AUFORT William

Sujet Choisi : Machines de Turing : Applications (leçon 913)

Références : Les 2 principaux : { Sipser, Introduction to the theory of computation ; Arora, Barak, Computational Complexity, a modern approach }
+ quelques trucs dans Floyd, Le langage des machines

913

Objectif principal : Modéliser mathématiquement un calcul, un algorithme.

I] Machines de Turing

a) Définitions et premiers exemples

Déf 1 : Une machine de Turing est un 7-uplet $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ où :
- Q est un ensemble fini d'éléments appelés états
- Σ est un ensemble fini appelé alphabet d'entrée
- Γ est un ensemble fini appelé alphabet de bande, qui contient un symbole $\sqcup \notin \Sigma$ (le blanc).
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la fonction de transition
- $(q_0, q_{accept}, q_{reject}) \in Q$ sont respectivement les états initial, acceptant et terminant ($q_{reject} \neq q_{accept}$).

Déf 2 : Une configuration d'une machine de Turing est la donnée d'un état q et de deux mots $u, v \in \Gamma^*$. q est l'état actuel de la machine, le ruban contient uv, la tête de lecture est sur la première lettre de v.

Ex 3 : [q] $\to$ |1|0|1|1|0|1|⊔|⊔|... est la configuration $101q101$

Remarque 4 : δ permet de passer d'une configuration à une autre.

Ex 5 : Si $\delta(q, 1) = (q', 0, \rightarrow)$ alors la configuration suivant $101q101$ est $1010q'01$.

Déf 6 : On dit qu'une machine M accepte un mot $w \in \Sigma^*$ s'il existe une suite finie de configuration $C_1 \dots C_n$ telle que C_1 est la configuration initiale de M sur w, C_n est une configuration acceptante (ie $q = q_{accept}$), et la configuration suivant C_i est C_{i+1}.

Déf 7 : Le langage reconnu/accepté par une machine de Turing $L(M)$ est l'ensemble des w tels que M accepte w.

Remarque 8 : Trois issues sont possibles pour w : la machine peut accepter, rejeter, ou boucler.

Déf 9 : M est un décideur si M s'arrête sur toute entrée.

Déf 10 : Un langage est décidable si un décideur reconnaît ce langage.

Remarque 11 : Représenter une machine à la manière d'un automate (transitions étiquetées selon δ) est rapidement pénible (cf annexe 1).

Remarque 12 : On peut donner une description informelle (pseudo code) pour décrire le fonctionnement d'une machine de Turing.

Ex 13 : Pour décider $L_1 = \{ w\#w ; w \in \{0,1\}^* \}$,
1) Regarder tout le mot pour voir s'il y a un seul symbole #. Si ce n'est pas le cas, rejeter.
2) Faire des allers-retour des deux côtés de # pour voir si les lettres sont égales. Si oui, les marquer d'1 x. Rejeter sinon.
3) Quand tous les symboles de gauche ont été marqués, s'il ne reste plus de symbole à droite, accepter. Sinon rejeter.

Ex 14 : $L_2 = \{ a^n b^n c^n, n \in \mathbb{N} \}$ peut être décidé par une machine de Turing.

Rem 15 : L_2 n'est pas algébrique, mais il est décidable.

b) Autres modèles, équivalences

Déf 16 : Une machine de Turing à plusieurs rubans est une machine où $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{\leftarrow, \rightarrow\}^k$, où k est le nombre de rubans, $k \geq 2$.

Prop 17 : Toute machine à k rubans est équivalente à une machine classique à 1 ruban.

Rem 18 : Il faut une convention pour que la machine de Turing de la déf 1 soit à 1 direction (symbole de début de bande $\triangleright$, condition sur δ...). On peut définir alors une machine à ruban biinfini (infini des deux côtés).

Prop 19 : Toute machine à ruban biinfini est équivalente à une machine à un ruban unidirectionnel.

Déf 20 : Une machine de Turing non déterministe est une machine où

$\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{\leftarrow, \rightarrow\})$. (plusieurs choix pour la transition.

Prop 21 : Toute machine de Turing non déterministe est équivalente à une machine de Turing (déterministe).

