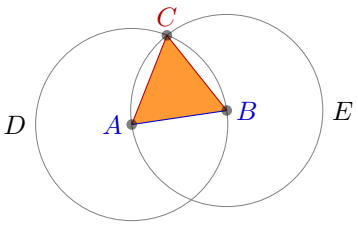
Chapitre 7

Modèles de calculs

Nous avons utilisé jusque-là à de multiples reprises la notion d'algorithme, sans en avoir donné une définition formelle. Intuitivement, on peut se dire qu'un algorithme est une méthode automatique, qui peut s'implémenter par un programme informatique, pour résoudre un problème donné.

Par exemple, la technique familière pour réaliser une addition, une multiplication, ou une division sur des nombres enseignée à l'école primaire correspond à un algorithme. Les techniques discutées pour évaluer la valeur de vérité d'une formule propositionnelle à partir de la valeur de ses propositions sont aussi des algorithmes. Plus généralement, nous avons décrit des méthodes de démonstration pour le calcul propositionnel ou le calcul des prédicats qui peuvent se voir comme des algorithmes.

Exemple 7.1 *L'exemple suivant est repris du manuel du paquetage* TikZ-PGF *version 2.0, lui-même inspiré des* Éléments *d'*Euclide.



Algorithme:
Pour construire un triangle équilatéral ayant pour coté AB: tracer le cercle de centre A de rayon AB; tracer le cercle de centre B de rayon AB. Nommer C l'une des intersections de ces deux cercles. Le triangle ABC est la solution recherchée.

Nous allons voir dans les chapitres suivants que tous les problèmes ne peuvent pas être résolus par algorithme, et ce même pour des problèmes très simples à formuler : par exemple,

– il n'y a pas d'algorithme pour déterminer si une formule close du calcul des prédicats est valide dans le cas général ;

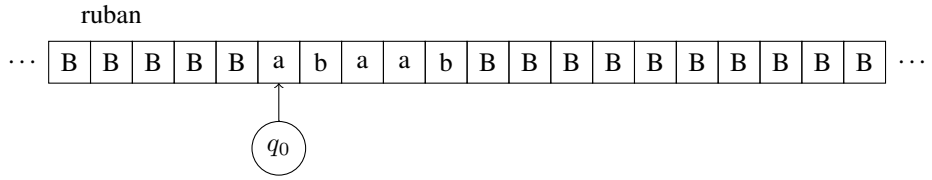


FIGURE 7.1 – Machine de Turing. La machine est sur l'état initial d'un calcul sur le mot $abaab$.

– il n'y a pas d'algorithme pour déterminer si un polynôme multivarié (à plusieurs variables) à coefficients entiers possède une racine entière (10*ème problème de Hilbert*).

Historiquement, c'est en fait la formalisation de ce que l'on appelle une démonstration, et des limites des systèmes de preuve formelle qui a mené aux modèles que l'on va discuter. On a ultérieurement compris que la notion capturée par ces modèles était beaucoup plus large que simplement la formalisation de la notion de démonstration, et couvrait en réalité une formalisation de tout ce qui est calculable par dispositif informatique digital. Cela reste d'actualité, puisque les ordinateurs actuels sont digitaux.

Par ailleurs, plusieurs formalisations ont été proposées de ces notions de façon indépendante, en utilisant des notions à priori très différentes : en particulier par Alonzo Church en 1936, à l'aide du formalisme du λ-*calcul*, par Turing en 1936, à l'aide de ce que l'on appelle les *machines de Turing*, ou par Post en 1936, à l'aide de systèmes de règles très simples, appelés *systèmes de Post*. Par la suite, on s'est convaincu que de nombreux formalismes étaient équivalents à tous ces modèles.

L'objet de ce chapitre est de définir quelques modèles de calculs et de montrer qu'ils sont équivalents à celui de la machine de Turing. Nous finirons en évoquant ce que l'on appelle la *thèse de Church-Turing*.

Les modèles que l'on va décrire peuvent être considérés comme très abstraits, mais aussi et surtout très limités et loin de couvrir tout ce que l'on peut programmer avec les langages de programmation évolués actuels comme CAML ou JAVA. Tout l'objet du chapitre est de se convaincre qu'il n'en est rien : tout ce qui programmable est programmable dans ces modèles.

7.1 Machines de Turing

7.1.1 Ingrédients

Une machine de Turing (déterministe) (voir la figure 7.1) est composée des éléments suivants :

1. Une mémoire infinie sous forme de ruban. Le ruban est divisé en cases. Chaque case peut contenir un élément d'un ensemble Σ (qui se veut un alphabet). On suppose que l'alphabet Σ est un ensemble fini.

2. une tête de lecture : la tête de lecture se déplace sur le ruban.
3. Un programme donné par une *fonction de transition* qui pour chaque état de la machine q, parmi un nombre fini d'états possibles Q, précise selon le symbole sous la tête de lecture :
   (a) l'état suivant $q' \in Q$;
   (b) le nouvel élément de Σ à écrire à la place de l'élément de Σ sous la tête de lecture ;
   (c) un sens de déplacement pour la tête de lecture.

L'exécution d'une machine de Turing sur un mot $w \in \Sigma^*$ peut alors se décrire comme suit : initialement, l'entrée se trouve sur le ruban, et la tête de lecture est positionnée sur la première lettre du mot. Les cases des rubans qui ne correspondent pas à l'entrée contiennent toutes l'élément B (symbole de blanc), qui est un élément particulier de Σ. La machine est positionnée dans son état initial q_0 : voir la figure 7.1.

A chaque étape de l'exécution, la machine, selon son état, lit le symbole se trouvant sous la tête de lecture, et selon ce symbole, elle
– remplace le symbole sous la tête de lecture par celui précisé par sa fonction transition ;
– déplace (ou non) cette tête de lecture d'une case vers la droite ou vers la gauche suivant le sens précisé par la fonction de transition ;
– change d'état vers l'état suivant.

Le mot w est dit accepté lorsque l'exécution de la machine finit par atteindre l'état d'acceptation.

