CH.7 Machines de Turing

- 7.1 Les machines de Turing
- 7.2 La reconnaissance, la génération et le calcul
- 7.3 Les modèles équivalents
- 7.4 Le problème de l'arrêt

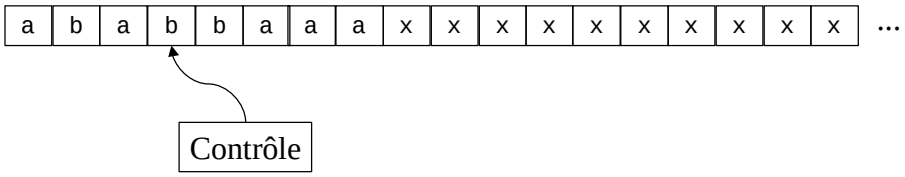
7.1 Les machines de Turing

Modèle introduit en 1936 par Alan Turing pour formaliser la notion d'algorithme.

Apparaît en même temps que le théorème d'incomplétude de Gödel (1931) qui établit qu'il n'existe pas d'algorithme pour décider la vérité d'un prédicat du premier ordre sur les nombres entiers, problème posé par Hilbert au début du siècle.

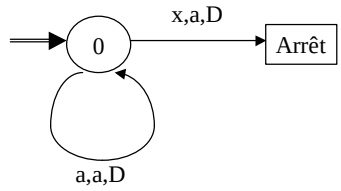
Un des derniers descendants de cet esprit est la démonstration par Matijasevitch de l'indécidabilité de l'existence de solutions d'une équation en nombres entiers (1974).

Machine de Turing :



Bande infinie (mémoire) et tête de lecture-écriture.
Contrôle fini (programme) avec changement d'état, déplacement de la tête et écriture en fonction du symbole lu sur la bande.
La bande contient au départ un certain mot complété par des x (blancs).
Elle est dans un état initial et la tête est sur la première case.
La machine s'arrête lorsqu'elle atteint un état spécial d'arrêt.

Par exemple :



Si la bande au départ contient n symboles a justifiés à gauche, à l'arrêt, celle-ci en contient n + 1.

L'arrêt de la machine n'est pas garanti (boucles, déplacement infini de la tête vers la droite).

7.2 La reconnaissance, la génération et le calcul

Trois utilisations possibles des machines de Turing.

·**Reconnaissance** :
Au début la bande contient un certain mot fini. Ce mot est **accepté** si la machine s'arrête dans un état d'arrêt **terminal**.

Un langage reconnu par une machine de Turing est appelé **récursivement énumérable** R(M). Si un mot n'est pas dans le langage, la machine peut ne pas s'arrêter (donc pas de **décision**).

Un langage pour lequel il existe une machine de Turing qui s'arrête pour toutes les entrées est appelé **récursif**. L'appartenance d'un mot à ce langage peut être décidée.

·**Génération**
Au début, la bande est vide. La machine N écrit sur la bande des mots, l'un après l'autre, pas nécessairement dans l'ordre.
L'ensemble de ces mots constitue un langage, fini ou infini G(N).

Théorème :
Si L = R(M) est récursivement énumérable, alors il existe N telle que L = G(N). Réciproquement, si L = G(N), alors il existe M telle que L = R(M).

Ce théorème est effectif dans les deux sens.

Théorème :
Si L = R(M) est récursif, alors il existe N telle que L = G(N) et telle que N écrive les mots de L dans l'ordre lexicographique. Réciproquement, si L = G(N) et si N écrit les mots de L dans l'ordre lexicographique, alors il existe M telle que L = R(M) et M s'arrête sur toute entrée (donc L est récursif).

Ce théorème est effectif dans le sens direct. Mais la réciproque n'est pas effective ! On sait que M existe, mais on ne sait pas la construire.

·**Calcul** :
Au début, la bande contient n symboles 1. A l'arrêt, elle contient f(n) symboles 1. Elle peut ne pas s'arrêter.
Si elle s'arrête toujours, la fonction est définie pour tout n. Elle est appelée **fonction récursive totale**.
Sinon elle est appelée **fonction récursive partielle**.
On peut étendre cette définition aux fonctions de plusieurs variables.

Par exemple, la fonction f(n) = n + 1 (fonction successeur) est récursive totale. De même pour toutes les fonctions arithmétiques usuelles.

