Fondements de l'informatique
Logique, modèles, et calculs

Chapitre: Autres modèles de calculs

Cours INF412

de l'Ecole Polytechnique

Olivier Bournez

bournez@lix.polytechnique.fr

Version du 16 juillet 2023

Autres modèles de calculs

1 Machines RAM

Le modèle de la machine de Turing peut paraître extrêmement rudimentaire. Il n'en demeure pas extrêmement puissant, et capable de capturer la notion de calculable en informatique.

L'objectif de cette section est de se persuader de ce fait : tout ce qui est programmable par un dispositif de calcul informatique digital actuel peut se simuler par une machine de Turing. Pour cela, on va introduire un modèle très proche (le plus proche en fait que je connaisse) de la façon dont fonctionnent les processeurs actuels : le *modèle des machines RAM*.

1.1 Modèle des machines RAM

Le modèle des *machines RAM (Random Access Machine)* est un modèle de calcul qui ressemble beaucoup plus aux langages machine actuels, et à la façon dont fonctionnent les processeurs actuels.

Une machine RAM possède des registres qui contiennent des entiers naturels (nuls si pas encore initialisés). Chaque machine est supposée avoir un nombre infini de registres, indexés par les entiers naturels. Les instructions autorisées dépendent du processeur que l'on veut modéliser (ou de l'ouvrage que l'on consulte qui décrit ce modèle), mais elles incluent en général la possibilité de :

1. copier le contenu d'un registre dans un autre ;
2. faire un adressage indirect : récupérer/écrire le contenu d'un registre dont le numéro est donné par la valeur d'un autre registre ;
3. effectuer des opérations élémentaires sur un ou des registres, par exemple additionner 1, soustraire 1 ou tester l'égalité à 0 ;
4. effectuer d'autres opérations sur un ou des registres, par exemple l'addition, la soustraction, la multiplication, division, les décalages binaires, les opérations binaires bit à bit.

Dans ce qui suit, nous réduirons tout d'abord la discussion aux *SRAM (Successor Random Access Machine)* qui ne possèdent que des instructions des types 1., 2. et 3. Nous verrons qu'en fait cela ne change pas grand chose, du moment que chacune des opérations du type 4. se simule par machine de Turing (et c'est le cas de tout ce qui est évoqué plus haut).

1.2 Simulation d'une machine RISC par une machine de Turing

Nous allons montrer que toute machine RAM peut être simulée par une machine de Turing.

Pour aider à la compréhension de la preuve, nous allons réduire le nombre d'instructions des machines RAM à un ensemble réduit d'instructions (*RISC reduced instruction set* en anglais) en utilisant un unique registre x_0 comme accumulateur.

Définition 1 *Une machine RISC est une machine RAM dont les instructions sont uniquement de la forme :*

1. $x_0 \leftarrow 0$;
2. $x_0 \leftarrow x_0 + 1$;
3. $x_0 \leftarrow x_0 \ominus 1$;
4. ***if*** $x_0 = 0$ ***then*** *aller à l'instruction numéro* j ;
5. $x_0 \leftarrow x_i$;
6. $x_i \leftarrow x_0$;
7. $x_0 \leftarrow x_{x_i}$;
8. $x_{x_0} \leftarrow x_i$.

Clairement, tout programme RAM avec des instructions du type 1., 2. et 3. peut être converti en un programme RISC équivalent, en remplaçant chaque instruction par des instructions qui passent systématiquement par l'accumulateur x_0.

Exemple 1 *Par exemple : l'instruction* $x_i \leftarrow x_j$ *peut être remplacée par la suite des deux instructions* $x_0 \leftarrow x_j$ *et* $x_i \leftarrow x_0$.

Théorème 1 *Toute machine RISC peut être simulée par une machine de Turing.*

Démonstration: La machine de Turing qui simule la machine RISC possède 4 rubans. Les deux premiers rubans codent les couples (i, x_i) pour x_i non nul. Le troisième ruban code l'accumulateur x_0 et le quatrième est un ruban de travail.

