La thèse de Church

Martinet Lucie

6 juin 2008

Table des matières

1 Introduction

La thèse de Church propose comme modèle de calcul les machines de Turing. Ici, nous ne chercherons pas à démontrer cette thèse, ce qui est impossible, ni à la réfuter. Nous présenterons plutôt les outils mathématiques qui permettent de l'énoncer et de la comprendre. Les méthodes mathématiques exposées ici sont à la base de celles utilisées pour l'implantation des langages de programmations sur machine.

La thèse de Church propose de définir la notion de calcul effectif comme calcul effectuable par machine de Turing. Un calcul effectif est un calcul qui démontre qu'une solution, à un problème donné, existe. De plus, ce calcul donne une méthode pour trouver cette solution, c'est à dire qu'il présente une procédure ou un calcul, qui se termine en un nombre fini d'étapes déterministes, ou ce qu'on appelle plus couramment un algorithme. Le problème est donc de définir la notion d'effectivité, qui se réfère à un *modèle de calcul*. C'est pourquoi, nous présenterons différents modèles de calcul, à commencer par les automates, dont nous montrerons les limites.

Dans une première partie, nous commencerons par donner quelques définitions sur les langages et les automates déterministes. Nous présenterons ensuite, les automates non déterministes et établirons leur équivalence avec les automates déterministes. Après cela, nous évoquerons quelques variantes d'automates, qui permettent de faciliter les constructions.

Comme les automates ne donnent pas la possibilité d'écrire, à l'inverse des machines séquentielles, nous étudierons brièvement ces dernières dans une seconde partie. En effet, celles-ci permettent l'impression d'un mot en cours de lecture, et sont utilisées, en particulier, pour simuler un calcul mental. Nous pouvons par exemple construire une machine séquentielle qui calcule le quotient de la division d'un entier par 4.

Dans une troisième partie, nous verrons de façon précise, quels sont les langages pouvant être reconnus par un automate. Pour cela, nous décrirons les langages rationnels et énoncerons le théorème de l'étoile.

Nous étudierons, dans une quatrième partie, les machines de Turing : il s'agit d'un autre modèle de calcul à accès séquentiel. Nous montrerons que ces machines peuvent reconnaître des langages et effectuer des calculs, que nous pourrions faire sur une feuille de papier, représentée par un ruban. Les machines de Turing peuvent se présenter sous différentes formes (un ou plusieurs rubans, de même que nous avons la possibilité d'organiser nos calculs sur une ou plusieurs feuilles). Nous verrons que ces différentes machines sont toutes équivalentes.
Nous terminerons cette partie par quelques exemples de construction de machines de Turing pour se convaincre de l'effectivité de celles-ci. Nous rappelons que la thèse de Church s'énonce de la manière suivante :

$$\text{calculable effectivement} \iff \text{calculable par machine de Turing}$$

D'après cette thèse, si une machine de Turing peut faire un calcul, c'est que ce calcul peut être décrit en un nombre fini d'étapes déterministes, ce qui paraît clair. Pour obtenir l'équivalence proposée par Church, il faut donc se convaincre

de l'implication inverse, c'est à dire que tout calcul qualifiable d'effectif est bien calculable par machine de Turing.

Pour cela, nous présenterons, dans une dernière partie, les machines à accès direct. Ces dernières sont beaucoup plus proches des langages de programmation que nous utilisons aujourd'hui. Enfin, nous montrerons leur équivalence avec les machines de Turing.

2 Quelques définitions sur les langages et automates

Les automates permettent de donner une version formalisée du calcul mental, c'est à dire sans écriture possible. Nous pouvons grâce à eux, décrire de manière simple, des langages relativement compliqués. Les automates, comme les machines de Turing, répondent à la question : le mot w appartient-il au langage $\mathcal{L}$? Ces deux modèles de calcul sont très utilisés en informatique car ils sont utiles dans plusieurs applications, notamment pour l'analyse lexicale. De plus, leur implémentation est assez facile et ils donc pratique à utiliser.

Avant d'aborder la définition concrète des automates, il nous faut définir les différents objets dont un automate est composé.

2.1 Les langages

Définition 2.1. On appelle *alphabet* tout ensemble fini non vide de symboles.

Les éléments d'un alphabet Σ sont appelés *lettres*. Un alphabet Σ est une suite finie de lettres.

On note Σ^* l'ensemble des mots pouvant être formés à partir des lettres de Σ. Les mots sont en fait une suite finie de lettres de Σ.

Définition 2.2. Soit w un mot de Σ^*.

On appelle *longueur* de w le nombre de lettres dont w est composé. La longueur du mot w se note $|w|$.

On note ε le mot vide, dont la longueur est zéro.

Définition 2.3. Soient v et w deux mots sur un même alphabet Σ.

On appelle *concaténation* de v et w le mot obtenu en écrivant, à la suite, les lettres de v, suivies des lettres de w.

La concaténation de v et w se note $v.w$ ou simplement vw.

La concaténation est une opération associative, dont l'élément neutre est le mot vide ε.

On note w^n la concaténation de n mots identiques w. Si $n = 0$, alors $w^n = \varepsilon$.

Définition 2.4. On appelle *langage* sur un alphabet Σ, toute partie de Σ^*.

Voici quelques exemples de langages

1. Le langage $\mathcal{L}_1$ formé des éléments de $\{0,1\}^*$ qui sont la représentation binaire d'un nombre divisible par 4.
2. Le langage $\mathcal{L}_2$ formé des éléments de l'alphabet $\{a,b\}$ qui sont la concaténation d'un mot formé d'une suite de a puis d'une suite de b et qui contient autant de a que de b. $\mathcal{L}_2$=$\{a^n b^n\}$.

3. Le langage $\mathcal{P}$ des parenthèses bien formées, défini comme suit :
   i) le mot vide ε appartient à $\mathcal{P}$.
   ii) si deux mots v et w appartiennent à $\mathcal{P}$, alors vw appartient encore à $\mathcal{P}$.
   iii) soit l'alphabet $\Sigma = \{(,)\}$, soit la fonction $\gamma : \Sigma^* \longrightarrow \Sigma^*$ qui à tout mot $w \in \Sigma^*$ associe (w).
   Si w appartient à $\mathcal{P}$, alors $\gamma(w) = (w)$ appartient aussi à $\mathcal{P}$.

Si $\mathcal{L}$ et $\mathcal{L}'$ sont deux langages sur un alphabet, alors on définit :

i) $\mathcal{L} \cup \mathcal{L}'$ *la réunion* des deux langages ;

ii) $\mathcal{L}\mathcal{L}'$ le *produit de concaténation*, qui est l'ensmble des mots pouvant s'écrire vw, avec $v \in \mathcal{L}$ et $w \in \mathcal{L}'$. On écrit $\mathcal{L}^2$ le langage $\mathcal{L}\mathcal{L}$.

iii) $\mathcal{L}^*$ la réunion des langages de la forme $\mathcal{L}^n$, pour n entier. Nous conviendrons que $\mathcal{L}^0$ est le langage vide $\{\varepsilon\}$.

Ces quelques définitions données, nous pouvons maintenant décrire ce qu'est un automate.

2.2 Les automates finis

Un automate fini est une machine pouvant se trouver dans un nombre fini de configurations différentes, aussi appelées *états*. L'automate reçoit une suite d'instructions (sous forme de signaux), qui provoque un changement d'état, aussi appelé *transition*. Un automate peut également être muni d'un dispositif lui permettant d'émettre un message lors de chaque transition.

2.2.1 Les automates déterministes : quelques définitions et notations

Définition 2.5. Un automate est constitué :

1. d'un *alphabet* fini Σ ;
2. d'un *ensemble* fini non vide d'*états*, noté Q ;
3. d'une *fonction de transition* δ :

$$\begin{aligned} \delta : Q \times \Sigma &\longrightarrow Q \\ (p, a) &\longmapsto q \end{aligned}$$

   que l'on écrira aussi :

$$p \xrightarrow{a} q;$$

4. d'un *état de départ* noté p_0 ;
5. d'un sous-ensemble F de l'ensemble des états. Les éléments de F sont appelés les *états finaux* ou encore *états acceptants*.

Définissons maintenant la fonction $\bar{\delta}$ à partir de δ, ce qui nous permettra de passer d'un état à un autre, non plus par une lettre, mais par un mot :

$$\bar{\delta} : Q \times \Sigma^* \longrightarrow Q$$

Par récurrence sur le mot w, on obtient :

1. $\bar{\delta}(q, \varepsilon) = q$;
2. $\bar{\delta}(q, wa) = \delta\left(\bar{\delta}(q, w), a\right)$ si a $\in \Sigma$

On définit le langage $\mathcal{L}(\mathbf{M})$ reconnu par un automate $\mathbf{M}$ comme le sous-ensemble de Σ^* défini par :

$$\mathcal{L}(\mathbf{M}) = \left\{ w : p_0 \xrightarrow{w} F \right\}$$

On dit aussi que l'automate $\mathbf{M}$ *accepte* w.

2.2.2 Interprétation graphique

Un automate peut être représenté de différentes manières.

Il peut être représenté sous forme de tableau à double entrée (états pour les lignes et lettres pour les colonnes), contenant les valeurs de la fonction de transition. Cette représentation n'est pas très visuelle mais permet l'implémentation de l'automate dans un langage de programmation quelconque.

Pour faciliter la lecture, nous choisirons plutôt de représenter nos automates, par un diagramme qui se lit de la manière suivante :

1. Les états sont représentés par des cercles contenant éventuellement le nom de l'état en question.
2. Les transitions sont suggérées par des flèches, étiquetées par la lettre correspondant à la transition considérée.
3. L'état de départ est marqué par une flèche et les états finaux sont doublement cerclés.

A chaque mot traité par l'automate, est associé un chemin du diagramme. Les mots acceptés par le langage représenté par l'automate, sont ceux qui aboutissent à un état final.

Définition 2.6. On appelle *rebut* un état qui n'est pas un état final, mais dont aucune des transitions ne permet d'aboutir à un état final. (L'automate est ainsi bloqué en boucle infinie et ne peut aboutir à un état final.)

La convention veut que tous les rebuts d'un automate n'apparaissent pas sur les diagrammes.

Quelques exemples d'automates.

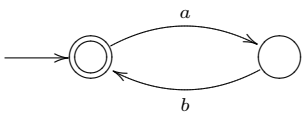


FIG. 1 – Automate reconnaissant le langage $(ab)^n$,$n \geq 0$

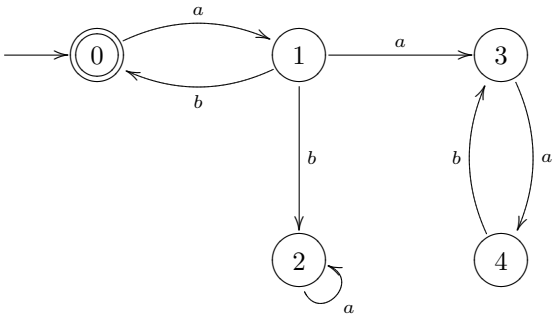


FIG. 2 – Automate reconnaissant le langage $(ab)^n$ avec trois rebuts.

Dans l'exemple ci-dessus, l'état 2 est un rebut. Les états 4 et 3 forment deux autres rebuts.

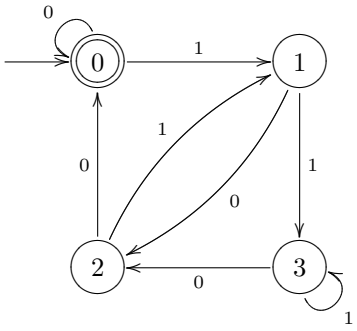


FIG. 3 – Automate reconnaissant les nombres divisibles par 4 en écriture binaire.

Explication de la figure 3.
Le nom des états q_i correspond au résultat de l'opération suivante :

$$\overline{w} \equiv q_i \, (mod.\, 4)$$

En effet, soit $\mathcal{L}$ le langage reconnu par l'automate ci-dessus. Si on considère que chaque mot est la représentation binaire d'un unique entier $\overline{w}$ alors on a :

1. $\overline{w0} = 2\overline{w}$
2. $\overline{w1} = 2\overline{w}+1$

Prenons l'exemple d'un nombre constitué de 3 lettres, noté w_3 on obtient alors l'arbre suivant :

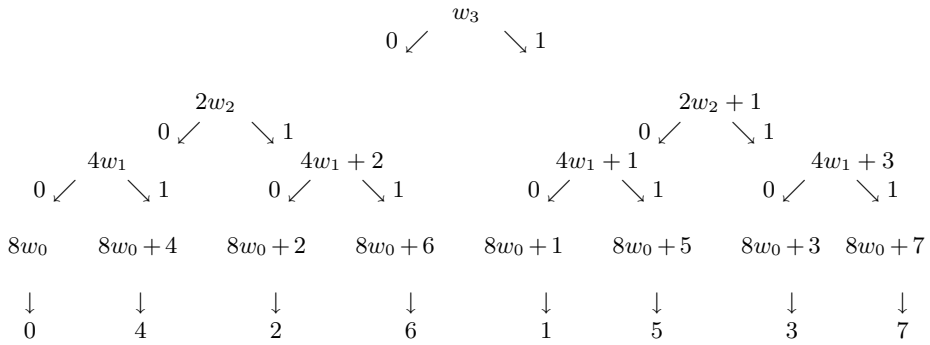


FIG. 4 – Arbre donnant tous les nombres pouvant être écrits en trois lettres sous forme binaire, leur valeur étant écrite sous forme décimale.

Pour lire les nombres sous leur forme binaire, il suffit de suivre les flèches en sens inverse et d'écrire, de gauche à droite, les lettres, dont ces dernières sont étiquetées.

Pour vérifier que le numéro des états correspond bien au résultat de l'opération citée ci-dessus, il suffit de suivre le chemin de chacun des mots, trouvés grâce à l'arbre, sur le diagramme de l'automate et de considérer l'état d'arrivée.

Si l'on est convaincu que l'automate calcule bien

$$\overline{w} \equiv q_i \, (mod.\, 4)$$

alors on comprendra que cet automate reconnaît uniquement le langage des nombres, en écriture binaire, qui sont divisibles par 4, car seul l'état 0 est un état final de l'automate et donc seuls les mots valant 0 modulo 4 sont acceptés par l'automate.

2.2.3 Les automates finis non déterministes

Les automates finis non déterministes diffèrent des automates finis déterministes car il est possible que plusieurs flèches étiquetées de la même lettre puissent partir du même état. De ce fait, à chaque mot peut correspondre plusieurs chemins. Les définitions obtenues pour les automates déterministes s'en trouvent donc quelque peu modifiées.