7.3 Les modèles équivalents

On peut généraliser les machines de Turing : plusieurs bandes, bande infinie dans les deux sens, introduction du non-déterminisme.
Ces modèles sont effectivement équivalents au modèle initial.

On peut définir des automates à deux ou plus de deux piles. De nouveau, ces modèles sont effectivement équivalents.

On peut définir des machines à accès direct (unités centrales avec langage d'assemblage). Ce modèle est encore équivalent.

Ce dernier point établit l'équivalence des machines de Turing et des ordinateurs munis d'un langage de programmation évolué (tel que C).

Cet ordinateur est bien sûr idéal (mémoire potentiellement infinie).

D'autres approches vers la théorie de la calculabilité ont été faites:
Church et le λ-calcul (1941),
Kleene et les fonctions récursives (1952),
Les systèmes de Post (1943).

Toutes ces approches sont équivalentes, ce qui renforce la **thèse de Post**, que tous les modèles raisonnables de calculabilité sont équivalents.

7.3 Le problème de l'arrêt

Beaucoup de résultats d'indécidabilité sont conséquences de (se réduisent à) l'indécidabilité de l'arrêt d'une machine de Turing.

Nous allons raisonner sur des programmes écrits en C, sur un ordinateur classique (disposant tout de même d'une mémoire infinie, et opérant avec des entiers arbitraires.

Un programme est une chaîne de caractères ASCII. Comme tel, il peut être vu comme équivalent à un nombre entier, écrit en base 128.

A tout entier correspond en effet une chaîne de caractères, qu'on peut essayer de compiler, puis, le cas échéant, d'exécuter.

Par exemple, le plus petit entier correspondant à un programme exécutable correspond à `main(){}` ; il vaut 61 791 792 819 502 589 !

Le plus petit entier correspondant à un programme écrivant un nombre est N, celui de `main(){printf("0");}`.

On peut passer un paramètre en ajoutant la déclaration de variable globale `int A = k` où k est un entier quelconque en tête du programme P_n. Puis on compile, on exécute si on peut, et on regarde la sortie. Si celle-ci est constituée d'un seul nombre r, on pose $f_n(k) = r$. Dans tous les autres cas, **en particulier si le programme boucle**, on pose $f_n(k) = -1$.

Ceci définit une fonction pour tout entier k.

Par exemple, $f_1(k) = f_2(k) = ... = -1$ pour tout k jusqu'à $f_N(k) = 0$ pour tout k.

Si n correspond à `main{printf("A");}`, la fonction correspondante est l'identité $f_n(k) = k$.

Si n correspond à `main(){for(;;);}`, la fonction correspondante est l'identité $f_n(k) = -1$. pour cause de boucle.

Si n correspond à `main(){printf("%d",++A);}`, la fonction correspondante est le successeur $f_n(k) = k + 1$.

Supposons qu'il existe un programme testant l'arrêt, c'est-à-dire une fonction récursive totale **`arret(k, n)`** retournant 1 si le programme de numéro n boucle avec comme paramètre k et retournant 0 sinon.

On peut alors construire un programme dépendant de la variable globale A, initialisée par **`int A = k`** faisant le calcul suivant :
- si **`arret(A, A)`** vaut 1, retourner 0 ;
- sinon, écrire A en base 128 ; la chaîne de caractères correspondante est le code du programme de numéro A ; e compiler ;s'il est exécutable et si son exécution renvoie une valeur r, retourner r + 1 ; dans les autres cas, retourner 0.

Le programme précédent ne boucle pas et retourne donc toujours une valeur, qui est $f_k(k) + 1$ lorsqu'on lui ajoute l'initialisation **`int A = k`**.

Ce programme a un numéro, s. Que vaut $f_s(s)$? Il vaut $f_s(s) + 1$. D'où une contradiction. La fonction **`arret(k, n)`** n'est donc pas calculable.

Théorème :
Etant donné un programme, le problème de savoir s'il boucle est indécidable.

Corollaires :
Soit la fonction f(k, n) qui vaut
0 si le programme n'est pas compilable ;
–1 s'il n boucle sur l'entrée k ;
0 s'il ne retourne pas un nombre ;
r s'il retourne le nombre r.
Cette fonction n'est pas récursive.

Soit la fonction g(k, n) = f(k, n) sauf qu'elle n'est pas définie lorsque le programme boucle. Cette fonction est récursive partielle, mais pas récursive totale.