Plus concrètement, pour un entier i, notons $\langle i \rangle$ son écriture en binaire. Le premier ruban code un mot de la forme

$$\ldots \mathbf{BB} \langle i_o \rangle \mathbf{B} \langle i_1 \rangle \cdots \mathbf{B} \ldots \langle i_k \rangle \mathbf{BB} \cdots .$$

Le second ruban code un mot de la forme

$$\ldots \mathbf{BB} \langle x_{i_o} \rangle \mathbf{B} \langle x_{i_1} \rangle \cdots \mathbf{B} \ldots \langle x_{i_k} \rangle \mathbf{BB} \cdots$$

Les têtes de lecture des deux premiers rubans sont sur le deuxième **B**. Le troisième ruban code $\langle x_0 \rangle$, la tête de lecture étant tout à gauche. Appelons *position standard* une telle position des têtes de lecture.

La simulation est décrite pour trois exemples. Notre lecteur pourra compléter le reste.

1. $x_0 \leftarrow x_0 + 1$: on déplace la tête de lecture du ruban 3 tout à droite jusqu'à atteindre un symbole **B**. On se déplace alors d'une case vers la gauche, et on remplace les **1** par des **0**, en se déplaçant vers la gauche tant que possible. Lorsqu'un **0** ou un **B** est trouvé, on le change en **1** et on se déplace à gauche pour revenir en position standard.
2. $x_{23} \leftarrow x_0$: on parcourt les rubans 1 et 2 vers la droite, bloc délimité par **B** par bloc, jusqu'à atteindre la fin du ruban 1, ou ce que l'on lise un bloc **B10111B** (**10111** correspond à 23 en binaire).

   Si la fin du ruban 1 a été atteinte, alors l'emplacement 23 n'a jamais été vu auparavant. On l'ajoute en écrivant **10111** à la fin du ruban 1, et on recopie le ruban 3 (la valeur de x_0) sur le ruban 2. On retourne alors en position standard.

   Sinon, c'est que l'on a trouvé **B10111B** sur le ruban 1. On lit alors $\langle x_{23} \rangle$ sur le ruban 2. Dans ce cas, il doit être modifié. On fait cela de la façon suivante :

   (a) On copie le contenu à droite de la tête de lecture numéro 2 sur le ruban 4.

   (b) On copie le contenu du ruban 3 (la valeur de x_0) à la place de x_{23} sur le ruban 2.

   (c) On écrit **B**, et on recopie le contenu du ruban 4 à droite de la tête de lecture du ruban 2, de façon à restaurer le reste du ruban 2.

   (d) On retourne en position standard.
3. $x_0 \leftarrow x_{x_{23}}$: En partant de la gauche des rubans 1 et 2, on parcourt les rubans 1 et 2 vers la droite, bloc délimité par **B** par bloc, jusqu'à atteindre la fin du ruban 1, ou ce que l'on lise un bloc **B10111B** (**10111** correspond à 23 en binaire).

   Si la fin du ruban 1 a été atteinte, on ne fait rien, puisque x_{23} vaut 0 et le ruban 3 contient déjà $\langle x_0 \rangle$.

   Sinon, c'est que l'on a trouvé **B10111B** sur le ruban 1. On lit alors $\langle x_{23} \rangle$ sur le ruban 2, que l'on recopie sur le ruban 4. Comme ci-dessus, on parcourt les rubans 1 et 2 en parallèle jusqu'à trouver **B**$\langle x_{23} \rangle$**B** où atteindre la fin du ruban 1. Si la fin du ruban 1 est atteinte, alors on écrit **0** sur le ruban 3, puisque $x_{x_{23}} = x_0$. Sinon, on copie le bloc correspondant sur le ruban 1 sur le ruban 3, puisque le bloc sur le ruban 2 contient $x_{x_{23}}$, et on retourne en position standard.

□

1.3 Simulation d'une machine RAM par une machine de Turing

Revenons sur le fait que nous avons réduit l'ensemble des opérations autorisées sur une machine *RAM* aux instructions du type 1., 2. et 3. En fait, on observera que l'on peut bien gérer toutes instructions du type 4., du moment que l'opération sous-jacente peut bien se calculer par machine de Turing : toute opération $x_0 \leftarrow x_0$ ''opération'' x_i peut être simulée comme plus haut, dès que ''opération'' correspond à une opération calculable.