La fonction de transition δ prend maintenant ses valeurs dans *l'ensemble des parties non vides de* Q, noté $P^*(Q)$.
Ainsi, les différents éléments de $\delta(p, a)$ constituent un choix de transitions possibles correspondant à l'état p et à la lettre a. On définit par récurrence sur la longueur du mot w une relation :

$$p \xrightarrow{w} q$$

par les relations suivantes :

1. $\delta(p, \varepsilon) = q$ si $p = q$,
2. $\delta(p, a) = q$ si q appartient à $\delta(p, a)$,

3. $\overline{\delta}(p, wa) = q$ s'il existe un état r tel qu'on ait $\overline{\delta}(p, w) = r$ et $\delta(r, a) = q$.

Le langage reconnu par un automate fini non déterministe $\mathbf{M}$ est défini comme pour les automates déterministes :

$$\mathcal{L}(\mathbf{M}) = \{w : p_0 \xrightarrow{w} F\}$$

2.2.4 Équivalence entre les automates déterministes et non déterministes

Théorème 2.1. *(Pour la preuve du résultat, nous renvoyons le lecteur à l'ouvrage de Stern, [1], page 11)*
Tout langage reconnu par un automate fini non déterministe est également reconnu par un automate fini déterministe.

Remarque la réciproque de ce théorème est évidente car un automate fini déterministe est un cas particulier d'automate fini non déterministe.

Nous allons vérifier ce théorème sur des exemples concrets, à partir d'un algorithme précis.

Méthode algorithmique
Partons d'un automate $\mathbf{M}$ fini non déterministe, défini sur l'alphabet Σ, reconnaissant un certain langage $\mathcal{L}$, et construisons un automate $\mathbf{M}'$ fini déterministe reconnaissant ce même langage.

1. D'abord, nous partons de l'état initial q_0 de $\mathbf{M}$, qui sera aussi l'état initial q'_0 de $\mathbf{M}'$.
2. Considérons ensuite, l'ensemble des transitions qui partent de q_0 et regroupons en un seul état de $\mathbf{M}'$, tous les états q de $\mathbf{M}$ tel que $\delta(q_0, a)=q$. Ce nouvel état de $\mathbf{M}'$ est atteint avec la lettre a en partant de l'état q_0 de $\mathbf{M}'$ (ex : si les états 1, 3 et 4 de $\mathbf{M}$ sont atteints avec la lettre a, en partant de q_0, on construit un état $q_{1,3,4}$ de $\mathbf{M}'$, atteint avec la lettre a en partant de q_0). Procédons de la même manière pour chacune des lettres de l'alphabet qui permettent une transition à partir de q_0.
3. A partir des états précédemment obtenus, on construit les états suivants comme suit.
   Pour chaque lettre e appartenant à Σ, nous allons procéder de la même manière. Pour chacun des états de $\mathbf{M}$ inclus dans l'état de $\mathbf{M}'$ venant d'être construit, on détermine les états de $\mathbf{M}$ qui sont atteints par la transition associée à la lettre e considérée. On fait l'union de tous les états ainsi obtenus et l'on détermine le nouvel état de $\mathbf{M}'$, atteint à partir de l'état obtenu en 2. par la transition associée à la lettre e considérée. Si cet état n'existe pas encore, alors il faut le construire et recommencer la méthode à partir de 2. tant que tous les états de $\mathbf{M}'$ n'ont pas été traités.

Exemple du passage d'un automate non déterministe à un automate déterministe

Exemple 1. Les automates des figures 5 et 6 reconnaissent le même langage

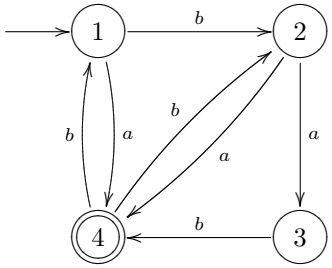


FIG. 5 – Un automate non déterministe.

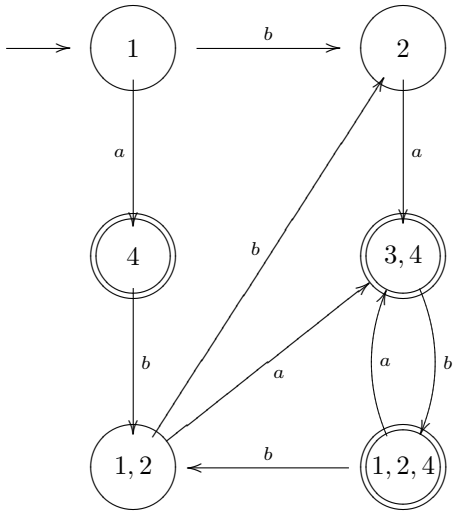


FIG. 6 – Un automate déterministe.

Nous allons montrer quelques étapes de la construction de l'automate de la figure 6, qui est un automate déterministe reconnaissant le même langage que celui de la figure 5 (qui est non déterministe), grâce à l'algorithme présenté ci-dessus. Pour faciliter l'explication, nous appellerons $\mathbf{M}$ l'automate de la figure 5 et $\mathbf{M}'$ l'automate de la figure 6. Nous désignerons les états numéro i de $\mathbf{M}$ par la notation q_i, $1 \leq i \leq 4$. Nous appelerons $q'_{i,j,k}$ un état q' de $\mathbf{M}'$, obtenu à partir des états i, j et k de $\mathbf{M}$, $1 \leq i, j, k \leq 4$ et $j \neq i \neq k$.

i) L'état initial de la figure 5 est l'état q_1, donc on commence par construire l'état q'_1, qui sera notre état initial.

ii) De l'état q_1, on peut passer avec la lettre a, uniquement à l'état final q_4, on crée donc un état final q'_4, atteint à partir de l'état q'_1 par la transition de la lettre a. De même avec la lettre b, on ne peut atteindre que l'état q_2, on crée donc l'état q'_2, atteint par la lettre b, à partir de l'état q'_1.

iii) Étudions maintenant le cas de l'état q_2. De cet état, on ne peut passer à un autre état qu'avec la transition a : il n'y a aucune transition b possible, ce cas est donc réglé. Ensuite, de q_2, on peut passer à l'état q_3 ou à l'état

final q_4, on construit donc un état final de M' $q'_{3,4}$ atteint par la transition a et venant de l'état q'_2.

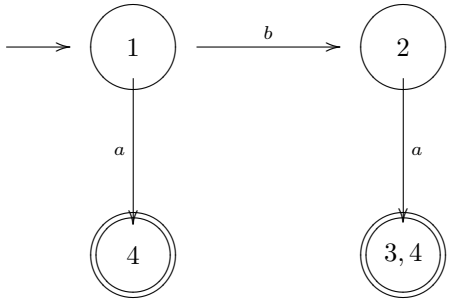


FIG. 7 – Première étape de la construction de l'automate déterministe.

iv) Continuons avec l'état $q'_{3,4}$. Pour déterminer l'état suivant, on étudie d'abord le cas de la transition par la lettre a sur $\mathbf{M}$. De q_3, il n'y a pas de transition possible avec a, de q_4 il n'y a pas de transition possible, donc de l'état de $\mathbf{M'}$ $q'_{3,4}$ il n'y aura aucune transition a possible. Passons maintenant à la transition par b. Nous remarquons que de q_3 on passe à q_4 par la transition b et de q_4 on arrive à q_2 et à q_1 par b. On construit donc un état $q'_{1,2,4}$ de $\mathbf{M'}$, atteint par la transition b, en venant de $q'_{3,4}$. On obtient donc l'étape 2 suivante

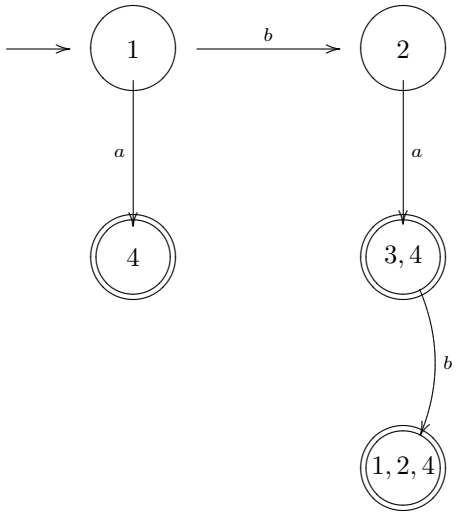


FIG. 8 – Étape 2.

v) Nous procédons de la même façon pour déterminer les états atteints à partir de $q'_{1,2,4}$. La transition par la lettre a peut se faire à partir de l'état q_1 de $\mathbf{M}$ vers q_4, à partir de l'état q_2 vers q_4 et q_3. L'état de $\mathbf{M'}$ atteint par la transition a à partir de $q'_{1,2,4}$ est donc $q'_{3,4}$, qui existe déjà. Continuons la construction avec la lettre b. Nous obtenons le nouvel état final $q_{1,2}$. Cela donne :

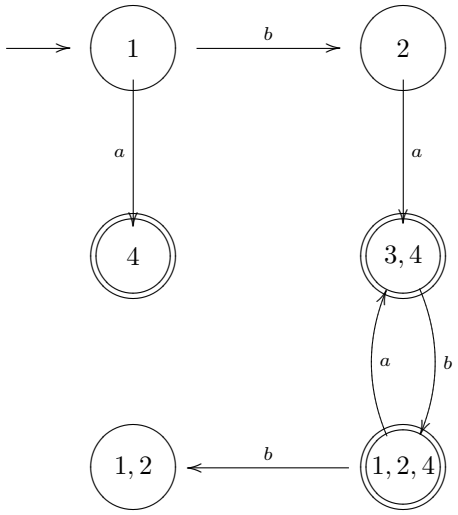


FIG. 9 – Étape 3.

vi) Le cas de l'état $q'_{3,4}$ étant déjà traité, on ne s'occupe que du nouvel état, de la même façon que les états précédents et on obtient une transition a allant à l'état $q'_{3,4}$ et une allant vers q'_2 atteint avec b, dont le cas a déjà été traité.

vii) Il ne faut pas oublier le cas de l'état q'_4 que l'on avait laissé de côté au début de la construction et l'on obtient l'automate de la figure 6.

Inconvénient de la méthode

Il est possible que l'application de cet algorithme mène à une «explosion combinatoire» : à partir d'un automate non déterministe $\mathbf{M}$ à n états, on peut construire un automate déterministe ayant jusqu'à 2^n-1 états (le nombre de toutes les combinaisons possibles entre les états de $\mathbf{M}$).

2.2.5 Les transitions instantanées et comment les éliminer

Les transitions instantanées permettent de passer d'un état à un autre, sans utiliser de lettre, ou plutôt en formant le mot vide.
Pour introduire ces transitions instantanées il nous faut donc la fonction de transition δ :

$$\delta : Q \times \Sigma \longrightarrow P^*(Q)$$

et la fonction de transition instantanée τ :

$$\tau : Q \longrightarrow P(Q)$$

où *P(Q)* représente l'ensemble des parties de Q.

Définition 2.7. On redéfinit de nouveau la relation :

$$\delta : P \xrightarrow{w} Q$$

par récurrence sur la longueur du mot w, comme suit :

1. $p \xrightarrow{\varepsilon} q$ si q appartient à l'ensemble des états pouvant être atteints à partir de p de façon instantanée.
2. $p \xrightarrow{wa} q$ s'il existe des états r, r' tels que $p \xrightarrow{w} r$, r' appartient à $\delta(r, a)$ et q appartient à l'ensemble des états pouvant être atteints à partir r' de façon instantanée.

Ces transitions instantanées peuvent en fait être éliminées.

Théorème 2.2. *(pour la preuve de ce résultat, nous renvoyons le lecteur à l'ouvrage de Stern, [1], page 15)*
Tout langage reconnu par un automate fini non déterministe avec transitions instantanées est également reconnu par un automate fini déterministe.

Pour construire cet automate déterministe, il suffit d'identifier l'état d'où part la transition à l'état dans lequel elle arrive. Pour le reste de la construction, nous reprendrons essentiellement la même méthode que dans la section 2.2.3.

Exemple
L'automate ci-dessous reconnaît le langage $a(b)^n a$, avec n appartient aux entiers naturels (i.e : n=0 est possible).

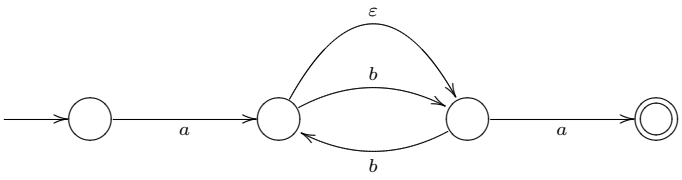


FIG. 10 – Automate avec transition instantanée reconnaissant un langage sur l'alphabet $\{a, b\}^*$

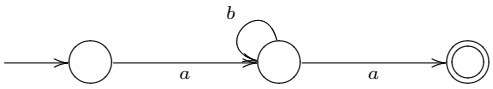


FIG. 11 – Automate sans transition instantanée reconnaissant le langage $a(b)^n a$.

3 Les machines séquentielles

Les machines séquentielles sont des automates munis d'un dispositif permettant d'émettre un message (sous forme d'une suite finie de symboles) lors de chaque transition. Elles nous permettent donc, en plus de reconnaître des langages, de faire du calcul mental comme nous allons le voir. Nous étudierons uniquement les machines séquentielles déterministes.

Définition 3.1. Une *machine séquentielle* est constituée :

1. de deux *alphabets* Σ et Γ ;
2. d'un ensemble fini non vide d'états Q ;
3. d'une *fonction de transition* $\delta : Q \times \Sigma \longrightarrow Q$;
4. d'une *fonction d'impression* $\sigma : Q \times \Sigma \longrightarrow \Gamma^*$;
5. d'un *état de départ* noté p_0 ;
6. d'un sous-ensemble F d'états de Q.

De même que l'on peut étendre δ en une fonction $\overline{\delta}$définie sur $Q \times \Sigma^*$, on peut étendre σ en $\overline{\sigma}$ définie sur $Q \times \Sigma^*$ en posant :

$$\overline{\sigma}(p, \varepsilon) = \varepsilon$$
$$\overline{\sigma}(p, wa) = \overline{\sigma}(p, w)\, \sigma\left(\overline{\delta}(p, w), a\right)$$

Définition 3.2. Si $\mathbf{M}$ est une machine séquentielle, on appelle fonction *calculée* par $\mathbf{M}$ la fonction :

$$f_{\mathbf{M}} : \Sigma^* \longrightarrow \Gamma^*$$

définie par : $f_{\mathbf{M}}(w) = \overline{\sigma}(p_0, w)$.

