*,* $w\ell$ *sur leurs rubans. La différence entre* I'_ℓ *et* I''_ℓ *est que cette dernière laisse sa tête de lecture sur le caractère suivant* $w\ell$*.*

Solution : L'insertion en fin de mot par I''_ℓ (à gauche ci-dessous) est assez simple, et I'_ℓ (à droite) s'en déduit :

La tête de lecture est emmenée d'abord sur la dernière lettre du mot, puis un caractère plus loin où le ℓ est inséré, avant de ramener la tête de lecture sur la première lettre dans le cas de I'_ℓ.

La machine I_ℓ, comme D et D', mémorise par le contrôle de la machine les lettres à copier :

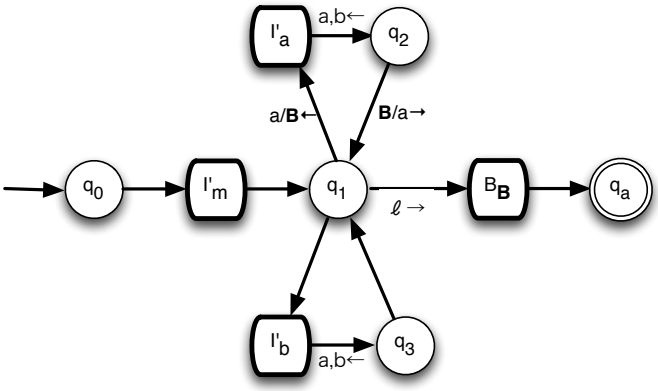


□

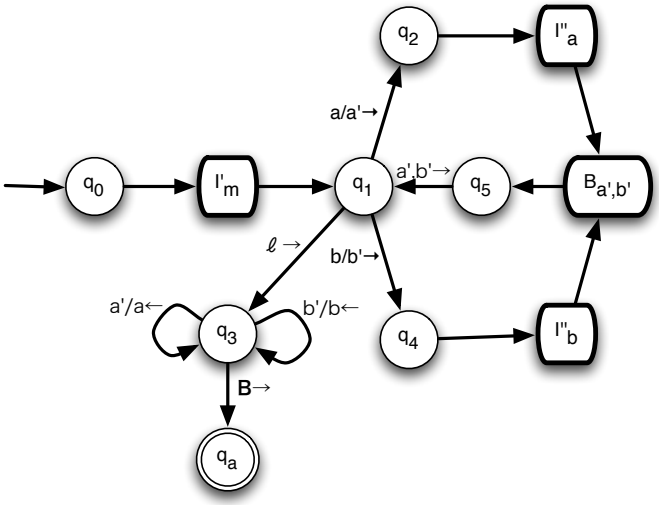
Question 4.4. *Écrire une machine* Copy^m_ℓ *qui prend un mot de la forme* $w_1\ell w_2$ *où* w_1 *et* w_2 *sont des mots sur* $\Sigma = \{a, b\}$*,* ℓ *et* m *sont des lettres différentes de* $\mathbf{B}$ *dans* $\Gamma \setminus \Sigma$*, et s'arrête avec le mot* $w_1\ell w_2 m w_1$ *sur son ruban.*

Solution : Une difficulté est, en cours de copie, de mémoriser la partie du mot déjà copiée. Voici deux solutions.

La machine ci-dessous marque la lettre en cours de copie par un blanc. Les transitions vers q_2 et q_3 exploitent le fait que la machine I' ramène la tête de lecture sur le premier caractère suivant le blanc précédent.



La seconde variante étend l'alphabet de travail par des lettres a', b' et remplace les lettres copiées par leur version 'primée' au fur et à mesure, avant de les nettoyer à la fin. La machine $\mathsf{B}_{a',b'}$ qui y est employée comme sous-routine est une variante de B qui cherche plusieurs lettres à la fois (obtenue en remplaçant x par a', b' dans la machine B_x).



Dans la suite on s'autorisera d'employer les noms de ces machines pour insérer ou supprimer des lettres, même lorsque les alphabets seront différents de $\{a, b\}$. Ces noms représenteront alors les machines construites selon les mêmes principes sur ces alphabets.