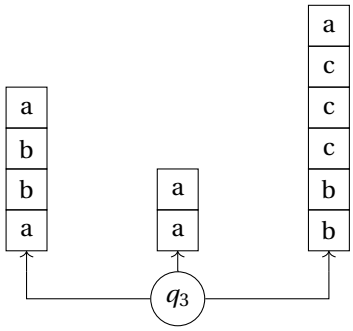

FIGURE 1 – Une machine à 3 piles.

2 Modèles rudimentaires

Le modèle des machines des Turing est extrêmement rudimentaire. On peut toutefois considérer des modèles qui le sont encore plus, et qui sont toutefois capable de les simuler.

2.1 Machines à $k \geq 2$ piles

Une *machine à k piles*, possède un nombre fini k de piles $r_1, r_2, \cdots, r_k$, qui correspondent à des piles d'éléments de Σ. Les instructions d'une machine à piles permettent seulement d'empiler un symbole sur l'une des piles, tester la valeur du sommet d'une pile, ou dépiler le symbole au sommet d'une pile.

Si l'on préfère, on peut voir une pile d'éléments de Σ comme un mot w sur l'alphabet Σ. Empiler ($push$) le symbole $a \in \Sigma$ correspond à remplacer w par aw. Tester la valeur du sommet d'une pile (top) correspond à tester la première lettre du mot w. Dépiler (pop) le symbole au sommet de la pile correspond à supprimer la première lettre de w.

Théorème 2 *Toute machine de Turing peut être simulée par une machine à* 2 *piles.*

Démonstration: Selon la formalisation de la page 110, une configuration d'une machine de Turing correspond à $C = (q, u, v)$, où u et v désignent le contenu respectivement à gauche et à droite de la tête de lecture du ruban i. On peut voir u et v comme des piles : voir la figure 2. Si l'on relit attentivement la formalisation page 110, on observe que les opérations effectuées par le programme de la machine de Turing pour passer de la configuration C a sa configuration successeur C' correspondent à des opérations qui se codent trivialement par des $push$, pop, et top : on peut donc construire une machine à 2 piles, chaque pile codant u ou v (le contenu à droite et à gauche de chacune des têtes de lecture), et qui simule étape par étape la machine de Turing. Par exemple, déplacer la tête à droite, consiste à lire le sommet de la seconde pile ($a \leftarrow top(2)$), et à empiler ce symbole a sur la premiere pile

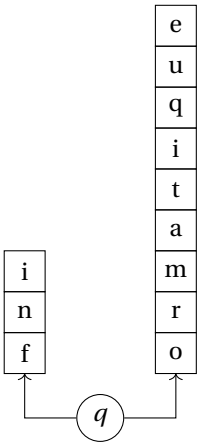

FIGURE 2 – La machine de Turing de l'exemple 7.3 vue comme une machine à 2-piles.

($push(1,a)$), et faire un dépilement sur la seconde pile ($pop(2)$). Déplacer la tête vers la gauche, consiste à lire le sommet de la premiere pile ($a \leftarrow top(1)$), empiler ce symbole a sur sur la seconde pile ($push(2,a)$) et faire un dépilement sur la premiere pile ($pop(1)$). Changer le symbole en face de la tête de lecture en le symbole a coniste à dépiler la seconde pile ($pop(2)$) , et empiler le symbole a sur la seconde pile ($push(2,a)$). □

2.2 Machines à compteurs

Nous introduisons maintenant un modèle encore plus rudimentaire : une *machine à compteurs* possède un nombre fini k de compteurs $r_1, r_2, \cdots, r_k$, qui contiennent des entiers naturels. Les instructions d'une machine à compteur permettent seulement de tester l'égalité d'un des compteurs à 0, d'incrémenter un compteur ou de décrémenter un compteur. Tous les compteurs sont initialement nuls, sauf celui codant l'entrée.

Remarque 1 *Les machines à compteurs sont généralement considérées comme des machines qui calculent des fonctions sur les entiers, ou comme reconnaissant des sous-ensembles de n-uplets d'entiers. Si l'on veut calculer sur les mots sur un alphabet, disons sur l'alphabet* $\Sigma = \{0,1\}$*, cela nécessite de coder les mots comme des entiers. Par exemple, on peut considérer chaque mot sur cet alphabet comme l'écriture en binaire d'un certain entier.*

Remarque 2 *On considère dans ce document des machines qui soit acceptent, ou bouclent. Le refus est encodé dans les simulations qui suivent par le fait que la machine ne s'arrête pas. Il serait possible de considérer des machines à compteur avec des*

instructions $Accepte$ et $Refuse$, pour simuler de façon plus fine l'acceptation ou le refus d'une machine de Turing.