On note $\mathcal{L}_q$ le langage accepté par l'automate défini à partir de $\mathbf{M}$ à l'aide de l'unique état final q ; les langages $\mathcal{L}_q$ réalisent, quand q décrit Q, une partition de Σ^* et $f_{\mathbf{M}}$ est l'unique fonction $f : \Sigma^* \longrightarrow \Gamma^*$ définie par :

$$f(\varepsilon) = \varepsilon,$$
$$f(wa) = f(w)\, \sigma(q, a), \text{ pour } w \in \mathcal{L}_q.$$

La représentation graphique des machines séquentielles est assez similaire à celle des automates, à la différence près, que chaque flèche de transition, est doublement étiquetée.
La fonction de transition est représentée par une lettre se situant à l'origine de la flèche, et la fonction d'impression, par une lettre placée au milieu de la flèche.

Voici deux exemples de machine séquentielle

Exemple 2. Nous allons d'abord présenter un exemple simple de machine séquentielle. Nous allons construire une machine qui reconnait le langage $(ab)^n$ et qui imprime les lettres qu'elle lit, en majuscule. L'alphabet d'impression est donc $\{A, B\}$. La machine que nous obtenons est celle de la figure 12.

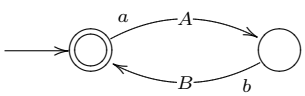


FIG. 12 – Machine à accès séquentiel changeant les lettres du langage $(ab)^n$ en majuscules

Exemple 3. Reprenons l'exemple de la figure 3, que l'on transforme en machine séquentielle, calculant la division entière par 4 du nombre codé en binaire par w.

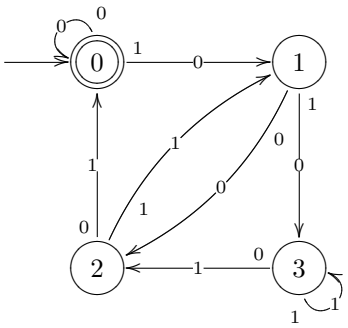


FIG. 13 – Machine séquentielle calculant la division par 4 d'un nombre donné.

Nous allons vérifier que la machine séquentielle, représentée ci-dessus, calcule bien la division entière par 4 d'un nombre codé en binaire, de la même façon que nous avons vérifié que l'automate de la figure 3 reconnaissait les nombres divisibles par 4. Reprenons l'exemple d'un nombre constitué de 3 chiffres, noté w_3 on obtient alors l'arbre de la figure 14.

En suivant le sens inverse des flèches et en écrivant de gauche à droite les chiffres à la racine de ces flèches, on obtient le nombre dont la machine cherche le quotient dans la division par quatre.
En procédant de même avec les chiffres inscrits au milieu des flèches, on obtient le quotient du nombre considéré par la machine, en écriture binaire.
Lorsqu'il y a plusieurs possibilités (nombres en gras soulignés ou nombres écrits normalement), nous avons pris pour convention, de ne considérer soit, que les nombres en gras soulignés, si l'on remonte les flèches en étant parti d'un nombre en gras souligné, soit, que les nombres écrits normalement, si l'on commence par un nombre écrit normalement.

4 Caractérisation des langages reconnus par un automate

Nous allons maintenant définir ce qu'est un langage rationnel. Il s'agit, en fait, d'un langage reconnaissable par un automate comme nous le verrons plus loin.

4.1 Les langages rationnels

Définition 4.1. Soit Σ un alphabet.
La classe des *langages rationnels d'alphabet* Σ est la plus petite classe qui a les propriétés suivantes :

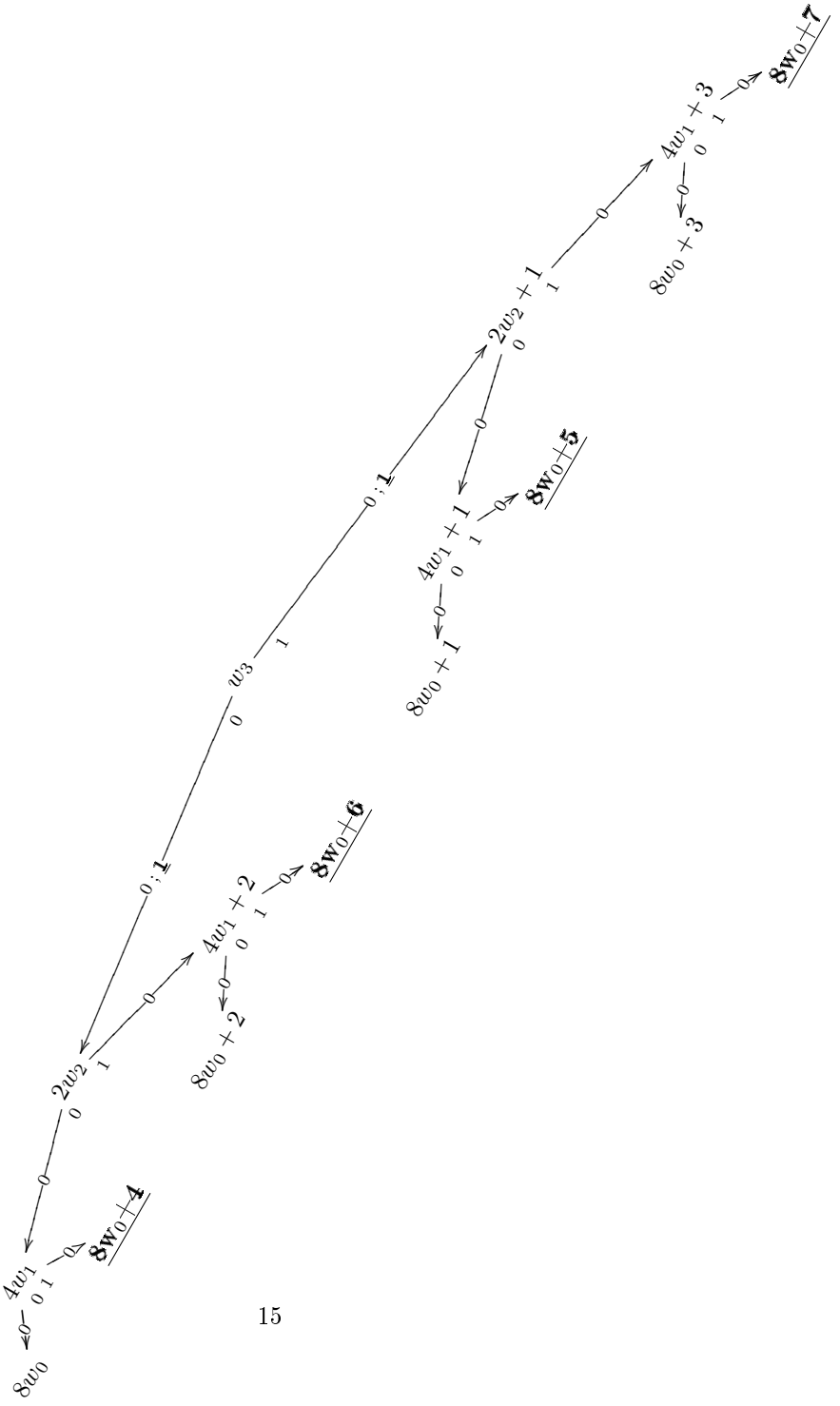
i) le langage $\emptyset$ est rationnel ;

ii) le langage réduit au seul mot vide est rationnel ;

iii) Pour chaque lettre a, le langage réduit au seul mot a est rationnel ;

iv) Si $\mathcal{L}$ et $\mathcal{L}$' sont rationnels, alors $\mathcal{L} \cup \mathcal{L}$', $\mathcal{L}\mathcal{L}$' et $\mathcal{L}^*$ sont rationnels.

FIG. 14 – Arbre donnant tous les nombres pouvant être écrits en trois lettres sous forme binaire et le résultat de leur division par 4.



Théorème 4.1. *Tout langage rationnel est reconnu par un automate fini.*

Définition 4.2. On appelle automate *normalisé* tout automate possédant un seul état final f distinct de l'état initial et dont aucune transition n'est issue de f.

Pour démontrer le théorème, nous allons d'abord présenter la construction des automates reconnaissant :

i) Le langage vide :
c'est un automate ayant un seul état qui est un rebut

ii) Le langage composé du seul mot vide :
c'est un automate dont l'état de départ est un état final et dont les autres états sont des rebuts

iii) Le langage composé uniquement du mot a :

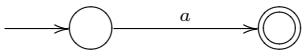


FIG. 15 – Automate reconnaissant le langage a.

iv) Le langage issu de la réunion de deux langages $\mathcal{L}$ composé du seul mot a et $\mathcal{L}'$ composé du seul mot b. Chacun des deux langages est représenté par un automate normalisé ayant un état initial respectivement p_0 et p'_0.

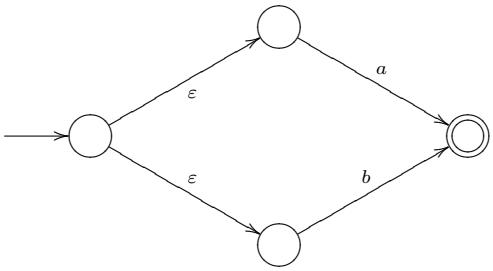


FIG. 16 – Automate reconnaissant le langage $a \cup b$

v) Le langage issu de la concaténation de deux langages $\mathcal{L}$ et $\mathcal{L}'$ représentés chacun par un automate normalisé ayant un état initial respectivement p_0 et p'_0 et constitués respectivement du mot a et du mot b. Nous noterons C l'état initial de L', que l'on identifie à l'état final de $\mathcal{L}$.

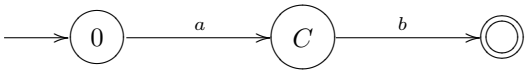


FIG. 17 – Automate reconnaissant la concaténation des deux langages a et b.

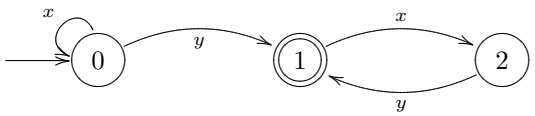
vi) L'automate reconnaissant l'étoile d'un langage.
Ce dernier est l'union des langages issus du produit de concaténation d'un

langage avec lui-même. Ces deux opérations ont été décrites ci-dessus. Il est donc possible de construire un automate qui reconnait l'étoile d'un langage rationnel.

Grâce à toutes ces constructions, on peut, à partir d'une expression rationnelle qui dénote le langage $\mathcal{L}$, construire un automate fini non déterministe avec transitions instantanées, qui reconnait $\mathcal{L}$. Si on veut obtenir un automate fini déterministe, il suffit d'utiliser la méthode que nous avons décrite dans le paragraphe 2.2.5.

Théorème 4.2. *(Pour une preuve plus formelle, nous renvoyons le lecteur à l'ouvrage de Stern, [1], page 27) Tout langage reconnu par un automate est rationnel.*

Comprenons ce théorème sur un exemple simple : nous allons décrire sur un exemple, un algorithme qui permet de trouver le langage que représente un automate.
Pour trouver le langage reconnu par l'automate suivant :



Nous allons construire des tableaux de cas possibles de transitions.

Pour ce faire, nous appellerons d, l'état duquel on part, et a, l'état d'arrivée.
Nous utiliserons aussi un état nommé k, qui représente les états qui peuvent servir d'états intermédiaires, entre les états d et a.

Si k est égal à 2, alors les états pouvant être intermédiaires sont les états 0 ou 1.
Après ces quelques précisions, nous allons appliquer les formules suivantes, pour chacun des tableaux que l'on va construire. Ces tableaux vont répertorier les mots pouvant être formés, en partant de chacun des états d, et arrivant aux états a. Ces tableaux dépendront de k.

$t[d,a,0]$ représente l'union des lettres permettant la transition de l'état d à l'état a.

$t[d,d,0]$ représente l'union du langage vide et des lettres permettant la transition de l'état d dans lui-même.

Si $d \neq k$ et $a \neq k$, alors on applique la formule suivante

$$t[d,a,k+1]{=}(t[d,a,k] \cup t[d,k,k](t[k,k,k])^* t[k,j,k])$$

Sinon, si $d \neq k$, alors on a :

$$t[d,k,k+1]{=}t[d,k,k](t[k,k,k])^*$$

Sinon, si $a \neq k$, alors :

$$t[k,a,k+1]=(t[k,k,k])^*t[k,a,k]$$

Sinon, on applique :

$$t[k,k,k+1]=(t[k,k,k])^*$$

Si une transition est impossible, la case correspondante du tableau sera vide.
Sur notre exemple cela donne :

	d=0	d=1	d=2
a=0	$\{\varepsilon\} \cup x$		
a=1	y		y
a=2		x	

FIG. 18 – Tableau $t[d,a,0]$

	d=0	d=1	d=2
a=0	x^*		
a=1	x^*y	$\{\varepsilon\}$	y
a=2		x	$\{\varepsilon\}$

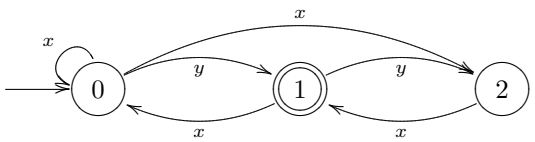
FIG. 19 – Tableau $t[d,a,1]$

	d=0	d=1	d=2
a=0	x^*	$\{\varepsilon\}$	
a=1	x^*y	$\{\varepsilon\}$	y
a=2	x^*yx	x	$\{\varepsilon\} \cup yx$

FIG. 20 – Tableau $t[d,a,2]$

Le langage reconnu par l'automate est $t[0,1,3] = x^*y \cup x^*yx(\{\varepsilon\} \cup yx)^*y = x^*y(x^*y)^*$.

Voici un autre exemple un peu plus compliqué :



Le langage reconnu par l'automate est $t[0,1,3] = x^*y(xx^*y)^* \cup x^*x \cup x^*y(xx^*y)^*y \cup xx^*y(x(\ xx^*y)^*\ (y \cup x^*y))^*x(xx^*y^*)$.

	d=0	d=1	d=2
a=0	$\{\varepsilon\} \cup x$	x	
a=1	y	$\{\varepsilon\}$	x
a=2	x	y	$\{\varepsilon\}$

FIG. 21 – Tableau $t[d, a, 0]$

	d=0	d=1	d=2
a=0	x^*	xx^*	
a=1	x^*y	xx^*y	x
a=2	x^*x	$y \cup xx^*y$	$\{\varepsilon\}$

FIG. 22 – Tableau $t[d, a, 1]$

	d=0	d=1	d=2
a=0	$x^* \cup \mathrm{a}^* y x^* x x^* y$	$(xx^*y)^* xx^*$	$x(xx^*y)^* xx^*$
a=1	$x^* y(xx^*y)^*$	$(xx^*y)^*$	$x(xx^*y^*$
a=2	$x^*x \cup x^*y(xx^*y)^*y \cup xx^*y$	$(xx^*y)^*y \cup xx^*y$	$x(\ xx^*y)^*\ (y \cup xx^*y)$

FIG. 23 – Tableau $t[d, a, 2]$

4.2 Les automates minimaux

L'automate minimal d'un langage $\mathcal{L}$ est l'automate qui possède le plus petit nombre d'états possibles et qui reconnait ce langage. C'est un automate dont la fonction de transition ne peut avoir le même résultat lorsqu'elle est appliquée, sur un même mot, à deux états différents.