7.1.2 Description

La notion de machine de Turing se formalise de la façon suivante :

Définition 7.1 (Machine de Turing) Une machine de Turing *est un 8-uplet*

$$M = (Q, \Sigma, \Gamma, B, \delta, q_0, q_a, q_r)$$

où :

1. *Q est l'ensemble fini des états ;*
2. *Σ est un alphabet* fini *;*
3. *Γ est l'alphabet de travail* fini *: $\Sigma \subset \Gamma$;*
4. *$B \in \Gamma$ est le caractère blanc ;*
5. *$q_0 \in Q$ est l'état initial ;*
6. *$q_a \in Q$ est l'état d'acceptation ;*
7. *$q_r \in Q$ est l'état de refus (ou d'arrêt) ;*
8. *δ est la fonction de transition : δ est une fonction (possiblement partielle) de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, |, \rightarrow\}$. Le symbole $\leftarrow$ est utilisé pour signifier un déplacement vers la gauche, $|$ aucun déplacement, $\rightarrow$ un déplacement vers la droite.*

Le langage accepté par une machine de Turing se définit à l'aide des notions de *configurations* et de la relation successeur entre configurations d'une machine de Turing. Une configuration correspond à toute l'information nécessaire pour décrire l'état de la machine à un instant donné, et pouvoir déterminer les états ultérieurs de la machine. A savoir :

- l'état ;
- le contenu du ruban ;
- la position de la tête de lecture.

Plus formellement,

Définition 7.2 (Configuration) Une configuration *est donnée par la description du ruban, par la position de la tête de lecture/écriture, et par l'état interne.*

Pour écrire une configuration, une difficulté est que le ruban est infini : le ruban correspond donc à une suite infinie de symboles de l'alphabet Γ de travail de la machine. Toutefois, à tout moment d'une exécution seule une partie finie du ruban a pu être utilisée par la machine. En effet, initialement la machine contient un mot en entrée de longueur finie et fixée, et à chaque étape la machine déplace la tête de lecture au plus d'une seule case. Par conséquent, après t étapes, la tête de lecture a au plus parcouru t cases vers la droite ou vers la gauche à partir de sa position initiale. Par conséquent, le contenu du ruban peut à tout moment se définir par le contenu d'un préfixe fini, le reste ne contenant que le symbole blanc B. Pour noter la position de la tête de lecture, nous pourrions utiliser un entier $n \in \mathbb{Z}$.

Nous allons en fait utiliser plutôt l'astuce suivante qui a le seul mérite de simplifier nos définitions et nos preuves ultérieures : au lieu de voir le ruban comme un préfixe fini, nous allons le représenter par deux préfixes finis : le contenu de ce qui est à droite de la tête de lecture, et le contenu de ce qui est à gauche de la tête de lecture. On écrira le préfixe correspondant au contenu à droite comme habituellement de gauche à droite. Par contre, on écrira le préfixe correspondant au contenu à gauche de la tête de lecture de droite à gauche : l'intérêt est que la première lettre du préfixe gauche est la case immédiatement à gauche de la tête de lecture. Une *configuration* sera donc un élément de $Q \times \Gamma^* \times \Gamma^*$.

Formellement :

Définition 7.3 (Notation d'une configuration) *Une configuration se note $C = (q, u, v)$, avec $u, v \in \Gamma^*$, $q \in Q$: u et v désignent le contenu respectivement à gauche et à droite de la tête de lecture du ruban i, la tête de lecture du ruban i étant sur la première lettre de v. On suppose que les dernières lettres de u et de v ne sont pas le symbole de blanc B.*

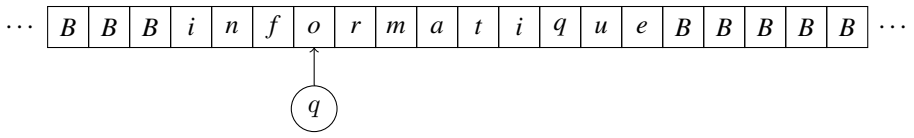
On fixe la convention que le mot v est écrit de gauche à droite (la lettre numéro $i+1$ de v correspond au contenu de la case à droite de celle de numéro i) alors que le mot u est écrit de droite à gauche (la lettre numéro $i+1$ de u correspond au contenu de la case à gauche de celle de numéro i, la première lettre de u étant à gauche de la tête de lecture).

Exemple 7.2 *La configuration de la machine représentée sur la figure 7.1 est* $(q_0, abaab, \epsilon)$.

On notera parfois autrement les configurations :

Définition 7.4 (Notation alternative) *La configuration (q, u, v) sera aussi vue/notée dans certaines sections ou chapitres comme/par $u\boldsymbol{q}v$, en gardant u et v écrit de gauche à droite.*

Exemple 7.3 *Une configuration comme*



se code par la configuration $(q, fni, ormatique)$, ou parfois par $inf\boldsymbol{q}ormatique$.

Une configuration est dite *acceptante* si $q = q_a$, *refusante* si $q = q_r$.

Pour $w \in \Sigma^*$, la configuration initiale correspondante à w est la configuration $C[w] = (q_0, \epsilon, w)$.

On note : $C \vdash C'$ si la configuration C' est le successeur direct de la configuration C par le programme (donné par δ) de la machine de Turing.

Formellement, si $C = (q, u, v)$ et si a désigne la première lettre [1] de v, et si $\delta(q, a) = (q', a', m')$ alors $C \vdash C'$ si

- $C' = (q', u', v')$, et
  - si $m' = |$, alors $u' = u$, et v' est obtenu en remplaçant la première lettre a de v par a' ;
  - si $m' = \leftarrow$, $v' = a'v$, et u' est obtenu en supprimant la première lettre de u ;
  - si $m' = \rightarrow$, $u' = a'u$, et v' est obtenu en supprimant la première lettre a de v.

Remarque 7.1 *Ces règles traduisent simplement la notion de réécriture de la lettre a par la lettre a' et le déplacement correspondant à droite ou à gauche de la tête de lecture.*

Définition 7.5 (Mot accepté) *Un mot $w \in \Sigma^*$ est dit* accepté *(en temps t) par la machine de Turing, s'il existe une suite de configurations $C_1, \cdots, C_t$ avec :*

1. *$C_0 = C[w]$;*
2. *$C_i \vdash C_{i+1}$ pour tout $i < t$;*
3. *aucune configuration C_i pour $i < t$ n'est acceptante ou refusante.*
4. *C_t est acceptante.*

Définition 7.6 (Mot refusé) *Un mot $w \in \Sigma^*$ est dit* refusé *(en temps t) par la machine de Turing, s'il existe une suite de configurations $C_1, \cdots, C_t$ avec :*

1. *$C_0 = C[w]$;*

1. Avec la convention que la première lettre du mot vide est le blanc B.

2. $C_i \vdash C_{i+1}$ *pour tout* $i < t$ *;*
3. *aucune configuration* C_i *pour* $i < t$ *n'est acceptante ou refusante.*
4. C_t *est refusante.*