□

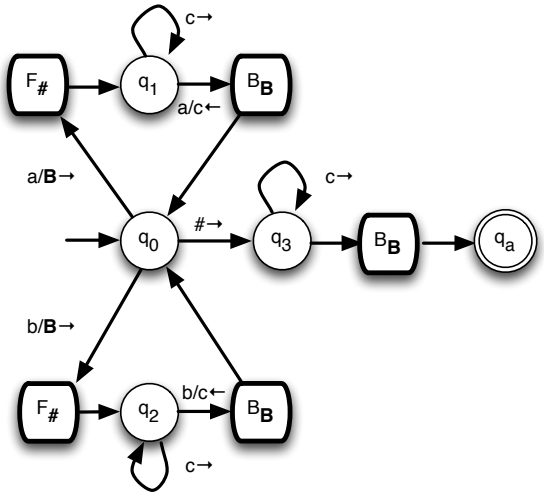
5 Carrés

Dans un premier temps, sauter cette section, et revenir dessus s'il reste du temps à la fin.

Un mot W est un *carré* s'il existe un mot w tel que $W = ww$. L'ensemble des carrés sur un alphabet Σ n'est pas un langage context-free. Cependant, il est possible de les reconnaître par une machine de Turing effectuant des aller-retours assez similaires à ceux de la question précédente.

Question 5.1. *Écrire une machine de Turing reconnaissant les mots de la forme $w\#w$ où w est un mot de l'alphabet $\Sigma = \{a, b\}$.*

Solution : La machine ci-dessous efface progressivement la partie du mot qui précède le caractère # en en remplaçant les lettres par des blancs, tout en remplaçant les lettres correspondantes après le # par une lettre $c \notin \Sigma$. Il ne reste plus à la fin qu'à vérifier qu'il ne reste pas de lettres en trop.

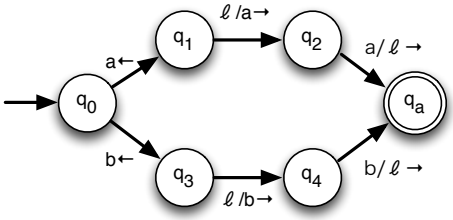


□

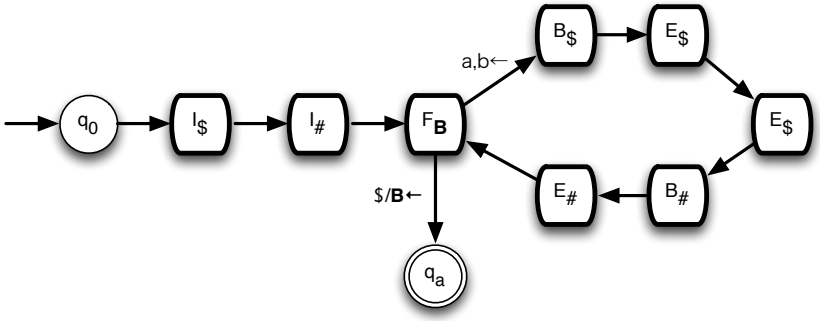
Question 5.2 (Plus difficile)**.** *Écrire une machine de Turing qui n'accepte que des mots de longueur paire, et laisse sur son ruban le mot qui lui a été donné en entrée avec un caractère* $\# \notin \Sigma$ *inséré entre ses deux moitiés.*

Solution : Une solution consiste à insérer deux lettres # et \$ en début de mot, puis à les décaler vers la fin du mot en faisant avancer la lettre \$ de deux lettres en deux lettres alors que la lettre # n'avance que d'une lettre à chaque étape. Lorsque \$ arrive en fin de mot, cela teste la parité, et il n'y a plus qu'à l'effacer, le # étant alors automatiquement à sa place.

