Chapitre 4

Calculabilité

4.1 Machines de Turing

Un alphabet est un ensemble de symboles. Le symbole blanc est dénoté par $\square$. Si Σ est un alphabet, alors $\Sigma_\square = \Sigma \cup \{\square\}$ dénotera l'alphabet Σ augmenté par le symbole blanc.

Définition 4.1. Une *machine de Turing* est un triplet $T = \langle \Sigma, Q, \Delta \rangle$, où
- Σ est l'*alphabet d'entrée*, tel que $\square \notin \Sigma$;
- Q est un ensemble fini d'*états*, avec deux états distingués $q_0, q_f \in Q$;
- $\Delta : Q \times \Sigma_\square \to Q \times \Sigma_\square \times \{-1, +1\}$ est la *fonction de transition.*

Remarque 4.2. On peut lire la fonction de transition

$$\Delta(q, \sigma) = (q', \sigma', d)$$

(avec $d \in \{-1, 1\}$) par : *si on se trouve dans l'état q et la tête de lecture lit la lettre σ, alors on remplace σ par σ', on rentre dans l'état q', et on déplace la tête sur le ruban en direction d* (c'est-à-dire, à gauche si $d = -1$, à droite si $d = +1$).

Définition 4.3. Une *configuration* de T est un triplet (r, x, q) où :
- r (le *ruban*) est une fonction $r : \mathbb{Z} \to \Sigma_\square$; en plus, on demande que $\{ z \in \mathbb{Z} \mid r(z) \neq \square \}$ soit fini ;
- x (la position de la *tête de lecture*) appartient à $\mathbb{Z}$,
- $q \in Q$ est l'état de contrôle.

Pour $w = \sigma_1, \ldots \sigma_n \in \Sigma^*$, la *configuration initiale sur entrée w* est le triplet $ci_w := (r_w, 0, q_0)$ où $r_w(x) = \sigma_{x+1}$ pour $x = 0, \ldots, n-1$, et $r_w(x) = \square$ sinon. Une *configuration* (r, x, q) est *finale* si $q = q_f$.

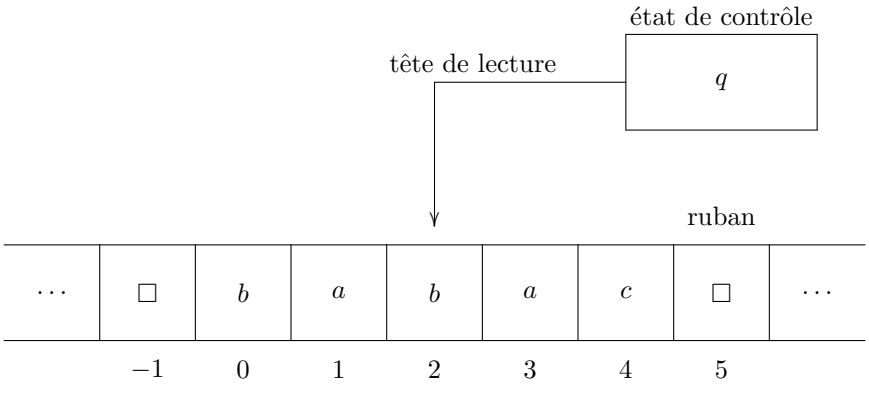

FIGURE 4.1 – Machine de Turing, intuitions

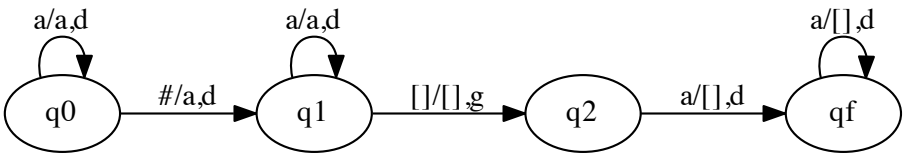

FIGURE 4.2 – Machine de Turing pour la somme

Définition 4.4. On pose $(r, x, q) \vdash_T (r', x', q')$ (à lire : la machine de Turing fait une étape de calcul de la configuration (r, x, q) vers la configuration (r', x', q')) si $\Delta(q, r(x)) = (q', \sigma', d)$ et le conditions suivantes sont satisfaites :

— $x' = x + d$,

— $r'(x) = \sigma'$ et $r'(z) = r(z)$ pour $z \neq x$.