Définition 7.7 (Machine qui boucle sur un mot) *On dit que la machine de Turing* boucle *sur un mot* w*, si* w *n'est ni accepté, et ni refusé.*

Remarque 7.2 *Chaque mot* w *est donc dans l'un des trois cas exclusifs suivants :*

1. *il est accepté par la machine de Turing ;*
2. *il est refusé par la machine de Turing ;*
3. *la machine de Turing boucle sur ce mot.*

Remarque 7.3 *La terminologie* boucle *signifie simplement que la machine ne s'arrête pas sur ce mot : cela ne veut pas dire nécessairement que l'on répète à l'infini les mêmes instructions. La machine peut boucler pour plusieurs raisons. Par exemple, parce qu'elle atteint une configuration qui n'a pas de successeur défini, ou parce qu'elle rentre dans un comportement complexe qui produit une suite infinie de configurations ni acceptante ni refusante.*

Plus généralement, on appelle *calcul de* Σ *sur un mot* $w \in \Sigma^*$, une suite (finie ou infinie) de configurations $(C_i)_{i\in\mathbb{N}}$ telle que $C_0 = C[w]$ et pour tout i, $C_i \vdash C_{i+1}$, avec la convention qu'un état acceptant ou refusant n'a pas de successeur.

Définition 7.8 (Langage accepté par une machine) *Le langage* $L \subset \Sigma^*$ accepté par M *est l'ensemble des mots* w *qui sont acceptés par la machine. On le note* $L(M)$*. On l'appelle* $L(M)$ *aussi* le langage reconnu *par* M*.*

On n'aime pas en général les machines qui ne s'arrêtent pas. On cherche donc en général à garantir une propriété plus forte :

Définition 7.9 (Langage décidé par une machine) *On dit qu'un langage* $L \subset \Sigma^*$ *est* décidé par Σ *par la machine si :*
- *pour* $w \in L$*,* w *est accepté par la machine ;*
- *pour* $w \notin L$ *(=sinon),* w *est refusé par la machine.*

Autrement dit, la machine accepte L et termine sur toute entrée.

On dit dans ce cas que la machine *décide* L.

7.1.3 Programmer avec des machines de Turing

La programmation avec des machines de Turing est extrêmement bas niveau. Nous allons voir que l'on peut toutefois programmer réellement beaucoup de choses avec ce modèle. La première étape est de se convaincre que plein de problèmes simples peuvent se programmer. A vrai dire, la seule façon de s'en convaincre est d'essayer soit même de programmer avec des machines de Turing, c'est-à-dire de faire les exercices qui suivent.

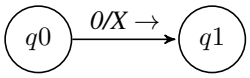
Exercice 7.1. *Construire une machine de Turing qui accepte exactement les mots w sur l'alphabet $\Sigma = \{0, 1\}$ de la forme $0^n 1^n$, $n \in \mathbb{N}$.*

Voici une solution. On considère une machine avec $Q = \{q0, q1, q2, q3, q4\}$, $\Gamma = \{0, 1, X, Y, B\}$, l'état d'acceptation $q4$ et une fonction de transition δ telle que :

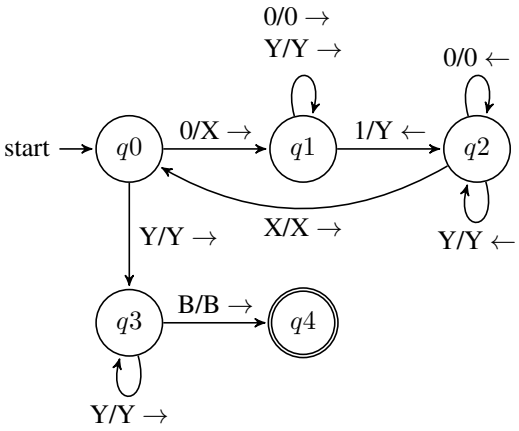
- $\delta(q0, 0) = (q1, X, \rightarrow)$;
- $\delta(q0, Y) = (q3, Y, \rightarrow)$;
- $\delta(q1, 0) = (q1, 0, \rightarrow)$;
- $\delta(q1, 1) = (q2, Y, \leftarrow)$;
- $\delta(q1, Y) = (q1, Y, \rightarrow)$;
- $\delta(q2, 0) = (q2, 0, \leftarrow)$;
- $\delta(q2, X) = (q0, X, \rightarrow)$;
- $\delta(q2, Y) = (q2, Y, \leftarrow)$;
- $\delta(q3, Y) = (q3, Y, \rightarrow)$;
- $\delta(q3, B) = (q4, B, \rightarrow)$.

On le voit, décrire de cette façon une machine de Turing est particulièrement peu lisible. On préfère représenter le programme d'une machine (la fonction δ) sous la forme d'un graphe : les sommets du graphe représentent les états de la machine. On représente chaque transition $\delta(q, a) = (q', a', m)$ par un arc de l'état q vers l'état q' étiqueté par a/a' m. L'état initial est marqué par une flèche entrante. L'état d'acceptation est marqué par un double cercle.

Exemple 7.4 *Par exemple, la transition $\delta(q0, 0) = (q1, X, \rightarrow)$ se représente graphiquement par :*



Selon ce principe, le programme précédent se représente donc par :



Comment fonctionne ce programme : lors d'un calcul, la partie du ruban que la machine aura visité sera de la forme $X^*0^*Y^*1^*$. A chaque fois que l'on lit un 0, on

le remplace par un X, et on rentre dans l'état $q1$ ce qui correspond à lancer la sous-procédure suivante : on se déplace à droite tant que l'on lit un 0 ou un Y. Dès qu'on a atteint un 1, on le transforme en un Y, et on revient à gauche jusqu'à revenir sur un X (le X qu'on avait écrit) et s'être déplacé d'une case vers la droite.

En faisant ainsi, pour chaque 0 effacé (i.e. X marqué), on aura effacé un 1 (i.e. marqué un Y). Si on a marqué tous les 0 et que l'on atteint un Y, on rentre dans l'état $q3$, ce qui a pour objet de vérifier que ce qui est à droite est bien constitué uniquement de Y. Lorsqu'on a tout lu, i.e. atteint un B, on accepte, i.e. on va dans l'état $q4$.

Bien entendu, une vraie preuve de la correction de cet algorithme consisterait à montrer que si un mot est accepté, c'est que nécessairement il est du type 0^n1^n. Nous laissons le lecteur s'en convaincre.

