École polytechnique — X2022 — INF412 Fondements de l'informatique

Machines de Turing

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.

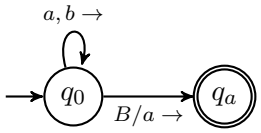
Définition. Le point de départ sera la définition du polycopié, rappelée ici :
Une machine de Turing est un octuplet $M = (Q, \Sigma, \Gamma, B, \delta, q_0, q_a, q_r)$ *où*

- Q *est l'ensemble fini des états ;*
- Σ *est un alphabet fini ;*
- $\Gamma \supset \Sigma$ *est l'alphabet de travail fini ;*
- $B \in \Gamma \setminus \Sigma$ *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.*

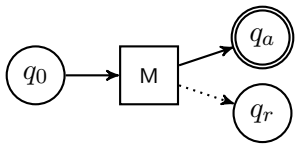
Notations. Une machine de Turing sera représentée par un graphe dont les sommets sont les états, les arcs représentant les transitions. Une transition $\delta(q_1, \ell) = (q_2, m, d)$, où q_1, q_2 sont dans Q, les lettres ℓ, m dans Γ et d est une direction sera représentée par une étiquette ℓ/md sur 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 qui ne sont pas représentés pointent vers l'état de refus. (Lorsque la fonction de transition est partielle, ceci revient à considérer que les cas où la machine s'arrête sans avoir accepté sont des cas de refus.)

Par exemple, voici une représentation de la machine de Turing acceptant les mots n'utilisant que les lettres a et b de Σ, et ajoutant un a à la fin du mot :



Sous-programmes. On s'efforcera de programmer les machines de Turing de façon *modulaire*, en réutilisant des machines déjà écrites. Pour cela, une machine M sera symbolisée de la forme ci-dessous, ce qui permet de les utiliser pour définir d'autres machines :



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.

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 sous-routines suivantes seront donc 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.*

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 similaires aux classiques automates finis déterministes, avec une convention différente sur les états d'acceptation. 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*.*

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* $\{(,)\}$*. 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.*

4 Effacements et Insertions

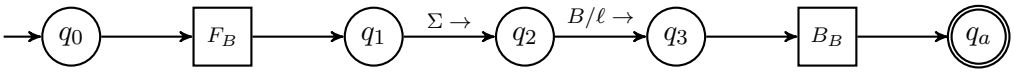
Question 4.1. *Écrire deux machines* D *et* D' *(pour* delete*) qui prennent en entrée un mot* ℓw *où* $\ell \in \Sigma = \{a, b\}$ *et* w *est un mot sur* Σ*, et s'arrêtent avec le mot* w *sur leur ruban. Il faut décaler d'une lettre vers la gauche toutes les lettres de* w*. La machine* D *termine avec 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, contrairement aux autres machines de ce sujet, elle 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).*

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, en décalant la suite du mot comme à la question précédente.*

À partir de la question qui suit, pour une meilleure réutilisation dans la suite, lorsqu'une machine de Turing s'arrêtera sur son état d'acceptation, sa tête de lecture sera d'abord revenue à sa position initiale.

Question 4.3. *Écrire une troisième machine I_ℓ, qui s'arrête avec ℓw sur son ruban, sans écrire sur les cases blanches à gauche de w.*

Avec les machines précédentes, il est facile d'écrire deux autres machines I'_ℓ (à droite ci-dessous) et I''_ℓ (à gauche) qui prennent en entrée un mot w sur l'alphabet $\{a, b\}$, et s'arrêtent avec respectivement les mots $w\ell$, $w\ell$ sur leurs rubans. La différence entre I'_ℓ et I''_ℓ est que cette dernière laisse sa tête de lecture sur le caractère suivant $w\ell$.



5 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 5.1. *Écrire une machine de Turing prenant en entrée le codage d'un k-uplet d'entiers et s'arrêtant avec le codage de 0.*

Question 5.2. *Écrire une machine de Turing qui calcule la fonction prenant en entrée un entier n et renvoyant $n + 1$.*

Question 5.3. *Écrire une machine de Turing qui calcule la fonction prenant en entrée deux entiers n et m et renvoyant $n + m$.*

6 Carrés

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

Question 6.2 (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.*

Question 6.3. *Combiner ces deux machines pour écrire une machine reconnaissant les carrés.*

7 Copies

Question 7.1. *Écrire une machine Copy_ℓ^m 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 B dans $\Gamma \setminus \Sigma$, et s'arrête avec le mot $w_1\ell w_2 m w_1$ sur son ruban.*

8 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. Comme ci-dessus, un k-uplet d'entiers $(n_1, \dots, n_k)$ est codé par le mot $a^{n_1+1}ba^{n_2+1}b\cdots a^{n_k+1}$.

8.1 Projection

Question 8.1. *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* $1 \leq i \leq k$*.*

8.2 Composition

Question 8.2. *À partir de* k *machines* $\mathsf{G}_1, \dots, \mathsf{G}_k$ *qui calculent des fonctions* $g_1, \dots, g_k$ *de* $\mathbb{N}^n$ *dans* $\mathbb{N}$ *sans écrire à gauche de leur entrée, et d'une machine* F *calculant une fonction* f *de* $\mathbb{N}^k$ *dans* $\mathbb{N}$*, construire une machine calculant l'application*

$$(x_1, \dots, x_n) \mapsto f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n)).$$

8.3 Récursivité

Question 8.3. *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*

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

8.4 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 8.4. *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* $(x_2, \dots, x_n)$ *et s'arrête avec sur son ruban le codage du plus petit* $y \in \mathbb{N}$ *tel que* $f(y, x_2, \dots, x_n) = 1$ *s'il en existe 1, et boucle sinon.*