On pose $(r, x, q) \vdash_T^* (r', x', q')$ s'il existe une suite de configurations $(r, x, q) = c_0, \ldots, c_n = (r', x', q')$ telle que $c_i \vdash_T c_{i+1}$, pour $i = 0, \ldots, n-1$.

Remarquez que, vue la définition donnée, une machine de Turing est déterministe et complète ; c'est-à-dire que, pour toute configuration c, il existe toujours une et une seule configuration c' telle que $c \vdash_T c'$. Nous allong faire l'hypothèse suivante :

$$\textit{pour tout } \sigma \in \Sigma_\square, \textit{ on a } \Delta(q_f, \sigma) = (q_f, \sigma, +1)\,.$$

C'est-à-dire, une machine de Turing, de qu'elle rentre dans une configuration finale, elle reste dans une configuration finale et ne change pas le contenu du ruban ; le plus qu'elle fait est de déplacer la tête de lecture vers la droite.

La raison d'imposer ce comportement découle de la nécessite de spécifier que veut dire que une machine de Turing calcule un résultat : le résultat sera le contenu du ruban, de la première configuration finale rencontrée. Autrement, on peut imaginer que la machine de Turing ne fait p[lus de calculs quand elle rentre dans un état finale.

Définition 4.5. On dit que T *s'arrête sur entrée* w (et on écrit cela par $T(w)\downarrow$) s'il existe une configuration finale cf telle que $ci_w \vdash_T^* cf$. Sinon, on dit que T *diverge sur entrée* w (et on écrit cela par $T(w)\uparrow$).

Si $T(w)\downarrow$, alors la machine rentre dans une première configuration finale (r_f, x, q_f) ; ensuite la tête de lecture se déplace vers la droite sans changer le contenu du ruban. Sauf si toutes les cases de r contiennent le symbole $\square$, nous pouvons alors poser

$$m = \min(r_f) = \max\{\, z \in \mathbb{Z} \mid \forall z' < z\ r(z') = \square \,\}\,, \quad M = \max(r_f) = \min\{\, z \in \mathbb{Z} \mid \forall z' > z\ r(z') = \square \,\}\,.$$

Le mot contenu sur le ruban qui se commence à la position $\min(r_f)$ et se termine à la position $\max(r_f)$,

$$r(m)r(m+1)\ldots r(M)\,,$$

est le *résultat du calcul de* T *sur entrée* w. Si w' est ce mot, nous écrirons alors $T(w) = w'$. Attentions : nous pouvons donner un sense à la relation $T(w) = w'$ seulement si la condition $T(w)\downarrow$ est satisfaite. Car une machine de Turing est déterministe, il existera au plus un seul w' tel que $T(w) = w'$.

4.2 Problèmes de décision

Définition 4.6. Un *problème de décision* est une fonction $P : I_P \to \{\, Oui, Non \,\}$. On dit que I_P est l'ensemble des *instances de* P.

Exemple 4.7. SAT est le problème de décision suivante : une instance de SAT (donc un élément de $I_{\texttt{SAT}}$) est un ensemble fini de clauses ; pour un tel ensemble de clauses $\mathcal{C}$, on aura $\texttt{SAT}(\mathcal{C}) = Oui$ si $\mathcal{C}$ est satisfaisable, et $\texttt{SAT}(\mathcal{C}) = NON$ sinon.

En principe, tous les problèmes peuvent se réduire à des problèmes de décision. Considérons, par exemple, le problème de trouver le nombre chromatique d'un graphe. Soit C_n le problème de décision, dont les instances sont les graphes non-orientés, et tel que $C_n(V, E) = Oui$ ssi (V, E) possède un coloriage avec n couleur. Pour trouver le nombre chromatique d'un graphe, nous pouvons résoudre les problèmes $C_1, \ldots C_k \ldots$ jusqu'à trouver une réponse Oui.