4.2.1 Définition

Définition 4.3. On dit qu'un langage $\mathcal{L}$ *sépare* deux mots v et w de Σ^* si v appartient à $\mathcal{L}$ et w n'appartient pas à $\mathcal{L}$ ou inversement.

Définition 4.4. Deux mots v et w sont *congrus* modulo $\mathcal{L}$ si quel que soit le mot x de l'alphabet Σ^*, $\mathcal{L}$ ne sépare pas xw et xv.

Exemple 4. Soit $\mathcal{L}$ le langage a^*ba^*, sur l'alphabet a, b. Ce langage n'admet pas les mots contenant plus d'un b.

Soit $v = b$ et $w = ba$. Prenons n'importe quel mot x de $\mathcal{L}$, par exemple b. Alors $vx = bb$ qui n'appartient pas à $\mathcal{L}$ et $wx = bab$ appartient à $\mathcal{L}$. Les deux mots ne sont donc pas congrus modulo $\mathcal{L}$.

Si on prend les mots $v = baa$ et $w = baaa$, prenons n'importe quel mot x appartenant à l'alphabet considéré, on obtient : $vx = baax$ et $wx = baaax$.

On voit facilement que ces deux mots sont congrus modulo $\mathcal{L}$ car si x n'est composé que de a, alors vx et wx appartiennent tous deux à $\mathcal{L}$.
Si x contient au moins b, les deux mots vx et wx seront constitués de plus d'un b, donc ces deux mots n'appartiendront pas au langage $\mathcal{L}$.

La relation définie ci-dessus est en fait une relation d'équivalence. Ici, les classes d'équivalence seront appelées classes de congruence modulo $\mathcal{L}$. Voici un théorème que nous ne démontrerons pas mais qui va nous permettre de construire les automates minimaux.

Théorème 4.3. *(Pour la preuve du résulat, nous renvoyons le lecteur à l'ouvrage de Jacques Stern,[1], page 31)*
Un langage est reconnu par un automate fini si et seulement si les classes de congruence modulo $\mathcal{L}$ sont en nombre fini.

En effet, si l'on considère un automate fini déterministe $\mathbf{M}$, reconnaissant le langage $\mathcal{L}$, et si deux mots sont tels que $\delta(q_0, v) = \delta(q_0, w)$, alors, quelque soit le mot x, on a $\delta(q_0, vx) = \delta(q_0, wx)$, donc les deux mots v et w ne sont pas séparés par $\mathcal{L}$, qui est reconnu par $\mathbf{M}$, et ces deux mots sont de la même classe de congruence modulo $\mathcal{L}$. La relation $\delta(q_0, v) = \delta(q_0, w)$ implique la relation de congruence et n'a qu'un nombre fini de classes : au plus le nombre d'états de $\mathbf{M}$, donc la relation de congruences modulo $\mathcal{L}$, a aussi un nombre fini de classes.

Le reste de la preuve consistera à montrer que l'autre implication : si un langage $\mathcal{L}$ a la propriété de définir un nombre fini de classes d'équivalence, et si $v \equiv w \mod \mathcal{L}$ alors, on aura la relation $\delta(q_0, v) = \delta(q_0, w)$.

Grâce à la preuve du théorème ci-dessus, on constate que le nombre minimal d'états, pour un automate reconnaissant un langage rationnel, est précisément le nombre de classes de congruences modulo $\mathcal{L}$, qu'il définit.

Un automate minimal est donc, un automate qui n'a pas de rebut, et qui ne peut avoir deux états distincts q et p tels que pour tout mot w, $\delta(q, w)$ est un état final, si et seulement si, $\delta(p, w)$ est un état final.
Un automate est donc minimal, s'il ne possède aucun rebut, et si tous ses états distincts ne sont pas équivalents deux à deux. Autrement dit, si à partir de deux états distincts de $\mathbf{M}$, on peut écrire le même mot w, alors ces deux états sont dits équivalents.

Nous allons maintenant décrire un algorithme permettant de construire un automate minimal, à partir d'un automate fini déterministe.

Le principe repose sur la construction de partitions que l'on affine au fur et à mesure.

On commence avec la partition suivante : l'ensemble des états finaux, que nous noterons F, et l'ensemble des états non acceptants $Q - F$, qui sont chacun des sous-ensembles de Q. On choisit ensuite, une lettre a de l'alphabet, sur lequel est construit L, et un sous-ensemble, que nous noterons X_1. Pour chacun des sous-ensembles X_i qui composent la partition, on procède comme suit.
Le but est de scinder en deux le sous-ensemble X_i considéré (celui-ci peut être X_1). Pour cela, on regroupe dans un même sous-ensemble X_i', les états q, qui appartiennent à X_i, et qui vérifient la propriété suivante : $\delta(q, a) = p$, avec p appartient à X_1. Dans un autre sous-ensemble X_i", on regroupe tous les états de X_i, qui ne vérifient pas cette propriété. Ainsi, le sous-ensemble X_i est

remplacé par les deux sous-ensembles X_i' et X_i" dans la partition de départ. Cette nouvelle partition peut être, elle aussi, raffinée par la même méthode. Pour obtenir la partition la plus fine, on raffine chaque nouvelle partition, jusqu'à obtenir une partition qui ne peut plus être affinée : pour chacune des lettres de l'alphabet et chacun des sous-ensembles, aucun sous-ensmble de la partition n'est scindé en deux. Si la partition finale ne contient que des sous-ensembles à un état, cela signifie que l'automate de départ était déjà l'automate minimal reconnaissant le langage rationnel $\mathcal{L}$.

Lorsque l'on a obtenu la partition la plus fine, on construit les états qui sont les sous-ensembles de la partition, et on les relie par les transitions qui conviennent.

Voici un exemple de minimisation à partir de l'automate de la figure 24.

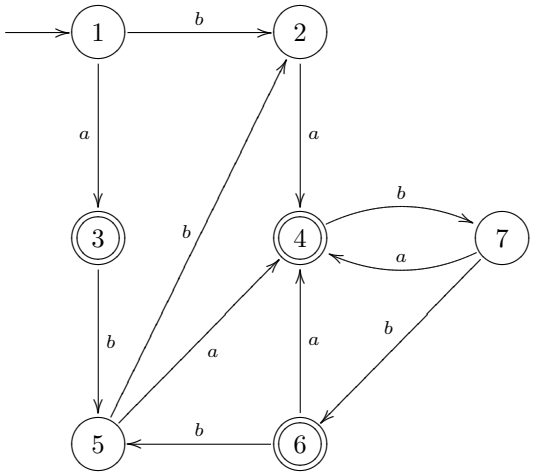


FIG. 24 – Un automate déterministe définissant un langage

Nous allons construire un automate minimal de cet automate. La partition de départ est constituée de deux ensembles : celui des états finaux (3,4,6) et celui des autres états (1,2,5,6,7). Ici il n'y a pas de rebut.

Nous allons établir une partition plus fine. Nous commencerons par considérer la lettre b et le sous-ensemble (3,4,6).
Nous obtenons ainsi une nouvelle partition : (3,4,6) (1,2,5) (7). Avec cette nouvelle partition, considérons la transition b et le sous-ensemble (7). Nous obtenons la nouvelle partition : (3,6) (4) (1,2,5) (7). On continue avec la lettre a et le sous-ensemble (4), on obtient : (3) (6) (4) (1) (2,5) (7). Il faut maintenant considérer la transition a et chacun des sous-ensembles que l'on a, pour essayer de séparer (2,5) et faire de même avec b. Si ces deux états ne sont pas séparés, c'est qu'ils sont équivalents, ce qui est le cas ici.
On obtient donc l'automate minimal suivant :

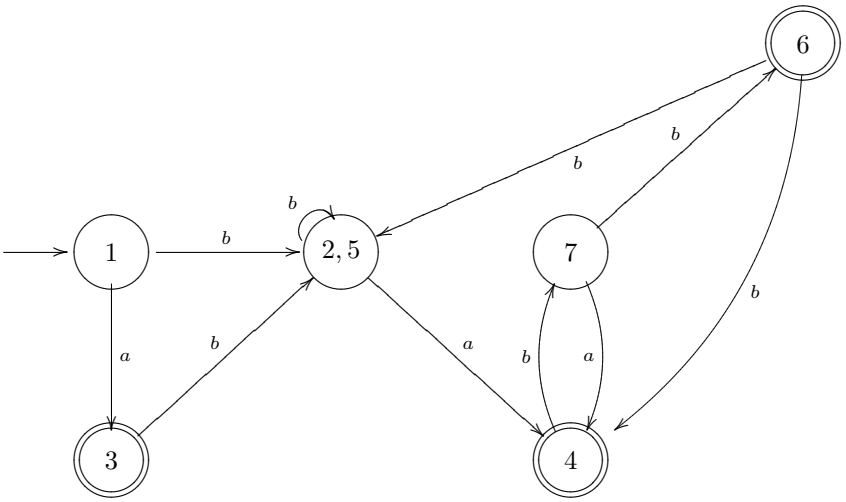


FIG. 25 – Un automate déterministe

4.2.2 Le lemme de l'étoile

Nous allons énoncer un théorème très important. En effet, celui-ci permet de reconnaître les langages rationnels , langages pouvant être reconnus par un automate.

Théorème 4.4. *(Pour la preuve de ce résultat, nous renvoyons le lecteur à l'ouvrage de Stern, [1], page 38)*
Soit $\mathcal{L}$ un langage rationnel sur l'alphabet Σ et soit n le nombre d'états de l'automate minimal qui reconnaît $\mathcal{L}$. Tout mot w de $\mathcal{L}$ de longueur $\geqslant n$ peut s'écrire sous la forme uxv de façon que l'on ait les propriétés suivantes :
i) $\mid ux \mid \leqslant n$
ii) $x \neq \varepsilon$
iii) $u\,(x)^*\,v$ *est inclus dans* $\mathcal{L}$.

Corollaire 4.5.

i) Le langage $a^n b^n$ $n \geq 1$ n'est pas rationnel.

ii) Le langage des parenthèses bien formées n'est pas rationnel.

Preuve 4.1. Nous ne démontrerons ici que le point ii) car les démonstrations sont similaires.
Supposons que le langage des parenthèses bien formées soit rationnel.
Notons n, le nombre d'états qui constitue l'automate minimal qui reconnaît ce langage, et appliquons le lemme de l'étoile au mot $(^n\ )^n$.
On obtient alors une décomposition

$$(^n\ )^n = uxv$$

où u, x, v sont des mots qui réalisent les conditions du lemme. On a alors $\mid ux \mid \leq n$, autrement dit, $ux = (^n$ et donc x est composé uniquement de parenthèses ouvrantes et s'écrit : $x = (^i$ avec $i \leq n-1$ et $i \neq 0$.
Le mot ux^2v s'écrit alors $(^{n+i}b^n$, mot qui ne peut appartenir au langage des parenthèses bien formées, ce qui contredit le lemme de l'étoile.

Ainsi, les automates finis permettent de reconnaître les langages rationnels ; les machines séquentielles nous permettent en plus, de faire du calcul mental : avec ces deux modèles de calcul, nous n'avons pas la possibilité d'utiliser de la mémoire.

Nous allons maintenant aborder un autre modèle de calcul qui nous permet de décrire tous les calculs effectifs (calculables par algorithme) et qui nous permettra de reconnaître le langage $a^n b^n$ $n \geq 1$.

5 Les machines de Turing

Nous commencerons par décrire les machines de Turing à un ruban.

5.1 Les machines à un ruban

Définition 5.1. Une *machine de Turing* est la donnée :
i) d'un alphabet fini Σ auquel on ajoute la lettre vide que l'on peut noter $\square$ ou ε, que nous noterons Σ^+ ;
ii) d'un ensemble non vide d'états Q ;
iii) d'une *fonction d'impression* $\sigma : Q \times \Sigma \longrightarrow \Sigma$;
iv) d'une *fonction de transition* $\delta : Q \times \Sigma \longrightarrow Q$;
v) d'une *fonction de déplacement* Δ définie sur $Q \times \Sigma$ à valeurs dans un ensemble à deux éléments, noté $\{+, -\}$;
vi) d'un *état de départ* noté q_0 ;
vii) d'un sous-ensemble F de l'ensemble des états, appelés *états finaux*.

Définition 5.2. On appelle *configuration* de la machine de Turing l'octuplet suivant : $(Q, \Sigma, \square, \sigma, \delta, \Delta, q, F)$ où Q est l'ensemble des états, Σ l'alphabet sur lequel les mots sont écrits, $\square$ le symbole blanc qui nous permet de déparer les mots entre eux, σ, δ, Δ les fonctions décrites plus haut et F l'ensemble des états finaux.

Le principe général d'une machine de Turing est le suivant : partant d'une configuration initiale, la machine va progressivement changer la configuration de la mémoire, représentée par un ruban de cases potentiellement infini, car nous ne mettons aucune restriction sur la taille des mots, qui seront traités par la machine. Chaque case contient un élément de Σ^+, mais seulement un nombre fini de cases contiennent une lettre de Σ. Pour accéder à un élément du ruban, on dispose d'une tête de lecture positionnée sur une case du ruban, lorsqu'on se trouve à l'état q.

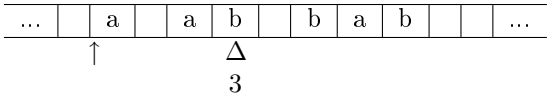


FIG. 26 – Un exemple de configuration de machine de Turing

Dans cette configuration, la case accessible est la numéro 4, elle contient la lettre b, nous sommes à l'état 3.

La flèche indique la case de départ, nous ne la ferons plus figurer par la suite. Le triangle représente la tête de lecture.
Pour simuler les changements de configuration produits par une machine de Turing, nous représenterons les instructions de la machine par un tableau à double entrée, contenant trois valeurs par case. Ces trois valeurs sont les résultats des trois fonctions énoncées plus haut, dans l'ordre suivant : la fonction d'impression, la fonction de déplacement et la fonction de transition. Pour que les calculs se terminent, nous conviendrons que des états terminaux ont été fixés, soit acceptants, soit refusants. Nous conviendrons également, qu'à partir d'une configuration, dont l'état est terminal, plus aucune transition n'est possible. Nous supposerons qu'il n'existe qu'un unique état acceptant et un unique état refusant. Nous utiliserons la lettre $\top$ pour l'état acceptant et la lettre $\bot$ pour l'état refusant. Pour tous les ensembles d'états Q nous supposerons que $Q \cap \{\top, \bot\}$ est vide.