Remarque 3 *C'est donc une machine RAM, mais avec un nombre très réduit d'instructions, et un nombre fini de registres.*

Plus formellement, toutes les instructions d'une machine à compteurs sont d'un des 4 types suivants.

— Inc(c, j) : on incrémente le compteur c puis on va à l'instruction j.
— Decr(c, j) : on décrémente le compteur c puis on va à l'instruction j.
— IsZero(c, j, k) : on teste si le compteur c est nul et on va à l'instruction j si c'est le cas, et à l'instruction k sinon.
— Halt : on arrête le calcul.

Par exemple, le programme suivant avec 3 compteurs

1. IsZero$(1,5,2)$
2. Decr$(1,3)$
3. Inc$(3,4)$
4. Inc$(3,1)$
5. Halt

transforme $(n,0,0)$ en $(0,0,2n)$: si l'on part avec $r_1 = n$, $r_2 = r_3 = 0$, alors lorsqu'on atteint l'instruction Halt, on a $r_3 = 2n$, et les autres compteurs à 0.

Exercice 1 *Pour chacune des conditions suivantes, décrire des machines à compteur qui atteignent l'instruction* Halt *si et seulement si la condition est initialement vérifiée.*

— $r_1 \geq r_2 \geq 1$
— $r_1 = r_2$ *ou* $r_1 = r_3$
— $r_1 = r_2$ *ou* $r_1 = r_3$ *ou* $r_2 = r_3$

Théorème 3 *Toute machine à k-piles peut être simulée par une machine à $k+1$ compteurs.*

Démonstration: L'idée est de voir une pile $w = a_1 a_2 \cdots a_n$ sur l'alphabet Σ de cardinalité $r-1$ comme un entier i en base r : sans perte de généralité, on peut voir Σ comme $\Sigma = \{0, 1, \cdots, r-1\}$. Le mot w correspond à l'entier $i = a_n r^{n-1} + a_{n-1} r^{n-2} + \cdots a_2 r + a_1$.

On utilise ainsi un compteur i pour chaque pile. Un $k+1$ème compteur, que l'on appellera *compteur supplémentaire*, est utilisé pour ajuster les compteurs et simuler chaque opération (empilement, dépilement, lecture du sommet) sur l'une des piles.

Dépiler correspond à remplacer i par i div r, où div désigne la division entière : en partant avec le compteur supplémentaire à 0, on décrémente le compteur i de r (en r étapes) et on incrémente le compteur supplémentaire de 1. On répète cette opération jusqu'à ce que le compteur i atteigne 0. On décrémente alors le compteur

supplémentaire de 1 en incrémentant le compteur i de 1 jusqu'à ce que le premier soit 0. A ce moment, on lit bien le résultat correct dans le compteur i.

Empiler le symbole a correspond à remplacer i par $i*r+a$: on multiplie d'abord par r : en partant avec le compteur supplémentaire à 0, on décrémente le compteur i de 1 et on incrémente le compteur supplémentaire de r (en r étapes) jusqu'à ce que le compteur i soit à 0. On décrémente alors le compteur supplémentaire de 1 en incrémentant le compteur i de 1 jusqu'à ce que le premier soit 0. A ce moment, on lit $i*r$ dans le compteur i. On incrémente alors le compteur i de a (en a incrémentations).

Lire le sommet d'une pile i correspond à calculer $i \ mod \ r$, où $i \ mod \ r$ désigne le reste de la division euclidienne de i par r : en partant avec le compteur supplémentaire à 0, on décrémente le compteur i de 1 et on incrémente le compteur supplémentaire de 1. Lorsque le compteur i atteint 0 on s'arrête. On décrémente alors le compteur supplémentaire de 1 en incrémentant le compteur i de 1 jusqu'à ce que le premier soit 0. On fait chacune de ces opérations en comptant en parallèle modulo r dans l'état interne de la machine. □

Théorème 4 *Toute machine à $k \geq 3$ compteurs se simule par une machine à 2 compteurs.*

Démonstration: Supposons d'abord $k = 3$. L'idée est coder trois compteurs i, j et k par l'entier $m = 2^i 3^j 5^k$. L'un des compteurs stocke cet entier. L'autre compteur est utilisé pour faire des multiplications, divisions, calculs modulo m, pour m valant 2, 3, ou 5.