Exemple 7.5 *Voici un exemple de calcul acceptant pour M : $q_00011 \vdash Xq_1011 \vdash X0q_111 \vdash Xq_20Y1 \vdash q_2X0Y1 \vdash Xq_00Y1 \vdash XXq_1Y1 \vdash XXYq_11 \vdash XXq_2YY \vdash Xq_2XYY \vdash XXq_0YY \vdash XXYq_3Y \vdash XXYYq_3B \vdash XXYYBq_4B$.*

Définition 7.10 (Diagramme espace-temps) *On représente souvent une suite de configurations ligne par ligne : la ligne numéro i représente la $i$ème configuration du calcul, avec le codage de la définition 7.4. Cette représentation est appelée un* diagramme espace-temps *de la machine.*

Exemple 7.6 *Voici le diagramme espace-temps correspondant au calcul précédent sur* 0011.

$\cdots$	B	B	B	B	q_0	0	0	1	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_1	0	1	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	0	q_1	1	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_2	0	Y	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	q_2	X	0	Y	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_0	0	Y	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	q_1	Y	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	q_1	1	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	q_2	Y	Y	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_2	X	Y	Y	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	q_0	Y	Y	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	q_3	Y	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	Y	q_3	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	Y	B	q_4	B	B	B	B	B	B	B	B	B	$\cdots$

Exemple 7.7 *Voici le diagramme espace-temps du calcul de la machine sur* 0010 *:*

$\cdots$	B	B	B	B	q_0	0	0	1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_1	0	1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	0	q_1	1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_2	0	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	q_2	X	0	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_0	0	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	q_1	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	q_1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	0	q_1	B	B	B	B	B	B	B	B	B	B	$\cdots$

Observons que sur la dernière configuration plus aucune évolution n'est possible, et donc il n'y a pas de calcul accepté partant de 0010.

Exercice 7.2. *[Soustraction en unaire] Construire un programme de machine de Turing qui réalise une soustraction en unaire : partant d'un mot de la forme* 0^m10^n*, la machine s'arrête avec* $0^{m\ominus n}$ *sur son ruban (entouré de blancs), où* $m \ominus n$ *est* $\max(0, m-n)$.

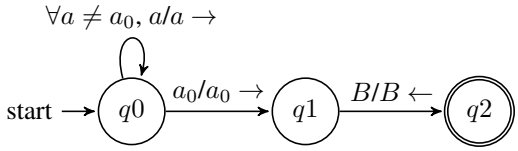
7.1.4 Techniques de programmation

Voici quelques techniques utilisées couramment dans la programmation des machines de Turing.

La première consiste à coder une information finie dans l'état de la machine. On va l'illustrer sur un exemple, où l'on va stocker le premier caractère lu dans l'état. Tant que l'information à stocker est finie, cela reste possible.

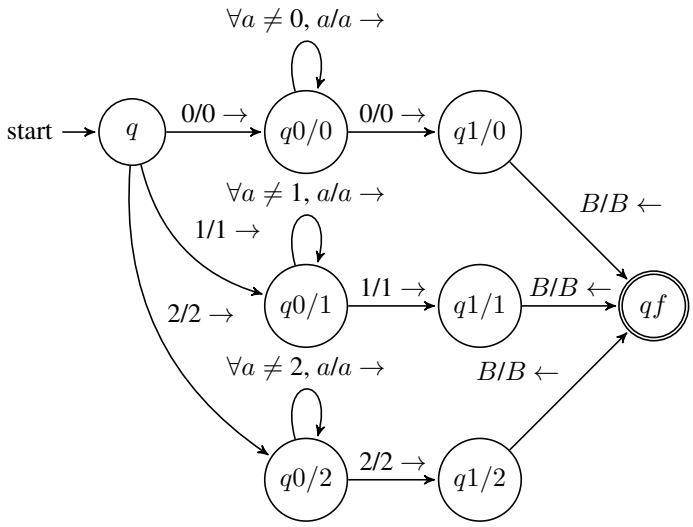
Exercice 7.3. *Construire un programme de machine de Turing qui lit le symbole en face de la tête de lecture et vérifie que ce dernier n'apparaît nul part ailleurs à droite, sauf sur la toute dernière lettre à droite.*

Si l'on fixe un symbole $a_0 \in \Sigma$ de l'alphabet Σ, il est facile de construire un programme qui vérifie que le symbole a_0 n'apparaît nul part sauf sur la toute dernière lettre à droite.



où $\forall a \neq a_0$ désigne le fait que l'on doit répéter la transition $a/a, \rightarrow$ pour tout symbole $a \neq a_0$.

Maintenant pour résoudre notre problème, il suffit de lire la première lettre a_0 et de recopier ce programme autant de fois qu'il y a de lettres dans Σ. Si $\Sigma = \{0, 1, 2\}$ par exemple :



On utilise donc le fait dans cet automate que l'on travaille sur des états qui peuvent être des couples : ici on utilise des couples q_i/j avec $i \in \{1, 2\}$, et $j \in \Sigma$.

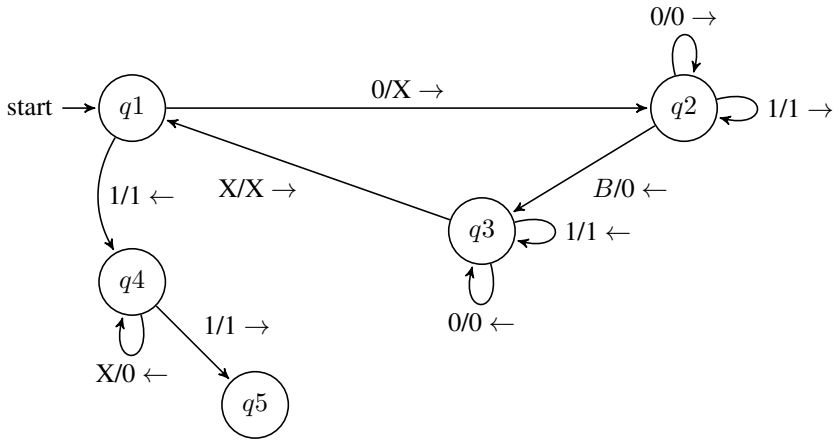
Une second technique consiste en l'utilisation de sous-procédures. Là encore, on va l'illustrer sur un exemple.