Exemple 5. Voici une machine de Turing et les suites de configurations du calcul effectué par la machine sur un mot abb

	a	b	□
→0	a,+,1	$\bot$	$\bot$
1	$\bot$	b,+,0	$\top$

FIG. 27 – La machine

Le calcul présenté ici se termine en deux étapes, et le mot n'est pas accepté par la machine.

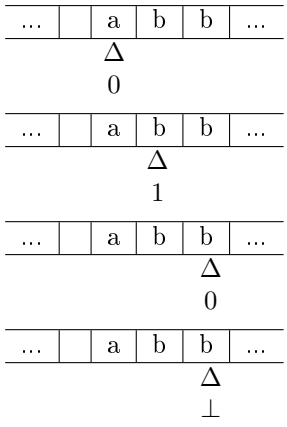


FIG. 28 – Les différentes configurations

Le modèle de calcul présenté ici nous permet de reconnaître un langage et de faire des calculs, comme nous allons le voir au cours des prochains paragraphes.

5.2 Reconnaissance d'un langage par une machine de Turing

Définition 5.3. Soit **M** une machine de Turing d'alphabet Σ, et w un mot sur Σ. On dit que **M** *accepte* (respectivement *refuse*) w si le calcul de **M** à partir de la configuration initiale associée à w mène, en un nombre fini d'étapes, à une configuration terminale acceptante (respectivement refusante).

Définition 5.4. Soit **M** une machine de Turing d'alphabet Σ, $\mathcal{L}$ un langage d'alphabet Σ. On dit que **M** décide $\mathcal{L}$ si **M** accepte tous les mots w de $\mathcal{L}$ et refuse tous les mots v n'appartenant pas à $\mathcal{L}$. On dit qu'un langage $\mathcal{L}$ est *MT-décidable* ou *décidable par machine de Turing* s'il existe au moins une machine de Turing qui décide ce langage.

Pour faciliter la démarche, la machine de Turing qui décide un langage $\mathcal{L}$ peut utiliser des lettres additionnelles n'appartenant pas à l'alphabet de $\mathcal{L}$.

Nous allons tout d'abord donner un exemple de machine de Turing qui décide un langage décidable par automate c'est à dire rationnel.
Certaines cases du tableau peuvent être vides : cela signifie que la machine ne pourra jamais se trouver dans la configuration représentée par cette case.

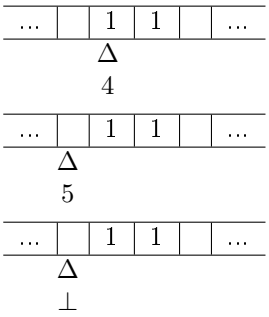
Exemple 6. Reprenons l'exemple du langage constitué de nombres divisibles par quatre, et dont on peut calculer le reste modulo 4.

	0	1	□
→0	0,+,0	1,+,0	□,-,1
1	0,-,2	1,-,4	
2	□,-,3,	1,-,5	
3	□,-,3	□,-,3	⊤
4	□,-,5	1,-,5	⊥
5	□,-,5	□,-,5	⊥

FIG. 29 – Tableau de changement de configuration de la machine

Vérifions maintenant sur un petit nombre tel que 3, écrit en base 2 :

...		1	1		...
		Δ			
		0			
...		1	1		...
			Δ		
			0		
...		1	1		...
				Δ	
				0	
...		1	1		...
			Δ		
			1		



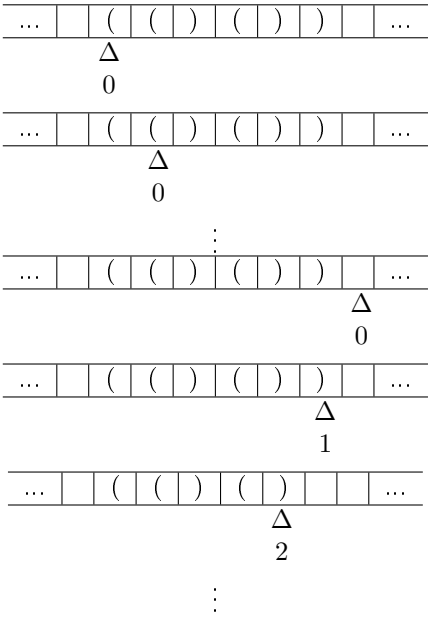
Le nombre 3 n'est donc pas divisible par 4 et a pour reste 3. Il est bien sûr possible de tester cette machine pour tous les nombres que l'on désire : il suffit de les écrire en binaire sur le ruban et de placer la tête de lecture sur le premier chiffre à gauche.

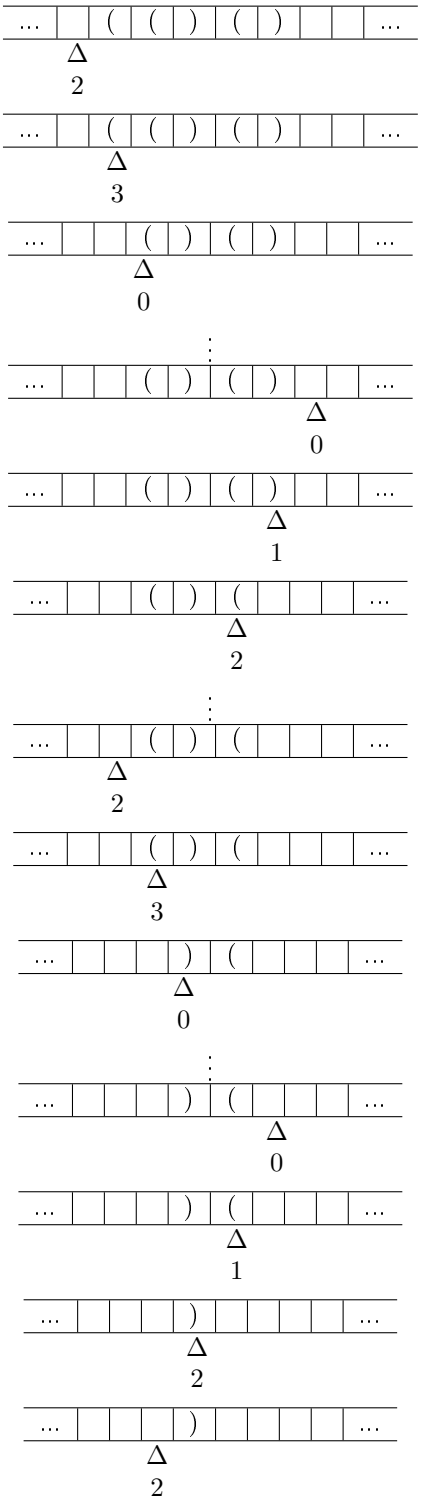
Exemple 7. Voici un exemple de machine de Turing qui décide le langage des parenthèses bien formées, qui n'est pas rationnel et donc non décidable par automate.

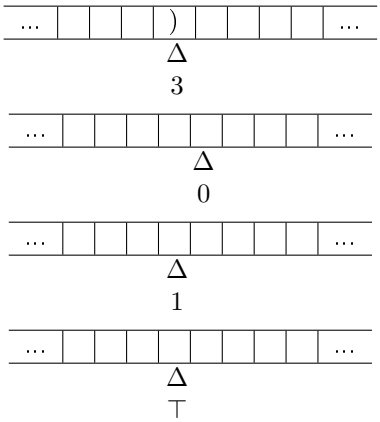
	(	)	□
→0	(,+,0	),+,0	□,-,1
1	(,□,⊥	□,-,2	⊤
2	(,-,2	),-,2	□,+,3
3	□,+,0	),□,⊤	⊥

FIG. 30 – La machine

Voici le déroulement des étapes effectuées par la machine sur un exemple.




… (( ) ( ) … Δ 2
… (( ) ( ) … Δ 3
… (( ) ( ) … Δ 0
… (( ) ( ) … Δ 0
… (( ) ( ) … Δ 1
… (( ) (… Δ 2
… (( ) (… Δ 2
… (( ) (… Δ 3
…) (… Δ 0
…) (… Δ 0
…) (… Δ 1
…) … Δ 2
…) … Δ 2



On peut aussi traiter, de la même manière que précédemment, un exemple qui termine sur l'état faux, comme le mot)(ou le mot ()).
Ces machines à un ruban sont donc déjà beaucoup plus puissantes que les automates, mais le calcul est très long à cause des allers et retours récurrents. Nous allons donc nous intéresser à une machine de Turing, munie de deux rubans, qui nous permettra principalement de faire des comparaisons rapides entre deux mots ou de rechercher un motif.

5.3 Les machines à deux rubans

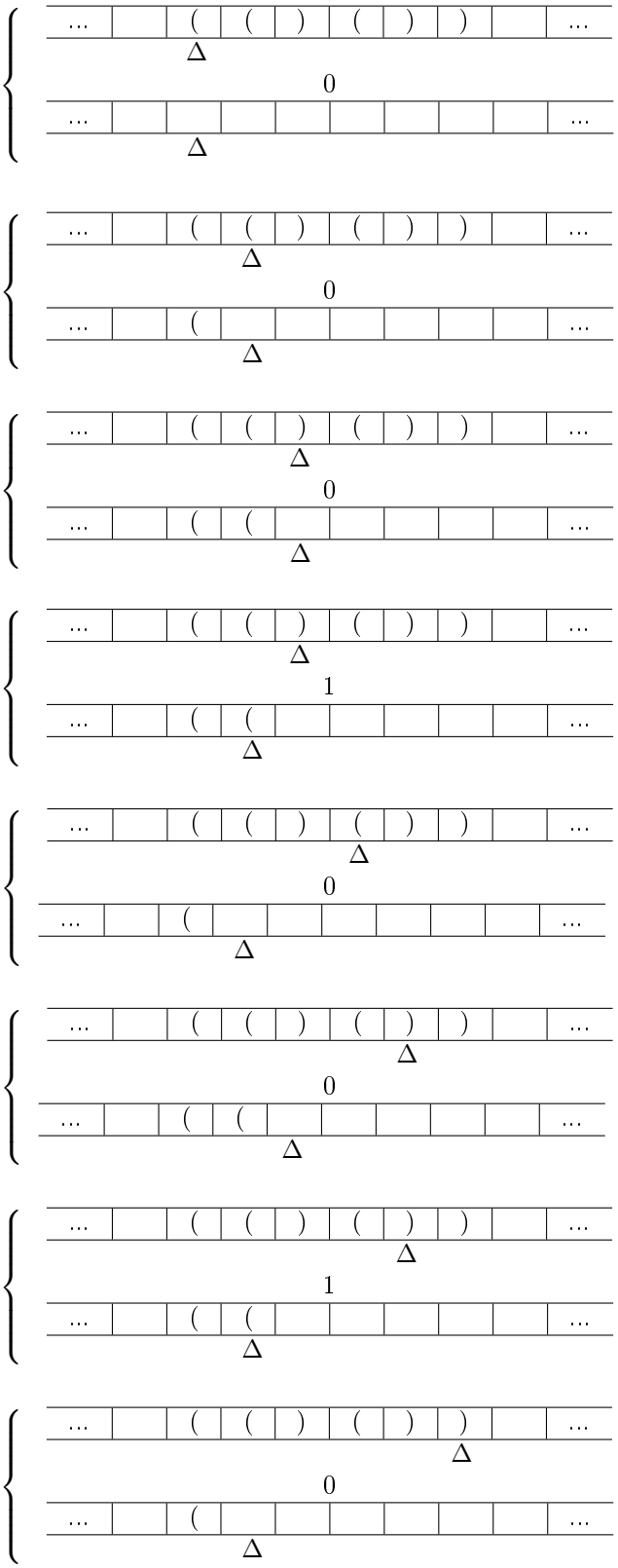
Ces machines fonctionnent exactement comme les machines à un ruban. Par contre, elles possèdent deux têtes de lecture qui se déplacent indépendamment l'une de l'autre, deux fonctions d'impression et deux fonctions de déplacement, qui peuvent être nulles et seront notez 0 dans ce cas. Attention, ces machines n'ont qu'une fonction de changement d'état.

Reprenons l'exemple du langage des parenthèses bien formées et appliquons une machine de Turing à deux rubans au mot (()()). Le tableau représentant les différentes configurations de la machine contiendra maintenant cinq variables par case : le résultat de chacune des deux fonctions d'impression, placés l'un au-dessus de l'autre, le résultat de chacune des deux fonctions de déplacement, positionnés de la même manière que précédemment et le résultat de la fonction de changement d'état qui sera placé à la fin.

Exemple 8.

	$\begin{matrix}(\\ \square\end{matrix}$	$\begin{matrix})\\ \square\end{matrix}$	$\begin{matrix}\square\\ \square\end{matrix}$	$\begin{matrix})\\ (\end{matrix}$	$\begin{matrix}(\\ \square\end{matrix}$	$\begin{matrix})\\ \square\end{matrix}$
→0	$\begin{matrix}(,+\\ (,+\end{matrix}\ 0$	$\begin{matrix}),0\\ ),-\end{matrix}\ 1$	$\begin{matrix}\square,0\\ \square,-\end{matrix}\ 2$	$\begin{matrix}(,+\\ (,+\end{matrix}\ 0$		
1			$\perp$	$\begin{matrix}),+\\ \square,\ 0\end{matrix}\ 0$		
2			$\top$		$\perp$	$\perp$

FIG. 31 – Le tableau de changement de configuration


… (( ) ( )) …
Δ
0
… …
Δ
… (( ) ( )) …
Δ
0
… (…
Δ
… (( ) ( )) …
Δ
0
… ((…
Δ
… (( ) ( )) …
Δ
1
… ((…
Δ
… (( ) ( )) …
Δ
0
… (…
Δ
… (( ) ( )) …
Δ
0
… ((…
Δ
… (( ) ( )) …
Δ
1
… ((…
Δ
… (( ) ( )) …
Δ
0
… (…
Δ

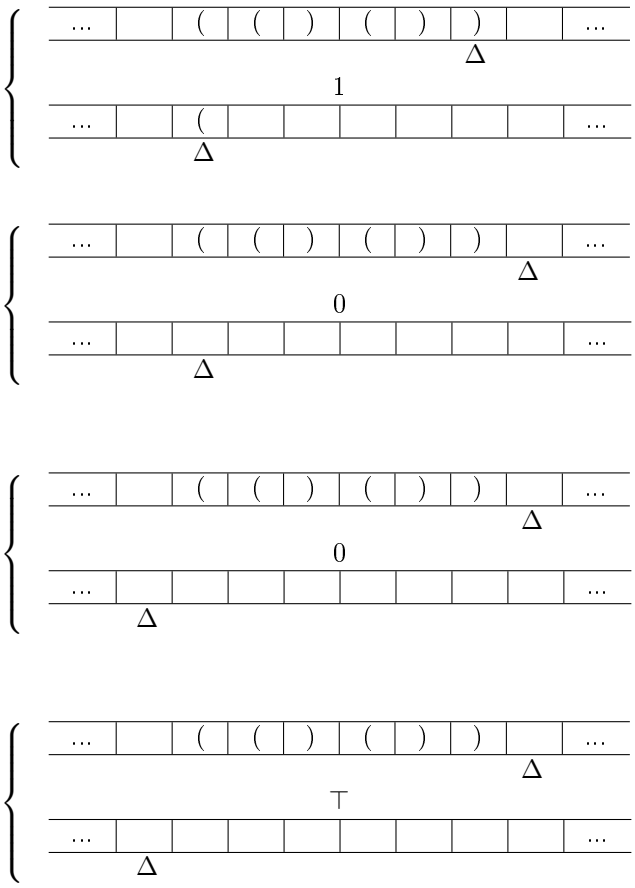


FIG. 32 – Succession des différentes configurations

Après cet exemple, on peut se demander quel lien il existe entre les machines à un et à deux rubans. Nous allons voir qu'en fait, ces machines sont équivalentes.