Un *codage* est une fonction injective qui permet de représenter les instances d'un problème comme des mots sur l'alphabet Σ d'une machine de Turing T. Nous allons supposer qu'il existe toujours un codage canonique des l'ensemble de réponses $\{ Oui, Non \}$ sur le langage d'une machine de Turing ; par conséquent, nous ne ferons pas mention de pas ce codage dans la suite.

Définition 4.8. Un problème de décision P est *décidable* s'il existe une machine de Turing T et un codage $\gamma : I_P \to \Sigma^*$ telle que :

1. $T(w)\downarrow$, pour tout $w \in \Sigma^*$;
2. pour toute instance $i \in I_P$, $T(\gamma(i)) = Oui$ ssi $P(i) = Oui$.

Attention, dans la définition précédente on ne peut pas omettre la première clause. Si on le fait, on obtient la définition de problème semi-décidable :

Définition 4.9. Un problème de décision P est *semi-décidable* s'il existe une machine de Turing T et un codage $\gamma : I_P \to \Sigma^*$ tels que, pour toute instance $i \in I_P$: $P(i) = Oui$ ssi $T(\gamma(i))\downarrow$ et $T(\gamma(i)) = Oui$.

La notion de semi-décidabilité est très différente, elle ne dit rien sur l'arrêt de la machine pour une instance i telle $P(i) = Non$: en fait, dans ce cas, on pourrait avoir $T(\gamma(i))\downarrow$ *ou bien* $T(\gamma(i))\uparrow$.

On peut aisément se convaincre de la caractérisation suivante :

Proposition 4.10. *Un problème de décision P est semi-décidable s'il existe une machine de Turing T et un codage $\gamma : I_P \to \Sigma^*$ tels que, pour toute instance $i \in I_P$,*
— *si $P(i) = Oui$ alors $T(\gamma(i))\downarrow$,*
— *si $P(i) = Non$ alors $T(\gamma(i))\uparrow$.*

Étant donne P, soit $\neg P$ le problème avec les mêmes instances de P tel que $(\neg P)(i) = Oui$ ssi $P(i) = Non$. On peut montrer la proposition suivante :

Proposition 4.11. *Si P et $\neg P$ sont semi-décidables, alors P est décidable.*

Observez que l'implication inverse, si P est décidable, alors P e $\neg P$ sont semi-décidables, est trivialement vraie.

4.3 Un problème indécidable

Le problème de l'ARRÊT a comme instances les couples (T, w) avec T une machine de Turing et w un mot sur l'alphabet Σ de la machine. $\texttt{ARRÊT}(T, w) = OUI$ si et seulement si la machine de Turing T s'arrête sur entrée w (ce que nous avons noté par $T(w)\downarrow$).

Proposition 4.12. *Le problème de l'ARRÊT n'est pas décidable.*

Démonstration. Par l'absurde : soit $\mathcal{N}$ une machine de Turing qui s'arrête toujours, et soit γ un codage, tels que, pour tout (T, w), on ait

$$T(w)\downarrow \quad \text{ssi} \quad \mathcal{N}(\gamma(T, w)) = Oui\,.$$

Soit Σ l'alphabet de $\mathcal{N}$; nous pouvons supposer que $\gamma(T, w)$ est de la forme $\alpha(T)\sharp\beta(w)$, où α est un codage des machines de Turing vers les mots sur l'alphabet $\Sigma \setminus \{\sharp\}$, et β est un codage des mots sur un langage avec un nombre arbitraire de symboles vers les mots sur $\Sigma \setminus \{\sharp\}$. On a donc :

$$T(w)\downarrow \quad \text{ssi} \quad \mathcal{N}(\alpha(T)\sharp\beta(w)) = Oui\,.$$

Construisons une machine de Turing $\mathcal{D}$ comme suit. $\mathcal{D}$ prend un mot w sur le ruban et il ajoute à la droite du mot w son codage $\beta(w)$, separé par le symbole $\sharp$. Le mot $w\sharp\beta(w)$ se trouvera à ce point sur le ruban. Ensuite $\mathcal{D}$ utilise $\mathcal{N}$ comme un sous-module, avec $w\sharp\beta(w)$ en entrée. Si $\mathcal{N}$ réponds Oui, alors $\mathcal{D}$ rentre dans une boucle infinie et ne s'arrête pas. Si $\mathcal{N}$ réponds Non, alors $\mathcal{D}$ rentre dans l'état final et s'arrête.