Exercice 7.4. *[Multiplication en unaire] Construire un programme de machine de Turing qui réalise une multiplication en unaire : partant d'un mot de la forme 0^m10^n, la machine s'arrête avec 0^{m*n} sur son ruban.*

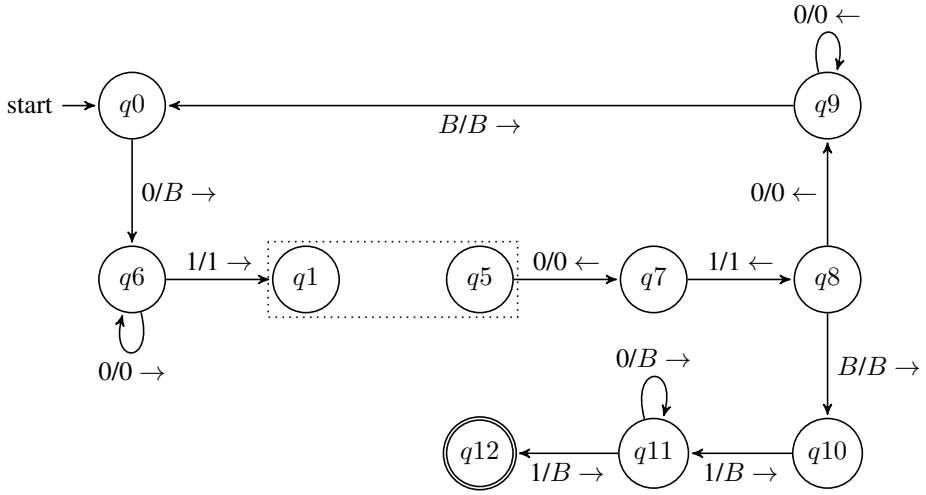
Une stratégie possible est la suivante :

1. le ruban contiendra un mot de la forme $0^i10^n10^{kn}$ pour un certain entier k ;
2. dans chaque étape, on change un 0 du premier groupe en un blanc, et on ajoute n 0 au dernier groupe, pour obtenir une chaîne de la forme $0^{i-1}10^n10^{(k+1)n}$;
3. en faisant ainsi, on copie le groupe de n 0 m fois, une fois pour chaque symbole du premier groupe mis à blanc. Quand il ne reste plus de blanc dans le premier groupe de 0, il y aura donc $m * n$ groupes dans le dernier groupe ;
4. la dernière étape est de changer le préfixe 10^n1 en des blancs, et cela sera terminé.

Le cœur de la méthode est donc la sous-procédure, que l'on appellera $Copy$ qui implémente l'étape 2 : elle transforme une configuration $0^{m-k}1q_10^n1^{(k-1)n}$ en $0^{m-k}1q_50^n1^{kn}$. Voici une façon de la programmer : si l'on part dans l'état q_1 avec une telle entrée, on se retrouve dans l'état q_5 avec le résultat correct.



Une fois que l'on a cette sous-procédure, on peut concevoir l'algorithme global.



où le rectangle en pointillé signifie "coller ici le programme décrit avant pour la sous-procédure".

On le voit sur cet exemple, il est possible de programmer les machines de Turing de façon modulaire, en utilisant des notions de sous-procédure, qui correspondent en fait à des collages de morceaux de programme au sein du programme d'une machine, comme sur cet exemple.

7.1.5 Applications

Répétons-le : la seule façon de comprendre tout ce que l'on peut programmer avec une machine de Turing consiste à essayer de les programmer.

Voici quelques exercices.

Exercice 7.5. *Construire une machine de Turing qui ajoute* 1 *au nombre écrit en binaire (donc avec des* $\mathbf{0}$ *et* $\mathbf{1}$*) sur son ruban.*

Exercice 7.6. *Construire une machine de Turing qui soustrait* 1 *au nombre écrit en binaire (donc avec des* $\mathbf{0}$ *et* $\mathbf{1}$*) sur son ruban.*

Exercice 7.7. *Construire une machine de Turing qui accepte les chaînes de caractère avec le même nombre de* $\mathbf{0}$ *et de* $\mathbf{1}$.

7.1.6 Variantes de la notion de machine de Turing

Le modèle de la machine de Turing est extrêmement robuste.

En effet, existe de nombreuses variantes possibles autour du concept de machine de Turing, qui ne changent rien en fait à ce que l'on arrive à programmer avec ces machines.

On peut en effet assez facilement se persuader des propositions suivantes.

Restriction à un alphabet binaire

Proposition 7.1 *Toute machine de Turing qui travaille sur un alphabet* Σ *quelconque peut être simulée par une machine de Turing qui travaille sur un alphabet* $\Sigma = \Gamma$ *avec uniquement deux lettres (sans compter le caractère blanc).*

Démonstration (principe) : L'idée est que l'on peut toujours coder les lettres de l'alphabet en utilisant un codage en binaire. Par exemple, si l'alphabet Σ possède 3 lettres a, b, et c, on peut décider de coder a par 00, b par 01 et c par 10 : voir la figure 7.2. Dans le cas plus général, il faut simplement utiliser éventuellement plus que 2 lettres.

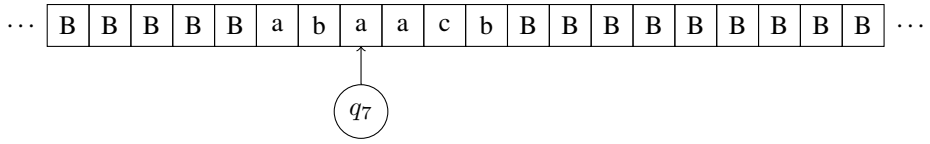
On peut alors transformer le programme d'une machine de Turing M qui travaille sur l'alphabet Σ en un programme M' qui travaille sur ce codage.

Par exemple, si le programme de M contient une instruction qui dit que si M est dans l'état q, et que la tête de lecture lit un a il faut écrire un c et se déplacer à droite, le programme de M' consistera à dire que si l'on est dans l'état q et que l'on lit 0 en face de la tête de lecture, et 0 à sa droite (donc ce qui est à droite de la tête de lecture commence par 00, i.e. le codage de a), alors il faut remplacer ces deux 0 par 10 (i.e. le codage de c) et se rendre dans l'état q'. En faisant ainsi, à chaque fois qu'un calcul de M produit un ruban correspondant à un mot w, alors M' produira un un ruban correspondant au codage de w en binaire lettre par lettre. □

Machines de Turing à plusieurs rubans

On peut aussi considérer des machines de Turing qui auraient plusieurs rubans, disons k rubans, où k est un entier. Chacun des k rubans possède sa propre tête de lecture. La machine possède toujours un nombre fini d'états Q. Simplement, maintenant la fonction de transition δ n'est plus une fonction de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, |, \rightarrow\}$. mais de $Q \times \Gamma^k$ dans $Q \times \Gamma^k \times \{\leftarrow, |, \rightarrow\}^k$: en fonction de l'état de la machine et de ce

Machine M sur l'alphabet $\{a, b, c\}$



Machine M' simulant M sur l'alphabet $\{0, 1\}$.