Théorème 5.1. *Toute machine à un ruban peut s'écrire avec une machine à deux rubans.*

En effet, il est toujours possible de rajouter un ruban sous le premier déjà existant, et de ne pas en tenir compte lors du changement d'état. Passer d'une machine à deux rubans à une machine à un ruban est un peu plus compliqué mais on y parvient tout de même.

Théorème 5.2. *Toute machine de Turing à deux rubans peut s'écrire à l'aide d'une machine à un ruban.*

Nous allons décrire une méthode qui nous permet de construire une machine à un ruban effectuant le même calcul qu'une machine à deux rubans. Il existe plusieurs variantes de cette méthode, nous en avons choisi une qui consiste à numéroter les cases du ruban et à diviser le ruban en parties égales de trois cases. La première case permet de savoir si la tête de lecture est présente sur l'un ou l'autre des deux rubans. La deuxième case contient la lettre stockée sur le premier ruban et la troisième, la lettre stockée sur le deuxième ruban. Ainsi, les cases s'écrivant 3n contiennent un vide s'il n'y a pas de tête de lecture pointant sur la case du ruban du haut ou du bas. Dans le cas contraire, elles contiennent une lettre particulière ($\emptyset$). Les cases s'écrivant $3n+1$ contiennent, dans le même ordre que sur la machine à deux rubans, les lettres du premier ruban. Les cases en $3n+2$ contiennent les lettres du deuxième ruban. Il y a donc, par rapport à la machine à deux rubans, des états supplémentaires pour chercher les deux têtes de lecture, puis agir en fonction. Il existe aussi des états qui correspondent à la machine à deux rubans, ces états sont dits *principaux*, et chaque retour à un état principal indique la fin d'un pas de calcul. Nous noterons $\triangle$ la tête de lecture du premier ruban, ∇ celle du deuxième ruban, $\diamondsuit$ lorsque les deux têtes de lecture se trouvent sur la même case et $\emptyset$ les cases qui ne contiennent rien. Ainsi, la machine venant d'être construite agit comme suit :

i) Recherche de la première tête de lecture, suivie de la lecture de la lettre contenue dans la case suivante. Puis retour au début du ruban, suivi de la recherche de la seconde tête de lecture, puis lecture de la deuxième lettre, contenue dans la case suivante.

ii) Modification de la dernière lettre lue en fonction des instructions données par la machine à deux rubans, puis, retour en arrière afin de remplacer la tête de lecture, que nous avions laissée en place, par $\emptyset$ si l'on avait ∇ ou par $\triangle$ si l'on avait $\diamondsuit$. Si la seconde tête de lecture se déplace vers la droite, on se déplace de trois cases vers la droite et on écrit ∇ si la case contient $\emptyset$. Sinon, on écrit $\diamondsuit$. Si le déplacement se fait sur la gauche, on se déplace de trois cases vers la gauche et on écrit ∇ si la case contient $\emptyset$. Sinon, on écrit $\diamondsuit$. Puis la machine retourne au début du ruban et recherche la première tête de lecture $\triangle$ ou $\diamondsuit$; on remplace ensuite la lettre contenue dans la case suivante par la valeur, donnée par la première machine à deux rubans, lorsqu'il y a lieu. Après cela, il nous faut procéder au déplacement de la première tête de lecture exactement de la même manière que décrit précédemment.

iii) Changement d'état principal.

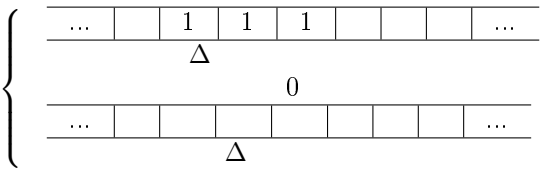
Pour fixer les idées, nous allons dérouler la méthode sur un exemple.
Nous montrerons les étapes de calcul pour passer de l'état principal 0 à l'état principal 1.
Pour cela,nous allons construire une machine qui permet de reconnaître si un mot sur l'alphabet $\{1\}$ est composé de 2^n 1.
Une des solutions consiste à construire une machine à deux rubans. Le premier ruban contiendra le mot à tester et sera parcouru une seule fois sans retour en arrière. Le deuxième contiendra le nombre 1 déjà lu par la machine, nombre qui va se modifier au cours du calcul de la machine.

Avec les états q_0 et q_1 nous calculons le nombre de 1 lus sur le premier ruban et nous avançons sur ce dernier. L'état q_2, permet de faire l'opération "plus

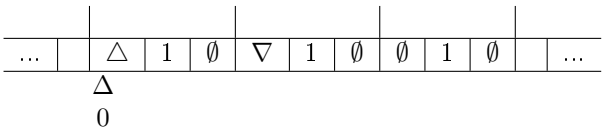
	$\begin{matrix}1\\\square\end{matrix}$	$\begin{matrix}1\\0\end{matrix}$	$\begin{matrix}1\\1\end{matrix}$	$\begin{matrix}\square\\\square\end{matrix}$	$\begin{matrix}\square\\1\end{matrix}$	$\begin{matrix}\square\\0\end{matrix}$
$\rightarrow 0$	$\begin{matrix}1\ ,\ +\\1\ ,\ +\end{matrix}1$	$\begin{matrix}1\ ,\ +\\1\ ,\ \square\end{matrix}2$		$\perp$	$\begin{matrix}\square\ ,\ \square\\1\ ,\ \text{-}\end{matrix}3$	$\begin{matrix}\square\ ,\ \square\\0\ ,\ \text{-}\end{matrix}0$
1	$\begin{matrix}1\ ,\ +\\0\ ,\ \square\end{matrix}0$			$\perp$		
2	$\begin{matrix}1\ ,\ \square\\1\ ,\ +\end{matrix}3$	$\begin{matrix}1\ ,\ \square\\1\ ,\ +\end{matrix}3$	$\begin{matrix}1\ ,\ \square\\0\ ,\ \text{-}\end{matrix}2$		$\perp$	
3	$\begin{matrix}1\ ,\ +\\\square\ ,\ \text{-}\end{matrix}0$	$\begin{matrix}1\ ,\ \square\\0\ ,\ +\end{matrix}3$	$\begin{matrix}1\ ,\ \square\\1\ ,\ +\end{matrix}3$	$\top$	$\perp$	$\perp$

FIG. 33 – Tableau de transitions

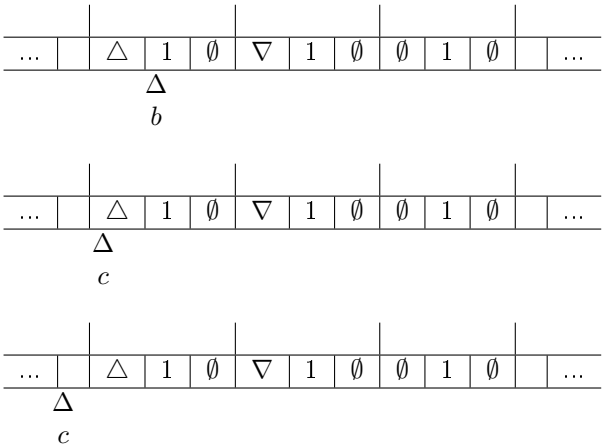
1" avec des retenues : lors de ce calcul, le premier ruban est ignoré. L'état q_3, permet de revenir à la fin du nombre binaire du deuxième ruban de manière à pouvoir recommencer l'opération "plus 1" sur le mot de ce dernier.
Déroulement sur le mot 111
La machine à deux rubans à l'état 0 :

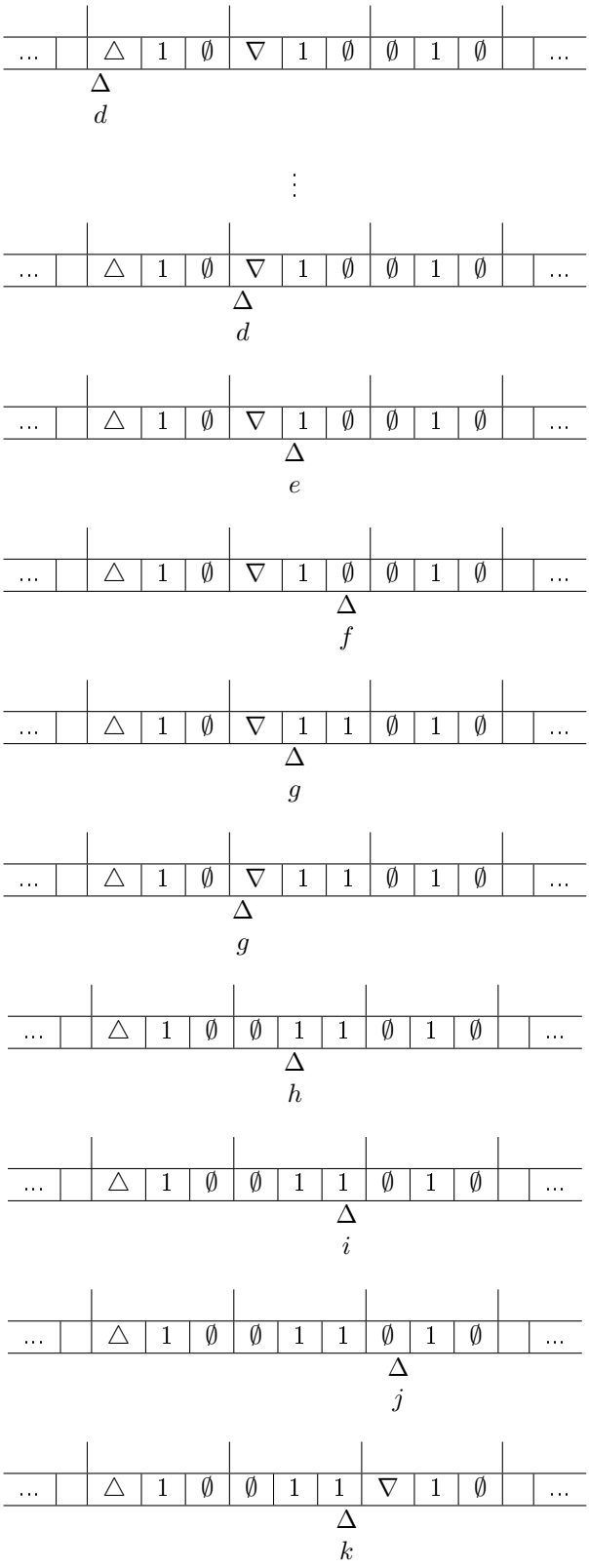


La machine à un ruban correspondant à l'état 0 :

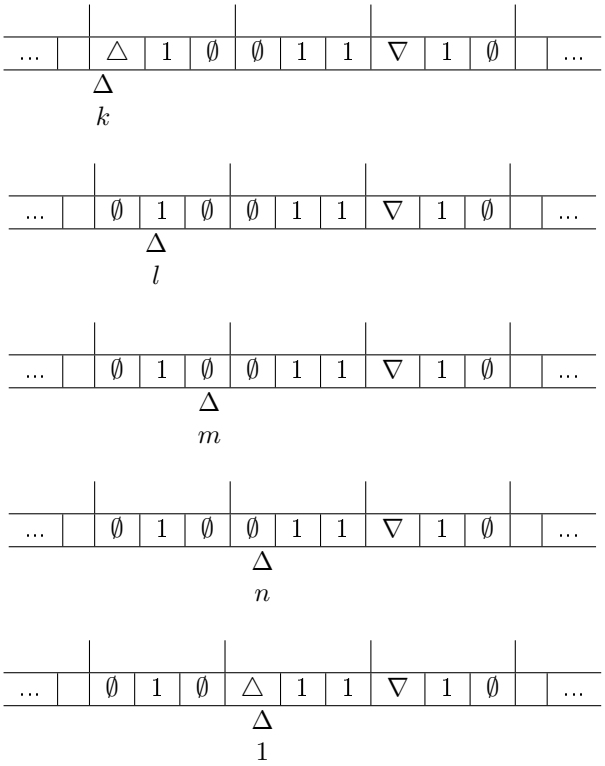


Voici maintenant les différentes configurations de la machine à un ruban avant de se retrouver à l'état principal 1. Les états intermédiaires seront nommés avec les lettres a, b, c, ...




... △ 1 ∅ ∇ 1 ∅ ∅ 1 ∅ ...
Δ
d
⋮
... △ 1 ∅ ∇ 1 ∅ ∅ 1 ∅ ...
Δ
d
... △ 1 ∅ ∇ 1 ∅ ∅ 1 ∅ ...
Δ
e
... △ 1 ∅ ∇ 1 ∅ ∅ 1 ∅ ...
Δ
f
... △ 1 ∅ ∇ 1 1 ∅ 1 ∅ ...
Δ
g
... △ 1 ∅ ∇ 1 1 ∅ 1 ∅ ...
Δ
g
... △ 1 ∅ ∅ 1 1 ∅ 1 ∅ ...
Δ
h
... △ 1 ∅ ∅ 1 1 ∅ 1 ∅ ...
Δ
i
... △ 1 ∅ ∅ 1 1 ∅ 1 ∅ ...
Δ
j
... △ 1 ∅ ∅ 1 1 ∇ 1 ∅ ...
Δ
k

⋮



La machine qui vient d'être décrite, fonctionne, mais n'est pas très performante. Nous allons envisager une solution sur une machine à un ruban, beaucoup plus efficace que celle décrite précédemment. Cette solution consiste à effacer un 1 au début du ruban, puis un 1 à la fin. Si on ne peut plus effacer de 1 au début, c'est que le mot est bien de la forme voulue, si on ne peut plus effacer de 1 à la fin, c'est que le mot n'est pas composé de 2^n fois la lettre 1. Cela donne le tableau de transitions suivant :

	1	□
→0	□,+,1	⊤
1	1,+,1	□,-,2
2	□,-,3	⊥
3	1,-,3	□,+,0

FIG. 34 – Tableau de transitions d'une machine de Turing à un ruban reconnaissant les mots formés de 2^n fois la lettre 1.