Il est commode de définir une machine intermédiaire E_ℓ, qui prend un mot de la forme ℓmw avec ℓ et m dans $\Sigma \cup \{\$, \#\}$ et renvoie $m\ell w$ en laissant la tête de lecture sur le ℓ. La machine rejette si m est un blanc :



À partir de là, le parcours peut s'écrire :



□

Question 5.3. *Combiner ces deux machines pour écrire une machine reconnaissant les carrés.*

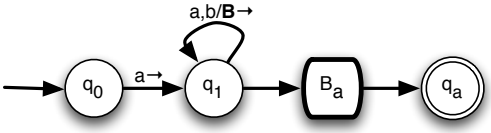
Solution : Il suffit de les mettre bout à bout. □

6 Arithmétique

Jusqu'ici les calculs se sont limités à des manipulations élémentaires sur des mots. Cependant, en réutilisant les machines précédentes, et en codant le k-uplet d'entiers $(n_1, \ldots, n_k)$ par le mot $a^{n_1+1}ba^{n_2+1}b\cdots a^{n_k+1}$, il est également possible de leur faire calculer beaucoup d'opérations arithmétiques. (On aurait pu aussi prendre des 0 et des 1, mais il sera plus facile de réutiliser les machines précédentes sur cet alphabet $\{a, b\}$.)

Question 6.1. *Écrire une machine de Turing prenant en entrée le codage d'un entier et s'arrêtant avec le codage de 0 (On dira qu'elle* calcule *0).*

Solution :



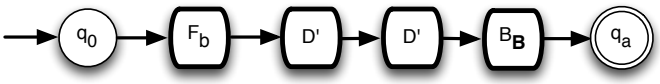
□

Question 6.2. *Écrire une machine de Turing qui calcule la fonction prenant en entrée un entier n et renvoyant $n + 1$.*

Solution : La machine est I_a. □

Question 6.3. *Écrire une machine de Turing qui calcule la fonction prenant en entrée deux entiers n et m et renvoyant $n + m$.*

Solution : Il faut enlever le b du milieu et l'un des a :



□

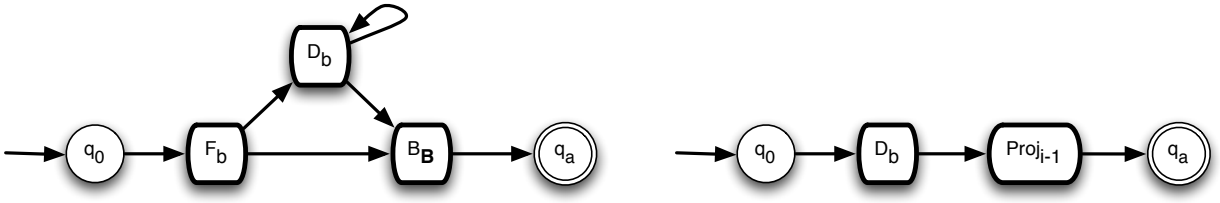
7 Complément : des machines de Turing pour les fonctions récursives primitives

L'objectif de cette dernière section est de montrer que, comme annoncé en première PC, les machines de Turing ont une expressivité suffisante pour calculer des fonctions récursives primitives.

7.1 Projection

Question 7.1. *Écrire une machine de Turing* Proj_i *qui calcule la fonction prenant en entrée un k-uplet d'entiers et renvoyant le i^e, avec $1 \leq i \leq k$.*

Solution : On distingue le cas de Proj_1 des autres, qui se définissent récursivement :

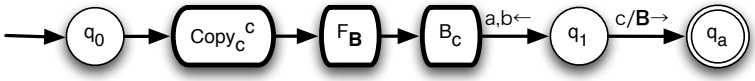


□

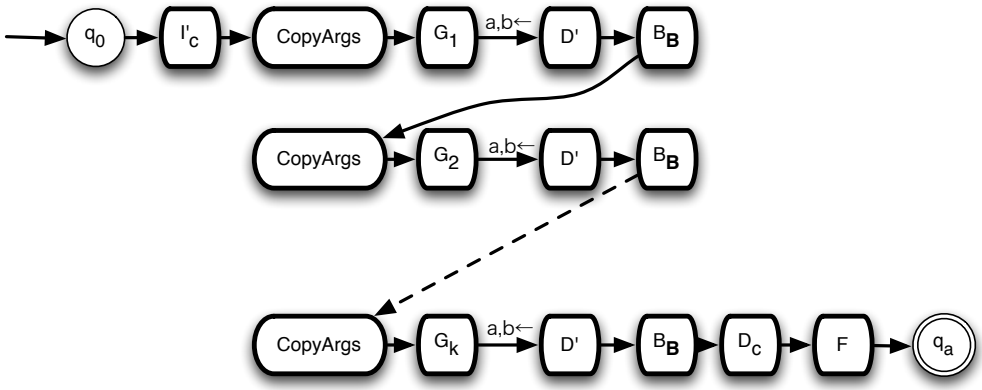
7.2 Composition

Question 7.2. *À partir de k machines* $\mathsf{G}_1, \ldots, \mathsf{G}_k$ *calculant des fonctions* $g_1, \ldots, g_k$ *de* $\mathbb{N}^n$ *dans* $\mathbb{N}$ *et d'une machine* F *calculant une fonction* f *de* $\mathbb{N}^k$ *dans* $\mathbb{N}$*, construire une machine calculant l'application* $(x_1, \ldots, x_n) \mapsto f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$.