Maintenant, posons nous la question si $\mathcal{D}$, avec entrée $\alpha(\Delta)$, s'arrête.

Si c'est le cas, c'est parce que $\mathcal{N}$, avec entrée $\alpha(\mathcal{D})\sharp\beta(\alpha(\mathcal{D}))$ a répondu Non ; par les hypothèses faites sur $\mathcal{N}$, cela est équivalent à dire que $\mathcal{D}$ avec entrée $\gamma(\mathcal{D})$ ne s'arrête pas (on a donc obtenu une contradiction).

Par ailleurs, si c'est ne pas le cas, c'est parce que $\mathcal{N}$, avec entrée $\alpha(\mathcal{D})\sharp\beta(\alpha(\mathcal{D}))$ a répondu Oui ; par les hypothèses sur $\mathcal{N}$, cela est équivalent à dire que $\mathcal{D}$ avec entrée $\alpha(\mathcal{D})$ s'arrête (on a donc une autre contradiction). □

Remarque 4.13. Que l'encodage $\gamma(T, w)$ soit de la forme $\alpha(T)\sharp\beta(w)$ est un raccourcis pour ne pas alourdir la preuve par des détails techniques, qui sont par ailleurs nécessaires. Soit $\mathcal{MDT}$ l'ensemble des Machine de Turing ; nous supposons en fait que :

1. il existe un alphabet fini Σ_0 tel que $\sharp \notin \Sigma_0$;
2. nous disposons d'un encodage canonique $\alpha : \mathcal{MDT} \to \Sigma_0^*$,
3. nous disposons un encodage canonique $\gamma : \mathbb{N}^* \to \Sigma_0^*$;
4. l'alphabet de n'importe quelle machine de Turing est de la forme $\Sigma = \{0, \ldots, n\}$ pour un quelque $n \geq 0$;
5. si P est un problème de décision tel que $I_P \subseteq \Sigma^*$, alors on a pas besoin d'encoder les instances de P comme des mots sur un autre alphabet.

Clairement, les hypothèses (1) à (4) ne posent pas de problèmes (via un renommage du symbol $\sharp$). L'hypothèse (5) peut être affaiblie, en demandant que si $I_P \subseteq \Sigma^*$, alors encodage de I_P soit calculé par une machine de Turing.

Étant dit cela et en dénotant par Σ_T l'alphabet d'une machine de Turing T, la façon correcte de définir le problème de l'arrêt est la suivante :

$$I_{\texttt{ARRÊT}} = \{\, \alpha(T)\sharp\beta(w) \in (\Sigma_0 \cup \{\sharp\})^* \mid T \text{ une machine de Turing, } w \in \Sigma_T^* \,\},$$
$$\texttt{ARRÊT}(\alpha(T)\sharp\beta(w)) = Oui \text{ ssi } T(w)\downarrow .$$

Remarquons en autre que le probleme de lárret est semi-décidable. En fait, le Theoreme suivant etablie quíl existe une machine de Turing U (car elle est dite universelle), qui se comporte comme un systeme décploitation : etant donnée un entrée la description d'une machine de Turing, elle simule la machine donnée en etrée, et va produire un codage de cette machine.

Théorème 4.14. *Il existe une machine de Turing U, telle que, pour tout toute entre de la forme $\alpha(T)\sharp\beta(w)$, on a que :*

1. *$T(w)\downarrow$ ssi $U(\alpha(T)\sharp\beta(w))\downarrow$;*
2. *si $T(w)\downarrow$, alors $U(\alpha(T)\sharp\beta(w)) = \beta(T(w))$.*

Corollaire 4.15. *Le problème de l'arrêt est semi-décidable.*

Car il suffit de construire une machine A, qui utilise la machine U comme un subroutine : de que U rentre état final (et donc elle retourne en tant que subroutine), A remplace le contenu du ruban par le codage de la réponse positive Oui.