L'état q_0 permet d'effacer la première lettre lorsque cela est possible. L'état q_1 permet d'aller à la fin du mot et est suivi de l'état q_2 qui efface la dernière lettre du mot lorsque cela est possible. L'état q_3 retourne au début du mot et est immédiatement suivi de l'état q_0.

5.4 Les machines de Turing à un nombre quelconque de rubans

Nous venons de voir qu'une machine de Turing à deux rubans est équivalente à une machine à un ruban. On peut aisément comprendre que toute machine à un nombre quelconque de rubans est équivalente à une machine à un seul ruban. En effet, lorsque nous avons une machine à n rubans, il suffit de considérer deux rubans de la machine et de les réduire à un seul ruban, comme nous l'avons vu pour les machines à deux rubans. Nous obtenons donc une machine à $n-1$ rubans. Il suffit de recommencer l'opération jusqu'à l'avoir pratiquée $n-1$ fois. A ce moment là, nous avons obtenu une machine à un ruban.

Maintenant que nous connaissons l'équivalence qui existe entre les machines à un ruban et à plusieurs rubans, il nous est permis d'utiliser indifféremment l'une ou l'autre des approches pour différents calculs. Nous allons voir différents exemples qui illustrent la fonctionnalité des machines de Turing.

5.5 Présentation de différentes machines de Turing

5.5.1 L'addition de deux nombres entiers

Nous allons construire une machine qui additionne deux entiers positifs ou négatifs écrits sous forme binaire. Nous noterons un entier de la manière suivante : nous ferons précéder d'un 1 ou d'un 0 l'entier considéré en écriture binaire selon qu'il est négatif ou positif. Pour simplifier les choses, nous construirons une machine qui trouve le signe du résultat de l'addition des deux entiers. Elle écrira ce signe sur le premier ruban après l'entier codé, et placera un blanc entre l'entier codé et le signe. Cette machine s'arrêtera à l'état vrai si les deux entiers sont de même signe, à l'état faux dans le cas contraire. Nous constuirons ensuite, une machine qui additionne et une machine qui soustrait. Pour faciliter le travail, nous présenterons uniquement une solution avec deux rubans .

Construisons la machine qui reconnaît le signe du résultat. Elle fonctionne de la manière suivante :

i) lecture de la première lettre de chacun des deux mots, puis, si on lit 1 et 1, alors on se place dans l'état q_1, si on rencontre deux 0, on se place à l'état q_2, si on voit 1 et 0, on se place à l'état q_3 et si on voit 0 et 1, on se place à l'état q_4. Dans tous les cas, les lettres qui sont lues sont effacées et les têtes de lecture des deux rubans avancent d'une case chacune.

ii) A partir des états q_1 et q_2, les actions qui se font, sont les mêmes : on avance jusqu'à trouver un blanc sur le premier ruban, on avance encore d'une case, on écrit : un 1 dans le cas q_1 et un 0 dans le cas q_2.

iii) A partir des états q_3 et q_4 les actions sont aussi identiques : le but est de reconnaître l'entier le plus grand des deux en valeur absolue. Le signe du plus grand des deux sera retenu. Pour cela, il faut comparer les lettres une à une, de gauche à droite. La première fois que la machine rencontre deux lettres différentes, elle se place dans un état q_5 si on a lu un 1 sur le premier ruban. Cet état suppose que le premier mot est le plus grand. Sinon, la machine se place dans un état q_6, ce qui suppose que le mot du

deuxième ruban est le plus grand.

Arrivé à l'un de ces deux derniers états, il suffit de lire les lettres restantes sur chacun des deux rubans sans les modifier. Dans le même temps, nous vérifions, dans le premier cas, que le premier mot n'est pas moins long que le deuxième. Dans le second cas, nous contrôlons que le mot du deuxième ruban n'est pas moins long que le mot du premier. Si pour chacun des cas la condition est respectée alors, on inscrira après l'entier du premier ruban, un blanc, puis un 1 si l'on était dans l'état q_5 précédé de l'état q_3, ou dans l'état q_6 précédé de l'état q_4. Dans les cas contraires, on écrira un 0.

iv) La machine place les têtes de lecture au début des deux entiers et s'arrête à l'état vrai si elle est passée par l'état q_1 ou q_2. Sinon, la machine s'arrête à l'état faux.

Nous allons maintenant décrire, par un tableau de changements d'états, la machine qui additionne deux entiers naturels et qui s'arrête à l'état vrai lorsqu'elle a fini le calcul.
L'état q_0 permet d'aller à la fin des deux entiers, le premier état permet d'additionner sans retenue, le deuxième fait la même chose avec retenue.

	$\square$ $\square$	1 1	0 0	1 0	1 0	$\square$ 1	$\square$ 0	1 $\square$	0 $\square$
$\rightarrow 0$	$\square$, - 1 $\square$, -	1 , + 0 1 , +	0 , + 0 0 , +	1 , + 0 0 , +	0 , + 0 1 , +	$\square$, 0 0 1 , +	$\square$, 0 0 0 , +	1 , + 0 $\square$, 0	0 , + 0 $\square$, 0
1	$\square$, - $\top$ $\square$, -	0 , - 2 $\square$, -	0 , - 1 $\square$, -	1 , - 1 $\square$, -	1 , - 1 $\square$, -	1 , - 1 $\square$, -	0 , - 1 $\square$, -	1 , - 1 $\square$, 0	0 , - 1 $\square$, 0
2	1 , - 0 $\square$, 0	1 , - 1 $\square$, -	1 , - 0 $\square$, -	0 , - 1 $\square$, -	0 , - 1 $\square$, -	0 , - 1 $\square$, -	1 , - 1 $\square$, -	0 , - 1 $\square$, 0	1 , - 1 $\square$, 0

Nous allons enfin construire une machine qui effectue la soustraction de deux entiers naturels, donnés dans le bon ordre (l'entier le plus grand est donné au premier ruban) et se place dans l'état vrai quand elle a fini le calcul :

	$\square$ $\square$	1 1	0 0	1 0	1 0	1 $\square$	0 $\square$
$\rightarrow 0$	$\square$, - 1 $\square$, -	1 , + 0 1 , +	0 , + 0 0 , +	1 , + 0 0 , +	0 , + 0 1 , +	1 , + 0 $\square$, 0	0 , + 0 $\square$, 0
1	$\square$, - $\top$ $\square$, -	0 , - 1 $\square$, -	0 , - 1 $\square$, -	1 , - 1 $\square$, -	1 , - 2 $\square$, -	1 , - 1 $\square$, 0	0 , - 1 $\square$, 0
2		1 , - 2 $\square$, -	1 , - 1 $\square$, -	0 , - 1 $\square$, -	0 , - 2 $\square$, -	0 , - 1 $\square$, 0	1 , - 1 $\square$, 0

Ensuite, pour construire une machine qui additionne deux entiers positifs ou négatifs, il suffit d'assimiler l'état final vrai, de la machine qui calcule le signe du résultat, avec l'état initial de la machine qui additionne deux entiers naturels. Pour l'état final faux, quelques modifications s'imposent : il faut faire en sorte d'échanger les deux rubans si l'entier le plus grand (sans son signe) se trouve sur le deuxième ruban, puis d'écrire le signe du résultat comme expliqué plus haut.

Ensuite, il suffit de passer à l'état initial de la machine à soustraire deux entiers naturels. Il ne nous reste plus qu'à récupérer, à la fin du premier ruban, le signe du résultat, à l'effacer et à le placer juste avant l'entier écrit sur ce ruban.

5.5.2 Reconnaissance des mots de la forme $w * w$

Une machine de Turing peut permettre de reconnaître si un mot est de la forme $w * w$ sur l'alphabet 0,1.
Nous donnerons ici une solution avec une machine à deux rubans et une autre avec un seul ruban.
La première solution consiste à recopier la première partie du mot w sur un deuxième ruban vide au départ, puis à ignorer le signe $*$ sur le premier ruban, et à retourner au début du mot du deuxième ruban sans toucher au premier.
Ces quelques opérations faites, il n'y a plus qu'à parcourir en parallèle le premier et le deuxième ruban, en vérifiant que nous sommes bien dans le cas où on lit deux 1 ou deux 0. Si ce n'est pas le cas, on s'arrête et on se place dans l'état faux. Sinon, on continue jusqu'à rencontrer deux blancs, cela signifie que c'est la fin du mot et dans ce cas, on se place dans l'état vrai.

La deuxième solution consiste à effacer la première lettre et à se placer dans l'état : j'ai lu la lettre 0 ou j'ai lu la lettre 1. Puis on parcourt le mot sans rien changer, jusqu'à rencontrer la lettre $*$. Dans ce cas, on change d'état et on lit la lettre suivante : si les deux lettres sont différentes, on s'arrête et on se place dans l'état faux. Si c'est la même lettre que nous avons effacée au début, on la supprime, et on retourne au début du mot pour recommencer la manoeuvre, jusqu'à ne plus pouvoir effacer de 0 ou de 1. Dans ce cas, on vérifie qu'après la lettre $*$ il n'y a plus d'autre lettre. Si cela se vérifie, la machine se place dans l'état vrai et s'arrête.

5.5.3 Une machine décidant si tous les mots qu'on lui fournit sont tous différents

Cet exemple montre qu'une machine de Turing peut décider si oui ou non des mots $w_1\ w_2\ w_3\ \ldots\ w_n$ (avec $n \geq 1$ entier naturel) sont tous différents. Ici encore, nous présenterons deux solutions.
Voici la première avec trois rubans.
L'idée principale est toujours de parcourir une seule fois le premier ruban, qui contient tous les mots les uns à la suite des autres séparés par des blancs. Sur le deuxième ruban, on écrit tous les mots sauf le premier que l'on écrit sur le troisième ruban : ce mot va être comparé à tous les autres mots. On compare le premier mot du deuxième ruban avec le mot du troisième. Dès que l'on trouve une lettre différente, on revient au début du mot du troisième ruban et on finit de lire le mot du deuxième sans rien changer. Puis on se positionne au début du mot suivant. On compare de nouveau les deux derniers rubans jusqu'à rencontrer un blanc et on recommence. Nous procédons de cette manière jusqu'à ce qu'il n'y ait plus de mots à lire sur le deuxième ruban. Dans ce cas, on efface le troisième ruban et on lit le mot suivant sur le premier ruban, tout en l'effaçant sur le deuxième (c'est le premier mot de ce dernier). Dans le même temps, on le recopie sur le troisième ruban : c'est ce mot qui va maintenant être comparé à tous les autres sauf au premier mot car cela a déjà été fait. On se place au

début du premier mot du ruban deux et du ruban trois et on recommence la manoeuvre jusqu'à trouver deux mots semblables (et dans ce cas la machine se positionne à l'état faux), ou jusqu'à ce qu'il n'y ait plus de mots à lire sur le premier ruban (et alors la machine s'arrête à l'état vrai).

Voici la deuxième solution, appliquée sur un ruban. Pour arriver au résultat, la machine va utiliser quatre lettres n'appartenant pas à l'alphabet $\{0, 1\}$.
Le principe est simple : on commence par comparer les deux premiers nombres comme dans l'exemple 2 (deuxième méthode mais à la place d'une * on a un blanc et au lieu d'effacer les lettres, on les transforme en υ si c'est un 1 et en δ si c'est un 0). Si la machine arrive simultanément à la fin des deux mots, c'est qu'ils sont identiques et elle s'arrête sur l'état faux. Si elle rencontre deux lettres différentes, la machine finit de lire le deuxième mot en remplaçant tous les 1 par υ et tous les 0 par δ. Arrivée à la fin, elle remonte le mot en changeant cette fois-ci tous les υ en Υ et tous δ en Δ jusqu'au prochain blanc. Dans ce cas de figure, elle remplace tous les δ par 0 et tous les υ par 1. En voyant un blanc à nouveau elle revient au début du mot et compare la première lettre qui est un 1 ou un 0, la transforme comme indiqué plus haut, lit le reste du mot sans rien changer puis compare le chiffre transformé avec le prochain chiffre rencontré (si la machine lit δ, υ, Δ ou Υ, elle continue de parcourir le ruban sans rien changer).
Si la machine rencontre deux blancs d'affilée, c'est qu'elle a déjà comparé le premier mot à tous les autres. A ce moment là, elle lit le ruban de droite à gauche et transforme les Δ en 0, les Υ en 1, les 1 et les υ en υ, les 0 et les δ en δ. Après toutes ces transformations, la machine lit le ruban de gauche à droite et transforme le premier 1 (resp. 0) en υ (resp. δ) qu'elle rencontre et recommence le processus au début.
Si, lorsque la machine parcourt le ruban de gauche à droite à la recherche du premier chiffre, elle lit deux blancs d'affilée, c'est qu'elle a comparé tous les nombres et alors elle s'arrête à l'état vrai.

5.6 Réduction de l'alphabet

Jusqu'à présent, nous avons vu le côté pratique de rajouter des lettres additonnelles à l'alphabet de départ, pour résoudre certains problèmes. Nous allons voir qu'il est tout à fait possible de se restreindre à l'alphabet {0,1}, ce qui se rapproche de la pratique : le plus souvent, les machines telles que les ordinateurs ne permettent qu'un alphabet à deux lettres.
Pour se restreindre à l'alphabet {0,1}, à partir d'un alphabet Σ de n lettres (symbole blanc compris), il suffit d'écrire n sous forme binaire et de compter le nombre m de chiffres (0 ou 1) dont il est composé. Cela nous indique le nombre de chiffres que l'on doit utiliser pour coder chacune des lettres de l'alphabet Σ. Pour faciliter les choses, nous conviendrons de toujours coder le symbole blanc par

$$\underbrace{0 \ldots 0}_{m}$$

.
Ainsi, chaque mot de Σ^* de longueur l pourra être codé par un mot de $\{0,1\}^*$, de longueur lm.
Pour simuler une machine de Turing $\mathbf{M}$ faisant un calcul sur Σ, il suffit de

construire une autre machine de Turing $\mathbf{M}'$, sur l'alphabet {0,1} dont les états principaux sont ceux de $\mathbf{M}$ et les autres états permettent de lire chaque lettre e de l'alphabet Σ codée sur {0,1}. Il est évident qu'à chaque état principal, la tête de lecture est placée sur la première case des m cases possibles.