Solution : L'idée est d'aller calculer les fonctions $g_1, \ldots, g_k$ dans cet ordre, de plus en plus à droite sur le ruban, en insérant un blanc pour que les machines G_i n'effacent pas ce qui a déjà été calculé. On utilise une machine auxiliaire CopyArgs pour recopier les arguments $(x_1, \ldots, x_n)$ derrière un blanc :



Cette machine prend en entrée un mot de la forme $w_1 c w_2$, où w_1 codera les arguments $x_1, \ldots, x_n$, elle le transforme d'abord en $w_1 c w_2 c w_1$, puis en $w_1 c w_2 \mathbf{B} w_1$ et laisse la tête de lecture positionnée sur le premier caractère du w_1 final.
La composition se résume alors ainsi :



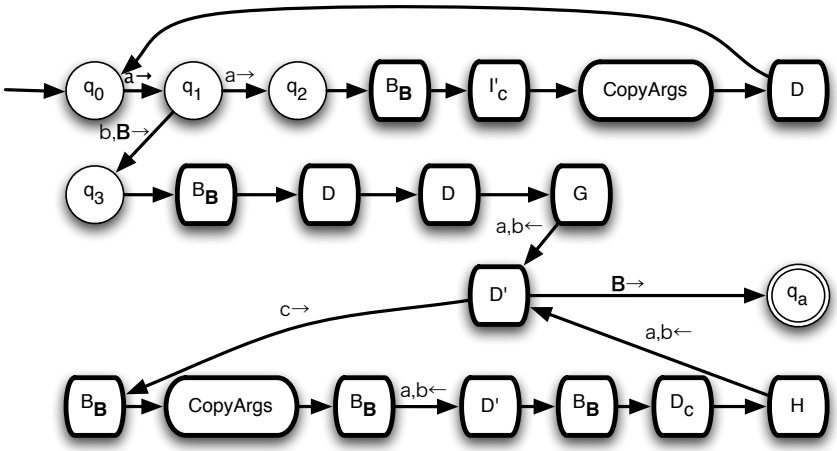
En tout début de calcul, la machine transforme l'argument w en wc, puis appelle la machine précédente pour obtenir $wc\mathbf{B}w$ avant d'appeler G_1 qui termine avec le mot $wc\mathbf{B}g_1(w)$, ensuite la tête de lecture est ramenée sur le blanc, qui est effacé par D' avant de ramener la tête de lecture tout au début du mot $wcg_1(w)$. La deuxième étape commence par transformer ce mot en $wcg_1(w)\mathbf{B}w$ avant de lancer l'appel à G_2. En dernière ligne, on a donc obtenu le mot $wcg_1(w) \cdots g_k(w)$. La machine D_c efface tout le début du mot jusqu'à c, et il ne reste plus qu'à appeler F. □

7.3 Récursivité

Question 7.3. *À partir d'une machine* G *calculant une fonction* g *de* $\mathbb{N}^{n-1}$ *dans* $\mathbb{N}$ *et d'une machine* H *calculant une fonction* h *de* $\mathbb{N}^{n+1}$ *dans* $\mathbb{N}$*, construire une machine calculant la fonction de* $\mathbb{N}^n$ *dans* $\mathbb{N}$ *définie récursivement par*

$$f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n),$$
$$f(x_1 + 1, x_2, \ldots, x_n) = h(f(x_1, \ldots, x_n), x_1, \ldots, x_n).$$

Solution :