4.4 Autres modèles de calcul et thèse de Chruch

D'autres modèles de calcul ont été proposé.

4.4.1 Fonctions récursives

La classe des fonction récursive est une classe de fonctions partielles (possiblement totales) de la forme $f : \mathbb{N}^n \to \mathbb{N}^m$, définie comme la plus petite classe de fonctions (partielles) contenant certaines fonction élémentaires et fermée sous certains schémas de définition.

Nous allons présenter ces schémas. En définissant la valeur $f(x)$ d'une fonction partielle, nous pouvons utiliser d'autres fonctions partielles qui peuvent ne pas être eux-mêmes définie sur ces valeurs. Si c'est le cas, alors il est implicite que la valeur de $f(x)$ n'est pas défini. Si nous souhaitons dire explicitement que la valeur de $f(x)$ n'est pas défini, alors nous allons écrire $f(x) = \perp$.

Schéma de duplication. Etant donné $f_i : \mathbb{N}^n \to \mathbb{N}^{n_i}$ avec $i = 1, \ldots, k$, posons

$$h(x) = (f_1(x), \ldots, f_n(x))$$

La fonction $h : \mathbb{N}^n \to \mathbb{N}^m$, avec $m = \sum_{i=1,\ldots,k} n_i$ est dite définie par duplication depuis les f_i.

Exemple 4.16. Si $f_1 = f_2$ est la fonction identité, alors la fonction diagonale $\Delta(x) = (x, x)$, qui duplique la valeur de x, est définie par duplication depuis f_1 et f_2.

Schéma de composition. Soit $f : \mathbb{N}^n \to \mathbb{N}^m$ et $g : \mathbb{N}^n \to \mathbb{N}^k$. Posons

$$h(x) = g(f(x)) \,.$$

On dit que $h : \mathbb{N}^n \to \mathbb{N}^k$ est définie par composition à partir de f et g.

Schéma de récursion primitive Soient $f : \mathbb{N}^{1+m} \to \mathbb{N}^m$ et $g : \mathbb{N}^n \to \mathbb{N}^m$. Posons :

$$h(0, y) = g(y) \,, \qquad\qquad h(x + 1, y) = f(x, h(x, y)) \,.$$

On dit que $h : \mathbb{N}^{1+n} \to \mathbb{N}^m$ est définie par récursion primitive depuis f et g.

Exemple 4.17. La fonction somme satisfait aux identités suivantes :

$$\begin{aligned} somme(0, y) &= y \,, \\ somme(x + 1) &= 1 + somme(x, y) \,. \end{aligned}$$

Ainsi, nous pouvons dire que la fonction somme est définie à partir de f et g, où g est la fonction identité, et $f(x, y) = 1 + y$.

Schéma de minimisation. Soit $f : \mathbb{N}^{1+n} \to \mathbb{N}^m$ une fonction partielle. Pour $y \in \mathbb{N}^n$, posons

$$G(y) = \{\, x \in \mathbb{N} \mid f(x, y) \neq \underbrace{(0, \ldots, 0)}_{m \text{ fois}} \,\} \,, \qquad g(y) = \begin{cases} \min G(y) \,, & \text{si } G(y) \neq \varnothing \\ \perp \,, & \text{sinon.} \end{cases}$$

On dit alors que g est définie par minimisation depuis f.

Définition 4.18. La classe de fonction récursives est la plus petite classe $\mathcal{R}$ de fonctions partielles telle que :

- toute fonction constante appartient à $\mathcal{R}$;
- toute projection appartient à $\mathcal{R}$;
- la fonction successeur s (avec $s(x) = x + 1$) appartient à $\mathcal{R}$;
- $\mathbb{R}$ sous le schémas de duplication, récursion primitive, minimisation.

Théorème 4.19. *Une fonction est calculable par une machine de Turing ssi elle est une fonction récursive.*

4.4.2 Thèse de Church

Ainsi, les deux modèles de calculs, coïncident en ce qui concerne ce qui calculent. La thèse de Church (qui n'est pas un théorème!!!) prétend que pour n'importe quel modèle de calcul, le fonctions calculés par ce modèles sont les mêmes que celle calculé par les machines de Turing.