Ainsi, cette nouvelle machine fonctionne en trois étapes :

1. Lecture de la case désignée par la tête de lecture, puis de celle qui suit, puis on continue jusqu'à avoir pratiqué l'opération m fois, ce qui entraîne la reconnaissance du symbole de Σ correspondant.

2. Modification des symboles lus et de la position de la tête de lecture.

3. Changement d'état principal.

Ainsi, avec toutes les opérations et tous les exemples différents que nous avons étudiés jusqu'à présent, on peut penser qu'il est possible de calculer tout ce que l'on souhaite avec une machine de Turing, du moment que l'on peut décrire ce calcul mécaniquement. Ceci nous amène à la thèse de Church d'après laquelle toute opération pouvant être décrite par un calcul mécanique est réalisable par machine de Turing. On peut aussi décrire cette thèse avec l'équivalence suivante : calculable effectivement $\Longleftrightarrow$ calculable par machine de Turing. Cette thèse ne peut être démontrée car la calculabilité par machine de Turing est une notion mathématique ce qui n'est pas le cas de la calculabilité effective. Cependant, nous pouvons réunir des arguments en faveur de cette thèse comme les équivalences que nous pouvons établir entre la calculabilité par machines de Turing et la calculabilité entre les machines à accès direct, que nous allons présenter dans le paragraphe suivant, ou la calculabilité par un algorithme implémentable dans un langage de programmation quelconque tel que C^{++} ou Scheme ou Java ....

6 Les machines à accès direct

La principale différence avec les machines de Turing réside dans le fait que les machines à accés direct nous permettent d'avoir une mémoire à accès direct et non plus à accès séquentiel comme avec les machines de Turing. Cela facilite grandement les calculs. Ce nouveau modèle de calcul se rapproche dans sa conception des calculateurs réels et des calculs pouvant être décrits par des algorithmes, implémentables dans des langages de programmation tels que C^{++}, Scheme, Java ....

6.1 Définition

Une *machine à accés direct* est composée

i) d'un *périphérique d'entrée*, à accès séquentiel, contenant une liste finie d'entiers positifs ou négatifs ;

ii) d'un nombre (potentiellement) infini de *registres* numérotés $r_0, r_1, \ldots, r_n, \ldots$ et contenant des entiers positifs ou négatifs. Le registre r_0 joue un rôle particulier et on lui donne le nom d'*accumulateur* ;

iii) d'un *périphérique de sortie* à accès séquentiel, contenant des entiers ;

iv) d'un compteur de programme, contenant un entier naturel et qui commence à 0 et augmente de 1 à chaque fin d'instruction, sauf lorsque l'instruction porte précisément sur l'agissement du compteur ;

v) d'un programme dont le rôle est de modifier la configuration de la machine suivant des instructions que nous définirons un peu plus loin.

Comme pour les machines de Turing, cette machine peut prendre diverses configurations.
La configuration initiale est telle que :

i) la tête de lecture du périphérique d'entrée est placée sur le premier élément de la liste ;

ii) tous les registres sont vides ;

iii) le périphérique de sortie contient la liste vide ;

iv) le compteur de programme est à 0.

6.2 Fonctionnement

Le programme d'une machine à accès direct est composé d'une suite finie d'instructions numérotées et chaque pas de calcul consiste à exécuter l'instruction dont le numéro est indiqué par le compteur de programme. Les instructions, qui sont très simples, se répartissent en quatre groupes :

1. les instructions d'entrée/sortie,
2. les instructions de transfert,
3. les opérations arithmétiques,
4. les instructions de contrôle.

Les instructions des trois premiers groupes agissent sur la configuration de la machine et le compteur de programme est augmenté de 1, alors que celles du dernier groupe ont une action sur le compteur de programme.
Voici la liste des instructions disponibles :

6.3 Instructions d'entrée/sortie

READ : permet de transférer l'entier du périphérique d'entrée pointé par la tête de lecture dans l'accumulateur (ce qui efface le contenu qui s'y trouvait) et avance la tête de lecture à l'entier suivant.

WRITE : permet d'ajouter le contenu de l'accumulateur sur le périphérique de sortie sans rien changer au contenu de l'accumulateur.

6.4 Instructions de transfert

LOAD : cette instruction s'applique à une valeur fournie par le programme et elle provoque le chargement de cette variable dans l'accumulateur. Il est possible de désigner la valeur à charger de trois manières différentes :

i) en *mode immédiat* LOAD $= i$. i est la donnée sous forme numérique,

ii) en *mode direct* LOAD i. i désigne, cette fois-ci, le registre contenant la valeur à charger.

iii) *mode indirect* LOAD ↑i. i désigne la valeur du registre qui contient l'indice du registre contenant la valeur à charger.

STORE : permet le transfert du contenu de l'accumulateur dans le i-ème registre (STORE i) ou dans le registre dont l'indice se trouve dans le registre i (STORE ↑i).

6.5 Opérations arithmétiques

ADD : ajoute, au contenu de l'accumulateur, la variable transmise soit en mode immédiat, soit en mode direct, soit en mode indirect.

MULT : multiplie le contenu de l'accumulateur par la variable transmise selon l'un des trois modes décrits plus haut.

DIV = 2 : remplace le contenu de l'accumulateur par son quotient entier dans la division par deux.

6.6 Instructions de contrôle

START : avance le compteur de programme à un.

GOTO n : donne la valeur n au compteur de programme.

IF$\geq$0 THEN n : donne la valeur n au compteur de programme si le contenu de l'accumulateur est positif ou nul. Dans le cas contraire, le compteur de programme est incrémenté de un.

STOP : arrête l'exécution du programme.

Nous aurions pu définir d'autres opérations telles que la soustraction ou la division, mais cela n'est pas nécessaire : avec les instructions dont on dispose, il est tout à fait possible d'écrire un programme qui calcule la soustraction de deux nombres ou leur division entière.
Nous allons maintenant écrire un petit programme qui permet à une machine à accès séquentiel de reconnaître si un entier est premier ou non. Cette machine écrira 0 si le nombre est premier, 1 sinon. En C^{++}, le programme s'écrit de la manière suivante :

```
int Entier Premier(int n)
{int i;
for(i=2; i<=n-1; i++)//on vérifie que i n'est pas plus grand ou égal à n
{
    if (n%i=0)
{return 1;//Si n est divisble
}
}
{
return 0;
}
}
```

Nous allons maintenant écrire un programme qui effectue le même calcul, sur une machine séquentielle.

```
0 START
1 READ                (on lit l'entier n sur le périphérique d'entrée)
2 STORE 1             (on met le nombre n dans le registre 1)
3 STORE 4             (ce registre sera utilisé pour calculer n − i ∗ x tant que
                       le résultat n'est pas négatif)
4 ADD =-1
5 STORE 2             (on met n − 1 dans le registre 2)
6 LOAD = 2
7 STORE 3
8 MULT-1
9 ADD 2
10 IF ≥ 0 THEN 14
11 LOAD = 0
12 WRITE
13 STOP
14 LOAD 3
15 MULT -1
16 ADD 4
17 STORE 4
18 IF ≥ 0 THEN 30
19 ADD 3
20 MULT -1
21 IF ≥ 0 THEN 27
22 LOAD 1
23 STORE 4
24 LOAD 3
25 ADD =1
26 GOTO 7
27 LOAD =1
28 WRITE
29 STOP
30 LOAD 3
31 MULT -1
32 ADD 4
33 STORE 4
34 GOTO 17
```

En fait, nous allons voir que toutes les opérations pouvant être effectuées par machines de Turing peuvent aussi l'être par machines à accès direct.

6.7 Simulation d'une machine de Turing par une machine à accès direct

Il existe plusieurs façons de simuler une machine de Turing par une machine à accès direct. Nous supposerons que la machine de Turing est à un ruban (ce qui ne dérange en rien car toute machine de Turing à plusieurs rubans est équivalente à une machine à un ruban, comme nous l'avons vu précédemment), que l'alphabet est $\{0, 1\}$ (ce qui n'apporte en fait aucune restriction non plus). Nous allons présenter une des solutions possibles, qui n'est pas la plus astucieuse. Une autre est présentée dans l'ouvrage de Jacques Stern.

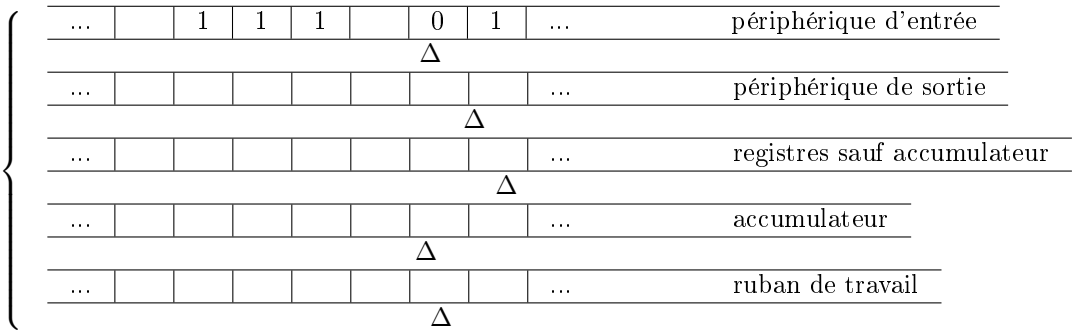
Une des solutions est d'associer un registre pour chacune des cases de la machine de Turing, de stocker dans d'autres registres le nom des états ainsi que celui de l'état courant qui sera modifié au cours du calcul et de stocker le numéro de la case où se trouve la tête de lecture. Une fois les registres remplis, il suffit de lire le registre dont le numéro se trouve dans le registre désignant la tête de lecture. Par exemple, si celle-ci est stockée dans le registre r_4, on accède à la case pointée par la tête de lecture avec la commande LOAD $\uparrow 4$.
Ensuite, on fait ADD = -1, puis IF ≤ 0 THEN, ce qui nous permet de retenir ce qu'il y avait dans l'accumulateur, puis on recherche le numéro de l'état courant grâce à son numéro de registre. On soustrait ensuite chacun des états de la machine de Turing à l'état courant et on regarde si le résultat, ainsi que son opposé, est positif ou nul. Si ce n'est pas le cas, on continue la manoeuvre avec les états qui n'ont pas encore été testés. Sinon, on change l'état courant comme indiqué par la machine de Turing, on modifie la lettre pointée par la tête de lecture en la retrouvant de la même manière que décrit au début. Puis on change la tête de lecture, et on fait un GOTO 1, pour recommencer la boucle.

Ainsi, tout calcul pouvant être fait par machine de Turing peut être fait par machine à accès direct. Nous allons voir que le contraire est vrai aussi.

6.8 Simulation d'une machine à accès direct par une machine de Turing

Tout d'abord, nous allons présenter quelques conventions d'écriture dont nous nous servirons par la suite. Nous noterons un entier de la manière suivante : nous ferons précéder d'un 1 ou d'un 0 l'entier considéré en écriture binaire selon qu'il est négatif ou positif et chaque registre sera écrit comme suit : $*v\mathfrak{C}x*$, l'entier x étant celui contenu dans le registre numéro v. L'entier v étant toujours positif, le signe ne sera pas précisé.

Ces mots sont ensuite placés dans une machine de Turing à cinq rubans comme expliqué sur le schéma ci-dessous.



Nous pouvons facilement comprendre que chacune des instructions d'une machine à accès direct est simulable par la machine de Turing présentée ci-dessus. Nous allons donner l'exemple de l'instruction STORE et de l'instruction ADD.

6.8.1 STORE 2

Cette instruction modifie les registres et donc le ruban trois.

i) écriture de la représentation du registre 2 sur le cinquième ruban : $*11\complement$;
ii) recherche sur le ruban 2 du motif qui se trouve sur le ruban cinq ;
iii) effacement du mot correspondant au registre et à son contenu ;
iv) parcours du ruban jusqu'au signal de fin de bande qui peut être $\natural$ par exemple et effacement de celui-ci, suivi de l'écriture du contenu du cinquième ruban, suivi immédiatemant du symbole $\complement$, puis du contenu du quatrième ruban (l'accumulateur) ;
v) Ajout d'un $\natural$ en fin de bande trois.

6.8.2 ADD 3

On effectue les tâches suivantes :

i) écriture de la représentation du registre 3 sur le cinquième ruban : $*111\complement$;
ii) recherche sur le ruban 3 du motif qui se trouve sur le ruban cinq ;
iii) copie sur le ruban cinq de la donnée contenue dans l'accumulateur, suivie de la donnée qui se trouve derrière le motif que l'on a trouvé en ii). Si le motif n'a pas été trouvé, la donnée est considérée comme nulle ;
iv) Calcul de la somme des deux nombres stockés sur le cinquième ruban, comme nous savons déjà le faire ;
v) effacement du ruban quatre et recopie sur ce dernier de la donnée obtenue sur le ruban cinq

Lorsque toutes les instructions ont été traduites sur des machines de Turing, pour simuler l'ordre du programme, il ne reste plus qu'à identifier l'état final de la première instruction avec l'état initial de la deuxième et ainsi de suite.

7 Conclusion

Ainsi, nous avons montré l'équivalence des machines de Turing avec les machines à accès direct. L'équivalence entre le langage reconnu par les machines à accès direct et les langages de programmation de haut niveau se comprend aisément car, en réalité, la conception de ces programmes s'appuie sur les machines à accès direct. En effet, les instructions simples de ces dernières permettent de construire toutes les instructions un peu plus compliquées des langages de programmation, telles que «tant que», «diviser»... Une fois ces équivalences acquises, on comprend facilement que l'opinion général des mathématiciens soit favorable à la thèse de Church, même s'il nous est impossible de la prouver. Pour la réfuter, il faudrait trouver un calcul qui puisse se décrire par une méthode que les mathématiciens accepteraient de qualifier d'algorithmique (ce dit d'une

méthode constituée d'une suite finie d'opérations élémentaires constituant un schéma de calcul) et qui ne soit pas calculable par machine de Turing, ou par machine à accès direct, puisqu'elles sont équivalentes, ou par n'importe quel langage de programmation utilisé par les informaticiens.

8 Bibliographie

1 Jacques Stern, *Fondements mathématiques de l'informatique*, McGraw-Hill, 1990.

2 Patrick Dehornoy, *Mathématiques de l'informatique*, Dunod, 2000.

3 Alan Turing, Jean-Yves Girard, *La machine de Turing*, Seuil, 1999.