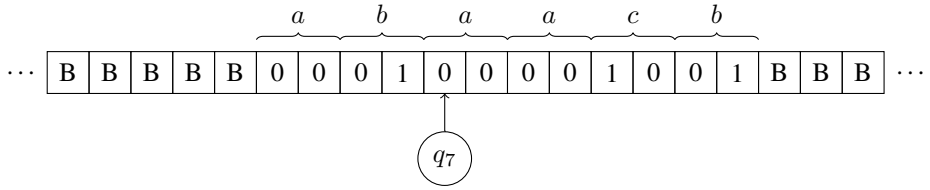


FIGURE 7.2 – Illustration de la preuve de la proposition 7.1.

qui est lu en face des têtes de lecture de chaque ruban, la fonction de transition donne les nouveaux symboles à écrire sur chacun des rubans, et les déplacements à effectuer sur chacun des rubans.

Il est possible de formaliser ce modèle, ce que nous ne ferons pas car cela n'apporte pas de réelle nouvelle difficulté.

On pourra se persuader du résultat suivant :

Proposition 7.2 *Toute machine de Turing qui travaille avec k-rubans peut être simulée par une machine de Turing avec un unique ruban.*

Démonstration (principe) : L'idée est que si une machine M travaille avec k rubans sur l'alphabet Γ, on peut simuler M par une machine M' avec un unique ruban qui travaille sur l'alphabet $(\Gamma \times \{0, 1\} \cup \{\#\})$ (qui est toujours un alphabet fini).

Le ruban de M' contient la concaténation des contenus des rubans de M, séparés par un marqueur $\#$. On utilise $(\Gamma \times \{0, 1\} \cup \{\#\})$ au lieu de $(\Gamma \cup \{\#\})$ de façon à utiliser 1 bit d'information de plus par case qui stocke l'information "la tête de lecture est en face de cette case".

M' va simuler étape par étape les transitions de M : pour simuler une transition de M, M' va parcourir de gauche à droite son ruban pour déterminer la position de chacune des têtes de lecture, et le symbole en face de chacune des têtes de lecture (en mémorisant ces symboles dans son état interne). Une fois connu tous les symboles en face de chacune des têtes de lecture, M' connaît les symboles à écrire et les déplacements à effectuer pour chacune des têtes : M' va parcourir son ruban à nouveau de

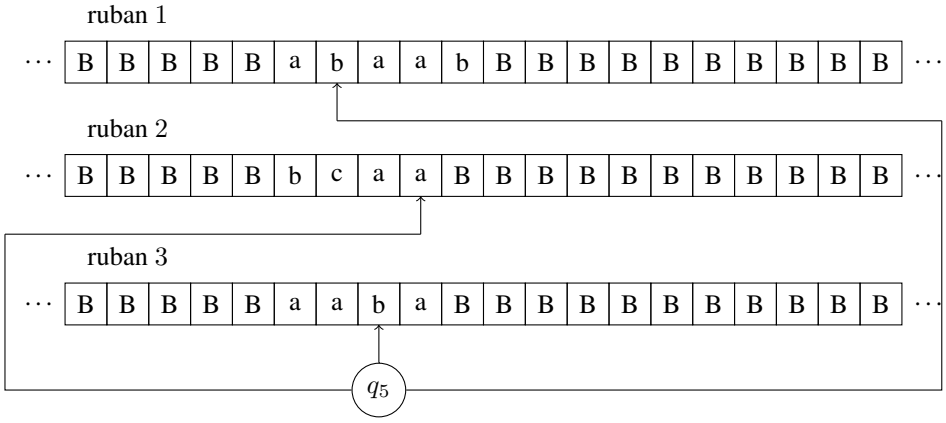


FIGURE 7.3 – Une machine de Turing à 3 rubans

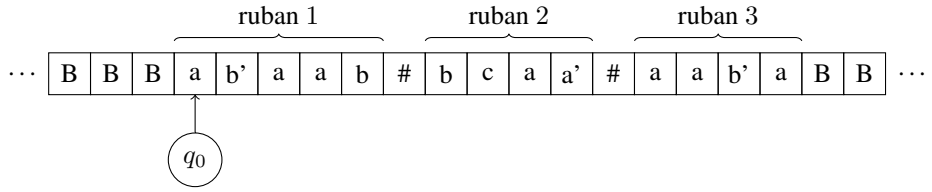


FIGURE 7.4 – Illustration de la preuve de la proposition 7.2 : représentation graphique d'une machine de Turing à 1 ruban simulant la machine à 3 rubans de la figure 7.3. Sur cette représentation graphique, on écrit une lettre primée lorsque le bit "la tête de lecture est en face de cette case" est à 1.

gauche à droite pour mettre à jour son codage de l'état de M. En faisant ainsi systématiquement transition par transition, M' va parfaitement simuler l'évolution de M avec son unique ruban : voir la figure 7.1.6 □

Machines de Turing non-déterministes

On peut aussi introduire le concept de machine de Turing non-déterministe : la définition d'une machine de Turing non-déterministe est exactement comme celle de la notion de machine de Turing (déterministe) sauf sur un point. δ n'est plus une fonction de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, |, \rightarrow\}$, mais une relation de la forme

$$\delta \subset (Q \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, |, \rightarrow\}).$$

En d'autres termes, pour un état et une lettre lue en face de la tête de lecture donnée, δ ne définit pas un seul triplet de $Q \times \Gamma \times \{\leftarrow, |, \rightarrow\}$, mais un ensemble de triplets. Intuitivement, lors d'une exécution la machine a la possibilité de choisir n'importe quel triplet.

Formellement, cela s'exprime par le fait que l'on peut passer de la configuration C à la configuration successeur C' si et seulement si on peut passer de C à C' (ce que nous notons $C \vdash C'$) avec les définitions précédentes, mais en remplaçant $\delta(q, a) = (q', a', m')$ par $((q, a), (q', a', m')) \in \delta$. Les autres définitions sont alors essentiellement inchangées, et comme pour les machines de Turing déterministes.

La différence est qu'une machine de Turing non-déterministe n'a pas une exécution unique sur une entrée w, mais éventuellement plusieurs : en fait, les exécutions de la machine sur un mot w donnent lieu à un arbre de possibilité, et l'idée est qu'on accepte (respectivement : refuse) un mot si l'une des branches contient une configuration acceptante (resp. de refus).