Def 22 : Une machine RAM est la donnée d'un système de contrôle (graphe dont les sommets sont les états de la machine, et les arêtes sont étiquetées par les opérations à effectuer) et d'une mémoire RAM (mémoire statique et registres en nombre fini)

Prop 23 On peut simuler une RAM avec une machine de Turing à plusieurs rubans.

parler des automates aussi

Remarque 24 Autres possibilités : alphabet minimal $\{0, 1, \sqcup\}$, machine pouvant rester sur place, ...

Remarque 25 : Nous n'avons pas parlé ici du temps nécessaire pour effectuer ces simulations. → domaine de la complexité (cf III).

Remarque 26 : Thèse de Church-Turing : Tout système de calcul physiquement réalisable peut être simulé par une machine de Turing. Nous venons d'en voir des exemples.

Ex 27 : fonctions récursives, λ calcul ...

II] Décidabilité / Indécidabilité

Remarque 28 : Si une machine de Turing représente tout ce qui peut être calculé, y a-t-il des fonctions non calculables ?

a) Définitions, premiers exemples

Def 29 Un langage est récursivement énumérable s'il est reconnu par une machine de Turing (cf déf 7). RE est l'ensemble de ces langages.

Déf 30 : Un langage est récursif s'il est décidable. L'ensemble de ces langages R.

Déf 31 : CO-RE est l'ensemble des langages L tels que $\bar{L} \in RE$.

Prop 32. RE, CO-RE et R sont stables par union et intersection.

Prop 33. R est stable par complémentaire.

Prop 34 : Il existe des langages qui ne sont pas récursivement énumérables.

Ex 35 : Soit $A_{MT} = \{\langle M, w\rangle$, M est une machine de Turing qui accepte $w\}$, où $\langle M, w\rangle$ est une chaîne de caractères qui encode la machine de Turing M et w. A_{MT} représente le problème de l'acceptation

Prop 36 A_{TM} est indécidable, mais est récursivement énumérable.

Prop 37 : $L \in R \iff L \in RE$ et $\bar{L} \in RE$

Corollaire 38 $\overline{A_{TM}}$ n'est pas récursivement énumérable.

Ex 39 $AU_{MT} = \{\langle M\rangle$, M est une machine de Turing qui s'arrête sur toute entrée$\}$ est le problème de l'arrêt universel

Prop 40 AU_{MT} n'est ni décidable ni récursivement énumérable.

Remarque 41 Ces problèmes sont beaucoup utilisés pour montrer que d'autres problèmes sont indécidables.

b) Technique de réduction

Remarque 42 : L'idée générale de la réduction est la suivante : pour prouver qu'un langage L est indécidable, on peut par l'absurde utiliser un décideur pour décider une autre langage L' connu comme étant indécidable. On dit qu'on réduit L à L'.

Prop 43 : Le problème de l'arrêt $HALT = \{\langle M, w\rangle / M$ s'arrête sur $w\}$ est indécidable, (par réduction vers le problème A_{TM}).

Prop 44 : Le problème du langage vide $EMPTY = \{\langle M\rangle / L(M) = \emptyset\}$ est indécidable (par réduction vers A_{TM}).

Remarque 45 : On a un résultat plus général sur les propriétés d'une machine de Turing :

Théorème 46 (de Rice) Si P est un langage contenant des codes de machines de Turing non trivial ($\exists \langle M_1\rangle \in P$, $\exists \langle M_2\rangle \notin P$), tel que si $L(M_1) = L(M_2)$, alors $\langle M_1\rangle \in L \iff \langle M_2\rangle \in L$, alors P est indécidable.

c) Exemples d'utilisation :

Prop 47 : L'ensemble des grammaires algébriques G telles que $L(G) = \Sigma^*$ est indécidable.

Remarque 48 : Le même problème pour les expressions rationnelles est, quant à lui, décidable.

913

Parler aussi du Speed-Up linéaire → justifie O(..)

DEV1

Ex 50 Une instance du Problème de Correspondance de Post (PCP) est un ensemble de dominos $\left[\frac{t_1}{b_1}\right], \dots \left[\frac{t_k}{b_k}\right]$. On veut trouver des indices $i_1 \dots i_\ell$ tels que $t_{i_1} \dots t_{i_\ell} = b_{i_1} \dots b_{i_\ell}$.