Pour incrémenter i, j ou k de 1, il suffit de multiplier m par 2, 3 ou 5 en utilisant le principe de la preuve précédente.

Pour tester si i, j ou $k = 0$, il suffit de savoir si m est divisible par 2, 3 ou 5, en utilisant le principe de la preuve précédente.

Pour décrémenter i, j ou k de 1, il suffit de diviser m par 2, 3 ou 5 en utilisant le principe de la preuve précédente.

Pour $k > 3$, on utilise le même principe, mais avec les k premiers nombres premiers au lieu de simplement 2, 3, et 5. □

Exercice 2 *Reprendre l'exercice précédent en utilisant systématiquement au plus 2 compteurs.*

En combinant les résultats précédents, on obtient :

Corollaire 1 *Toute machine de Turing se simule par une machine à 2 compteurs.*

Observons que la simulation est particulièrement inefficace : la simulation d'un temps t de la machine de Turing nécessite un temps exponentiel pour la machine à 2 compteurs.

3 Thèse de Church-Turing

3.1 Équivalence de tous les modèles considérés

Nous avons jusque-là introduit différents modèles, et montré qu'ils pouvaient tous être simulés par des machines de Turing, ou simuler les machines de Turing.

En fait, tous ces modèles sont équivalents au niveau de ce qu'ils calculent : on a déjà montré que les machines RAM pouvaient simuler les machines de Turing. On peut prouver le contraire. On a montré que les machines à compteurs, et les machines à piles simulaient les machines de Turing. Il est facile de voir que le contraire est vrai : on peut simuler l'évolution d'une machine à piles ou d'une machine à compteurs par machine de Turing. Tous les modèles sont donc bien équivalents, au niveau de ce qu'ils sont capables de calculer.

3.2 Thèse de Church-Turing

C'est l'ensemble de ces considérations qui ont donné naissance à la thèse de Church-Turing, exprimée très explicitement pour la première fois par Stephen Kleene, étudiant de Alonzo Church. Cette thèse affirme que "ce qui est effectivement *calculable* est calculable par une machine de Turing."

Dans cette formulation, la première notion de "*calculable*" fait référence à une notion donnée intuitive, alors que la seconde notion de "calculable" signifie "calculable par une machine de Turing", i.e. une notion formelle.

Puisqu'il n'est pas possible de définir formellement la première notion, cette thèse est une thèse au sens philosophique du terme. Il n'est pas possible de la prouver.

La thèse de Church est très largement admise.

4 Notes bibliographiques

Lectures conseillées Pour aller plus loin sur les notions évoquées dans ce chapitre, nous suggérons la lecture de [Hopcroft et al., 2001] en anglais, ou de [Wolper, 2001], [Stern, 1994] [Carton, 2008] en français.

D'autres formalismes équivalents aux machines de Turing existent. En particulier, on peut considérer la notion de fonctions récursives, qui est présentée par exemple dans [Dowek, 2008], [Stern, 1994] ou dans [Cori and Lascar, 1993].

Bibliographie Ce chapitre a été rédigé à partir de la présentation des machines de Turing dans [Wolper, 2001], et des discussions de [Hopcroft et al., 2001] pour la partie sur leur programmation. La partie sur les machines RAM est inspirée des livres [Papadimitriou, 1994] et [Jones, 1997].

Index

Bibliographie

[Carton, 2008] Carton, O. (2008). Langages formels, calculabilité et complexité.

[Cori and Lascar, 1993] Cori, R. and Lascar, D. (1993). *Logique Mathématique, volume II*. Masson.

[Dowek, 2008] Dowek, G. (2008). *Les démonstrations et les algorithmes*. Polycopié du cours de l'Ecole Polytechnique.

[Hopcroft et al., 2001] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition.

[Jones, 1997] Jones, N. (1997). *Computability and complexity, from a programming perspective*. MIT press.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

[Stern, 1994] Stern, J. (1994). Fondements mathématiques de l'informatique. *Ediscience International, Paris*.

[Wolper, 2001] Wolper, P. (2001). *Introduction à la calculabilité : cours et exercices corrigés*. Dunod.