La notion de mot w accepté est (toujours) donnée par définition 7.5.

Le langage $L \subset \Sigma^*$ *accepté par* M est (toujours) l'ensemble des mots w qui sont acceptés par la machine. On le note (toujours) $L(M)$. On appelle (toujours) $L(M)$ aussi *le langage reconnu* par M.

On évite dans ce contexte de parler en général de mot *refusé*.

On dira cependant qu'un langage $L \subset \Sigma^*$ est *décidé par* M si il est accepté par une machine qui termine sur toute entrée : c'est-à-dire telle que pour $w \in L$, la machine possède **un** calcul qui mène à une configuration acceptante comme dans la définition 7.5, et pour $w \notin L$, **tous** les calculs de la machine mènent à une configuration refusante.

On peut prouver le résultat suivant (ce que nous ferons dans un chapitre ultérieur).

Proposition 7.3 *Une machine de Turing non-déterministe peut être simulée par une machine de Turing déterministe : un langage L est accepté par une machine de Turing non-déterministe si et seulement si il est accepté par une machine de Turing (déterministe).*

Évidemment, on peut considérer une machine de Turing comme une machine de Turing non-déterministe particulière. Le sens moins trivial de la proposition est que l'on peut simuler une machine de Turing non-déterministe par une machine de Turing (déterministe).

Autrement dit, autoriser du non-déterministe n'étend pas le modèle, tant que l'on parle de *calculabilité*, c'est-à-dire de ce que l'on peut résoudre. Nous verrons qu'en ce qui concerne la *complexité*, cela est une autre paire de manches.

7.1.7 Localité de la notion de calcul

Voici une propriété fondamentale de la notion de calcul, que nous utiliserons à de plusieurs reprises, et que nous invitons notre lecteur à méditer :

Proposition 7.4 (Localité de la notion de calcul) *Considérons le diagramme espace-temps d'une machine M. Regardons les contenus possibles des sous-rectangles de largeur 3 et de hauteur 2 dans ce diagramme. Pour chaque machine M, il y a un nombre fini possible de contenus que l'on peut trouver dans ces rectangles. Appelons* fenêtres légales*, les contenus possibles pour la machine M : voir la figure 7.6.*

Par ailleurs, cela fournit même une caractérisation des diagrammes espace-temps d'une machine donnée : un tableau est un diagramme espace-temps de M sur une certaine configuration initiale C_0 si et seulement si d'une part sa première ligne correspond à C_0, et d'autre part dans ce tableau, le contenu de tous les rectangles de largeur 3 et de hauteur 2 possible sont parmi les fenêtres légales.

Démonstration : Il suffit de regarder chacun des cas possibles et de s'en convaincre, ce qui est fastidieux, mais sans aucune difficulté particulière. □

Nous y reviendrons. Oublions-la pour l'instant, et revenons à d'autres modèles.

Remarque 7.4 *C'est aussi vrai dans les autres modèles dans un certain sens. Toutefois, cela y est toutefois beaucoup plus difficile à formuler.*

7.2 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*.

7.2.1 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). 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 :

(a)

$\cdots$	B	B	B	B	q_0	0	0	1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_1	0	1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	0	q_1	1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_2	0	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	q_2	X	0	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	q_0	0	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	q_1	Y	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	q_1	0	B	B	B	B	B	B	B	B	B	B	$\cdots$
$\cdots$	B	B	B	B	X	X	Y	0	q_1	B	B	B	B	B	B	B	B	B	B	$\cdots$

(b)

1	0	B
Y	0	B

FIGURE 7.5 – (a). Le diagramme espace-temps de l'exemple 7.7, sur lequel est grisé un sous-rectangle 3×2. (b) La fenêtre (légale) correspondante.

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

FIGURE 7.6 – Quelques fenêtres légales pour une autre machine de Turing M : on peut rencontrer chacun de ces contenus dans un sous-rectangle 3×2 du diagramme espace-temps de M.

(a)
a	b	a
a	a	a

(b)
a	q_1	b
q_1	a	a

(c)
b	q_1	b
q_2	b	q_2

FIGURE 7.7 – Quelques fenêtres illégales pour une certaine machine M avec $\delta(q_1, b) = (q_1, c, \leftarrow)$. On ne peut pas rencontrer ces contenus dans un sous-rectangle 3×2 du diagramme espace-temps de M : En effet, dans (a), le symbole central ne peut pas changer sans que la tête lui soit adjacente. Dans (b), le symbole en bas à droite devrait être un c mais pas un a, selon la fonction de transition. Dans (c), il ne peut pas y avoir deux têtes de lecture sur le ruban.

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).

7.2.2 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 7.11 *Une machine RISC est une machine SRAM 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.

Théorème 7.1 *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

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

Le second ruban code un mot de la forme

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

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 $\mathbf{1}$ par des $\mathbf{0}$, en se déplaçant vers la gauche tant que possible. Lorsqu'un $\mathbf{0}$ ou un B est trouvé, on le change en $\mathbf{1}$ 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 $B\mathbf{10111}B$ ($\mathbf{10111}$ 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 $\mathbf{10111}$ 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 $B\mathbf{10111}B$ 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 $B\mathbf{10111}B$ ($\mathbf{10111}$ 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 $B\mathbf{10111}B$ 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 $\mathbf{0}$ 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.

□

7.2.3 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.

7.3 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.

7.3.1 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 $a \in M$ 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 7.2 *Toute machine de Turing peut être simulée par une machine à 2 piles.*

Démonstration : Selon la formalisation de la page 4, 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

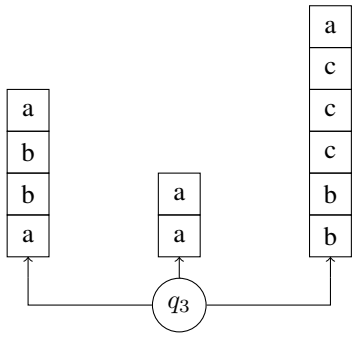


FIGURE 7.8 – Une machine à 3 piles.

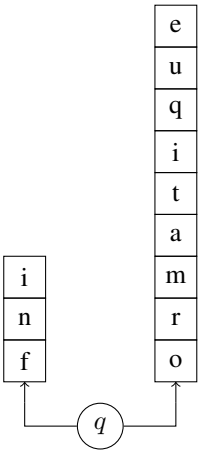


FIGURE 7.9 – La machine de Turing de l'exemple 7.3 vue comme une machine à 2-piles.

piles : voir la figure 7.9. Si l'on relit attentivement la formalisation page 4, 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. □

7.3.2 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 7.5 *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.