Prop 51: PCP est indécidable.

Prop 52: Le problème de terminaison d'un système de réécriture est indécidable.

III] Complexité algorithmique

Idée générale: Machine de Turing pour mesurer les complexités en ressources (temps, espace).

a) Complexités spatiales et temporelles

Déf 53: On dit que M calcule $f: \{0,1\}^* \to \{0,1\}^*$ en temps $T(n)$ si $\forall x$, sur l'entrée x, M renvoie $f(x)$ au bout d'au plus $T(|x|)$ étapes.

Remarque 54 On définit de même la complexité en espace.

Déf 55: $DTIME(T(n)) = \{L$ langage décidé par une machine M en temps $c.T(n)$ pour $c>0\}$

Déf 56: $P = \bigcup_{c \geq 1} DTIME(n^c)$.

Remarque 57: Interprétation: ce sont les problèmes résolvables en temps polynomial par rapport à la taille de leur entrée.

Remarque 58: On traite ici de problèmes de décision. Si on a un problème, on peut toujours se ramener à un problème de décision.

Ex 59: multiplication entière: on associe $L = \{\langle x, y, i \rangle$ / le $i$ème bit de xy est $1\}$, qui correspond à un problème de décision, avec $L \in P$.

Déf 60: On définit de même qu'en Déf 55 $NTIME(T(n))$ avec des machines non déterministes, et $NP = \bigcup_{c \geq 1} NTIME(n^c)$.

Déf 61: $f: \mathbb{N} \to \mathbb{N}$ est constructible en temps si $x \mapsto f(|x|)$ peut être calculée en temps $f(|x|)$.

Ex 62 log, tout polynôme, etc... sont calculables en temps.

Théorème 63 (Hiérarchie temporelle) Si f et g sont constructibles en temps:

1) $f \log f = o(g) \implies DTIME(f(n)) \subsetneq DTIME(g(n))$
2) $f(n+1) = o(g(n)) \implies NTIME(f(n)) \subsetneq NTIME(g(n))$.

Déf 64: On définit de même qu'en Déf 55 $SPACE(f(n))$, $NSPACE(f(n))$, PSPACE et NPSPACE.

Théorème 65 (Hiérarchie spatiale) Si f et g sont constructibles en espace, et $f(n) = o(g(n))$, alors $SPACE(f(n)) \subsetneq SPACE(g(n))$.

DEV2

Théorème 66 (Savitch): $NSPACE(f(n)) \subseteq SPACE(f(n)^2)$ si $f(n) \geq n$.

Corollaire 67: PSPACE = NPSPACE.

b) Complétude

Déf 68 Si L et L' sont 2 langages, on note $L \leq_p L'$ s'il existe $f: \{0,1\}^* \to \{0,1\}^*$ calculable en temps polynomial telle que $\forall x$, $x \in L \iff f(x) \in L'$

Déf 69 L' est NP-difficile si $\forall L \in NP$, $L \leq_p L'$. On dit que L' est NP-Complet si en plus $L' \in NP$.

Rem 70 On utilise en général cette caractérisation pour NP:

Thm 71 $L \in NP \iff \exists P$ polynôme et M une machine de Turing tels que $x \in L \iff \exists u \in \{0,1\}^{P(x)}$ (appelé certificat) / $M(x,u) = 1$.

Ex 72 SAT et 3-SAT sont NP-Complets (Cook-Levin)

Remarque 73 Idée générale: NP-Complet = le plus difficile dans NP.

Déf 74: On définit de même PSPACE-Complet, PSPACE-difficile.

Ex 75: Le problème d'universalité d'un langage rationnel est PSPACE-difficile.