Pour mémoriser l'état antérieur, cette machine procède comme la machine Comp en allant travailler plus à droite sur le ruban. La première opération consiste à tester si $x_1 = 0$. La première transition lit un premier a, et le branchement est effectué par q_1. La première ligne de la machine traite le cas où $x_1 \neq 0$, en transformant le mot w qui code $(x_1, \ldots, x_n)$ en $wc\mathbf{B}w'$ où w' code $(x_1 - 1, x_2, \ldots, x_n)$ grâce au D final. La machine repart alors en q_0. Lorsque la transition vers q_3 a finalement lieu, le ruban contient le mot

$$w_{x_1} c\mathbf{B} w_{x_1-1} c\mathbf{B} \cdots c\mathbf{B} w_0 c,$$

où w_i code $(i, x_2, \ldots, x_n)$, et la tête de lecture est sur le premier caractère du x_2 de w_0 (ou sur $\mathbf{B}$ si $n = 1$). La deuxième ligne de la machine calcule alors $g(x_2, \ldots, x_n)$ après avoir effacé $x_1 = 0$.

Ensuite, il faut aller dépiler les valeurs précédentes de x_n. Lorsque la machine arrive à la sortie de la machine D′ de la 3e ligne et que le calcul n'est pas terminé, le ruban contient le mot

$$w_{x_1} c\mathbf{B} w_{x_1-1} c\mathbf{B} \cdots c\mathbf{B} w_i c v,$$

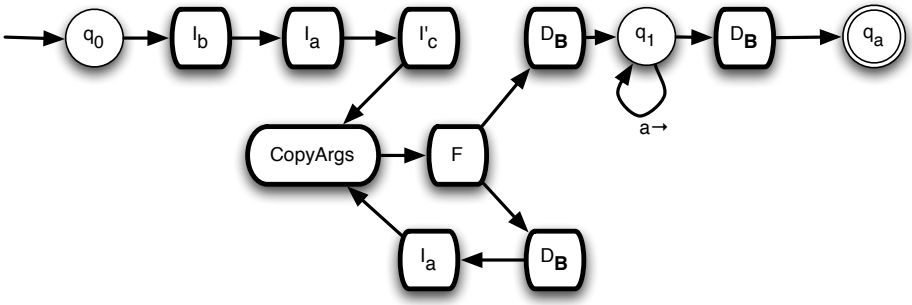
où v code $f(i - 1, x_2, \ldots, x_n)$. La dernière ligne transforme alors la portion $w_i cv$ en $w_i cv w_i$ puis $v w_i$, ce qui code $f(i - 1, x_2, \ldots, x_n), x_2, \ldots, x_n$ et permet d'appeler H, de sorte qu'à la sortie de la ligne, $w_i cv$ est remplacé par $f(i, x_2, \ldots, x_n)$. Ceci permet de recommencer tant qu'il reste un c à gauche, c'est-à-dire tant que la pile n'est pas vide. Ensuite, il y a un blanc à gauche et le calcul est terminé. □

7.4 Minimisation non-bornée

Les machines de Turing ont donc au moins l'expressivité des fonctions récursives primitives. Le sujet de la première PC se concluait en mentionnant que ce qui manquait aux fonctions primitives récursives pour avoir toute l'expressivité des machines de Turing était la minimisation non-bornée (correspondant au 'while' des langages de programmation). C'est par contre une opération très facile à implanter sur une machine de Turing.

Question 7.4. *À partir d'une machine* F *calculant une fonction* $f : \mathbb{N}^n \to \{0, 1\}$*, construire une machine, qui prend en entrée le codage de* $(x_2, \ldots, x_n)$ *et s'arrête avec sur son ruban le codage du plus petit* $y \in \mathbb{N}$ *tel que* $f(y, x_2, \ldots, x_n) = 1$ *s'il en existe 1, et boucle sinon.*

Solution :



Cette machine commence par insérer le codage de 0 en tête de ruban, puis elle ajoute un c à la fin de son entrée ainsi modifiée. Ensuite, elle utilise CopyArgs pour recopier le codage de $(0, x_2, \ldots, x_n)$ après un espace au-delà de ce c. La machine F est alors invoquée sur cet argument. Si elle réussit, alors le résultat est effacé, puis la tête de lecture est amenée juste après la fin de y et le reste du ruban est effacé. Si elle échoue, alors le résultat est effacé aussi, puis y est incrémenté (par I_a) et le calcul reprend. □