- $\mathrm{Inc}(c, j)$: on incrémente le compteur c puis on va à l'instruction j.
- $\mathrm{Decr}(c, j)$: on décrémente le compteur c puis on va à l'instruction j.
- $\mathrm{IsZero}(c, j, k)$: 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. $\mathrm{IsZero}(1, 5, 2)$
2. $\mathrm{Decr}(1, 3)$
3. $\mathrm{Inc}(3, 4)$
4. $\mathrm{Inc}(3, 1)$
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 7.8. *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 7.3 *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 7.4 *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 7.9. *Reprendre l'exercice précédent en utilisant systématiquement au plus* 2 *compteurs.*

En combinant les résultats précédents, on obtient :

Corollaire 7.1 *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.

7.4 Thèse de Church-Turing

7.4.1 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.

7.4.2 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.

7.5 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, 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 dans [Hopcroft et al., 2001] pour la partie

sur leur programmation. La partie sur les machines RAM est inspirée de [Papadimitriou, 1994] et de [Jones, 1997].

Index

(A_g, r, A_d), *voir* arbre binaire
(V, E), *voir* graphe
(q, u, v), 4, *voir* configuration d'une machine de Turing
., *voir* concaténation
A^c, *voir* complémentaire
$C[w]$, *voir* configuration d'une machine de Turing, initiale, 5
$F(G/p)$, *voir* substitution
$L(M)$, 6, 15, *voir* langage accepté par une machine de Turing
$\Leftrightarrow$, *voir* double implication
$\Rightarrow$, *voir* définition inductive / différentes notations d'une, *voir* implication
Σ, *voir* alphabet
Σ^*, *voir* ensemble des mots sur un alphabet
$\cap$, *voir* intersection de deux ensembles
$\cup$, *voir* union de deux ensembles
ϵ, *voir* mot vide
$\equiv$, *voir* équivalence entre formules, *voir* équivalence entre problèmes
$\exists$, *voir* quantificateur
$\forall$, *voir* quantificateur
λ-calcul, 2
$\leq_m$, *voir* réduction
$|w|$, *voir* longueur d'un mot
$\leq$, *voir* réduction
$\mathcal{P}(E)$, *voir* parties d'un ensemble
$\models$, *voir* conséquence sémantique
$\neg$, *voir* négation
$\not\models$, *voir* conséquence sémantique
$\ominus$, 9
$\subset$, *voir* partie d'un ensemble
$\times$, *voir* produit cartésien de deux ensembles
$\vdash$, 5, *voir* démonstration, *voir* relation successeur entre configurations d'une machines de Turing
$\vee$, *voir* disjonction
$\wedge$, *voir* conjonction
$u\mathbf{q}v$, 4, *voir* configuration d'une machine de Turing
$\langle\langle M\rangle, w\rangle$, *voir* codage d'une paire
$\langle M\rangle$, *voir* codage d'une machine de Turing
$\langle m\rangle$, *voir* codage
$\langle\phi\rangle$, *voir* codage d'une formule
$\langle w_1, w_2\rangle$, *voir* codage d'une paire

algorithme, 1
$Arith$, *voir* expressions arithmétiques
$Arith'$, *voir* expressions arithmétiques parenthésées

boucle, 5, 6

calcul
  λ-calcul, 2
  d'une machine de Turing, 6
calculabilité, 15
Church-Turing, *voir* thèse
codage
  notation, *voir* $\langle . \rangle$
  d'une formule
    notation, *voir* $\langle\phi\rangle$
  d'une machine de Turing
    notation, *voir* $\langle M\rangle$
  d'une paire
    notation, *voir* $\langle\langle M\rangle, w\rangle$, *voir* $\langle w_1, w_2\rangle$
complémentaire
  notation, *voir* A^c
  du problème de l'arrêt des machines de Turing

notation , *voir* $\overline{\text{HALTING} - \text{PROBLEM}}$
complétude
RE-complétude, *voir* RE-complet
complexité, 15
concaténation
notation, *voir* .
configuration d'une machine de Turing, 4
notation, *voir* (q, u, v), *voir* $u\mathbf{q}v$
acceptante, 5
initiale, 5
initiale,*notation*, *voir* $C[w]$
refusante, 5

décidé, *voir* langage
démonstration, 2
diagramme espace-temps, 8
décide, 6

équivalence
entre problèmes
notation, *voir* $\equiv$

fenêtres légales, 15
fonction
de transition d'une machine de Turing, 3

graphe
notation, *voir* (V, E)

HALTING − PROBLEM, *voir* problème de l'arrêt des machines de Turing
$\overline{\text{HALTING} - \text{PROBLEM}}$, *voir* complémentaire du problème de l'arrêt d'une machine de Turing

intersection de deux ensembles
notation, *voir* $\cap$

langage
accepté par une machine de Turing, 6
notation, *voir* $L(M)$
non déterministe, 15
décidé par une machine de Turing, 6
non déterministe, 15
reconnu par une machine de Turing
synonyme : langage accepté par une machine de Turing, *voir* langage accepté par une machine de Turing
le langage reconnu, 6
localité de la notion de calcul, 15
longueur d'un mot
notation, *voir* $|w|$

machines
de Turing, 2, 3
à plusieurs rubans, 12
non-déterministes, 14
restriction à un alphabet binaire, 12
techniques de programmation, 9
variantes, 12
RAM, 17
RISC, 18
SRAM, 18
à k piles, 20
à compteurs, 20
mot
accepté par une machine de Turing, 5
refusé par une machine de Turing, 5
vide
notation, *voir* ϵ

partie
d'un ensemble
notation, *voir* $\subset$
parties
d'un ensemble
notation, *voir* $\mathcal{P}(E)$
problème
10ème problème de Hilbert, 2
de l'arrêt des machines de Turing
notation, *voir* HALTING − PROBLEM
produit cartésien
de deux ensembles
notation, *voir* $\times$

R, *voir* décidable
RE, *voir* récursivement énumérable
récusivement énumérable
notation, *voir* RE
réduction

notation, *voir* $\leq$, *voir* $\leq_m$
relation successeur entre configurations d'une
machine de Turing, 4, 5
notation, *voir* $\vdash$

systèmes de Post, 2

thèse de Church-Turing, 2, 23

union de deux ensembles
notation, *voir* $\cup$

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*. Mason.

[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.