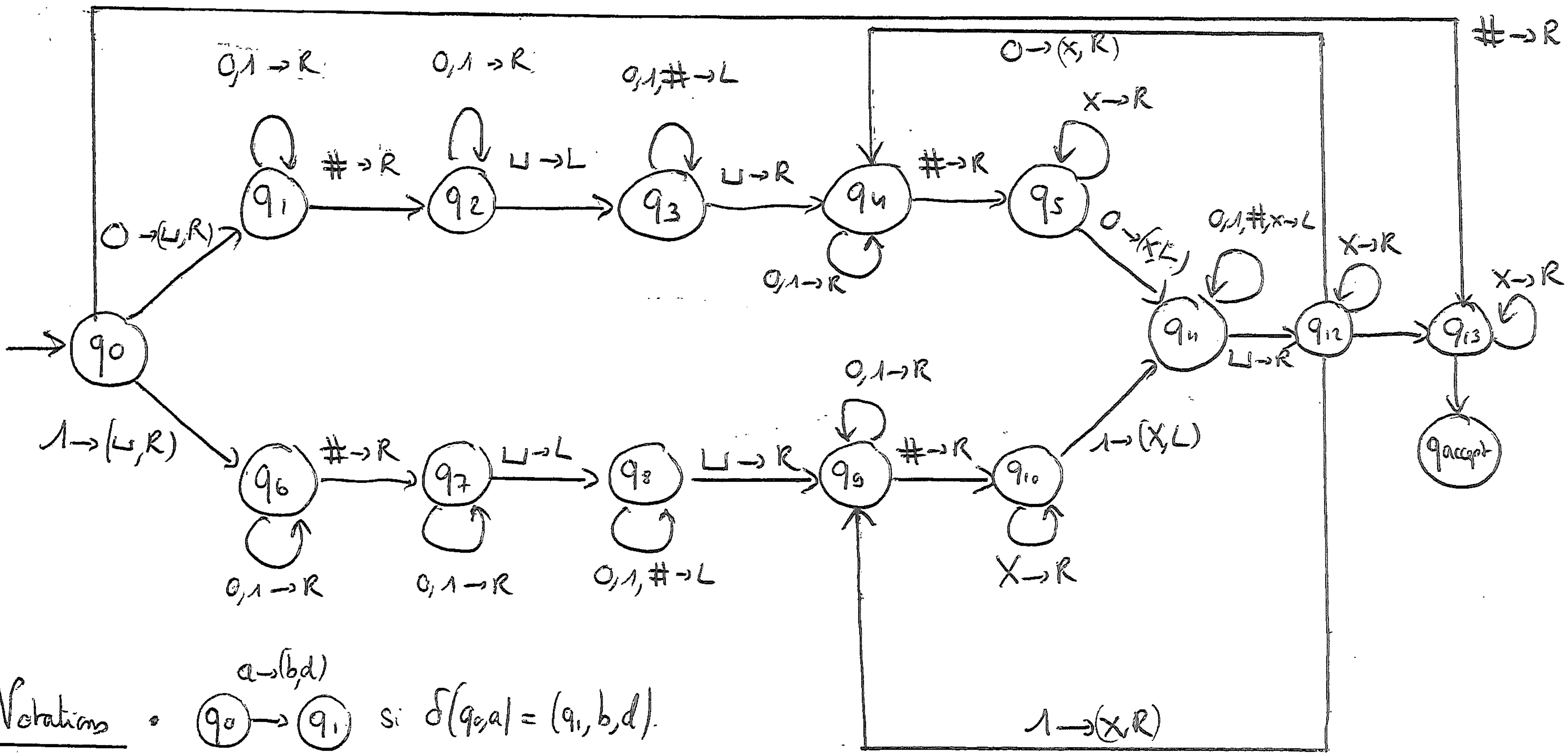
DEV3

Prop 76 Si $L \in NP$, L' NP-complet, $L' \leq_p L$, alors L est NP-Complet.

Remarque 77 Pour montrer qu'un problème est NP-Complet, il suffit alors de le réduire à un problème connu pour être NP-complet.

913

Annexe 1   Diagramme d'état d'une machine de Turing décidant $L_1 = \{w\#w,\ w \in \{0,1\}^*\}$ (cf Ex 13).



Notations

- $q_0 \xrightarrow{a \to (b,d)} q_1$ si $\delta(q_0,a) = (q_1,b,d)$.
- $q_0 \xrightarrow{a \to d} q_1$ si $\delta(q_0,a) = (q_1,a,d)$.

- L = « ← »   (pour clarifier le dessin, qui contient déjà des flèches)
- R = « → »