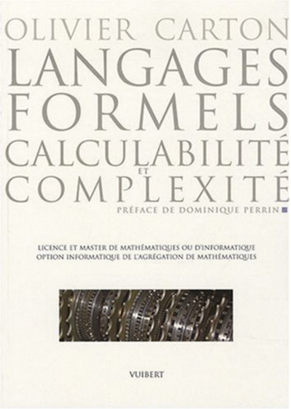
Calculabilité & Complexité

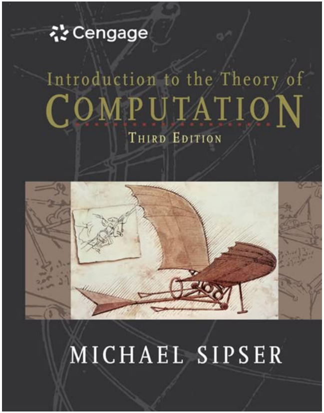
Calculabilité (2/5) : thèse de Church-Turing

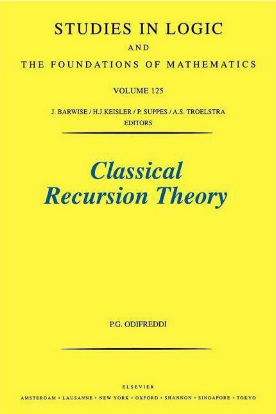
Nicolas Ollinger (LIFO, Université d'Orléans)

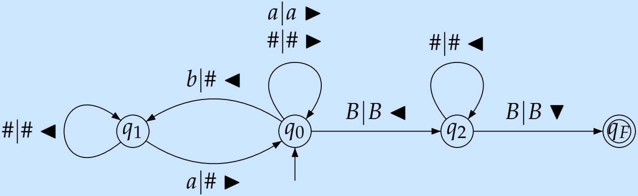
M1 info, Université d'Orléans — S2 2025/2026

Un peu de lecture





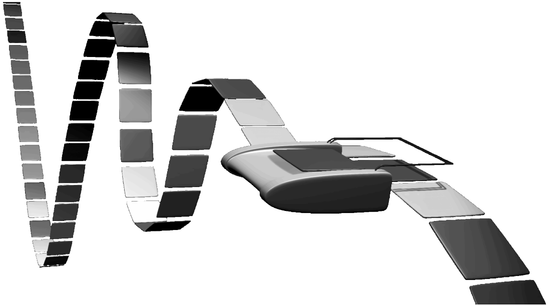




1. Dans l'épisode précédent. . .

Machines de Turing

La **machine de Turing** classique : un **contrôle fini** couplé à un **ruban** biinfini muni d'une **tête** d'E/S mobile pointant sur une cellule.



Formellement

Définition Une **MT** est un tuple $(Q, \Gamma, \Sigma, \delta, q_0, B, q_F)$ où

- Q est l'ensemble fini des **états** ;
- Γ est l'**alphabet** fini de **travail** ;
- $\Sigma \subseteq \Gamma$ est l'**alphabet** fini **d'entrée** ;
- $\delta : Q \times \Gamma \rightharpoonup Q \times \Gamma \times \{\blacktriangleleft, \blacktriangledown, \blacktriangleright\}$ est la **fonction de transition**, partielle ;
- $q_0 \in Q$ est l'**état initial** ;
- $B \in \Gamma \setminus \Sigma$ est le **symbole blanc** ;
- $q_F \in Q$ est l'**état d'acceptation** de la machine.

Une transition $\delta(q, a) = (q', b, \Delta)$ signifie :
« Dans l'état q, lorsque je lis le symbole a,
le remplacer par b, entrer dans l'état q' et se déplacer de Δ »

Échauffement

1. Montrons que la fonction suivante est **Turing-calculable** :

$$\begin{aligned} f:\ \{a,b\}^* &\to \{a,b,\#\}^* \\ u &\mapsto u\#u \end{aligned}$$

2. Montrons que le langage suivant est **récursif** :

$$L = \left\{u \in \{a,b\}^* \,\middle|\, |u|_a = |u|_b\right\}$$

3. Le langage suivant est-il **récursif** ?

$$\Pi = \left\{\mathrm{bin}(n) \,\middle|\, \begin{array}{l}\text{on trouve une suite de } n \text{ chiffres 6 consécutifs} \\ \text{dans l'écriture décimale de } \pi\end{array}\right\}$$

1. Dans l'épisode précédent...

Normalisation

Lorsqu'on choisit une MT pour reconnaître un langage récursif ou récursivement énumérable, il est toujours possible de choisir une MT **normalisée**.

Définition Une MT **normalisée** est une MT munie d'un état de rejet q_R et telle que :

1. tous les calculs finis terminent dans l'état q_F ou dans l'état q_R ;
2. toutes les transitions de la MT sont définies à l'exception de celles issues de q_F et q_R.

Proposition La famille des **languages récursifs** est close par passage au **complémentaire**.

Programmation concurrente

Étant données deux MT $\mathcal{M}_1$ et $\mathcal{M}_2$, il est possible d'en construire une troisième $\mathcal{M}$ qui **simule** les comportements de $\mathcal{M}_1$ et $\mathcal{M}_2$ en effectuant alternativement des pas de calcul de chacunes des machines sur deux pistes séparées codées sur son ruban.

Proposition Un langage est **récursif** si et seulement s'il est **récursivement énumérable** et **co-récursivement énumérable**.

Proposition La familles des langages **récursivement énumérables** est close par **union** et **intersection**.

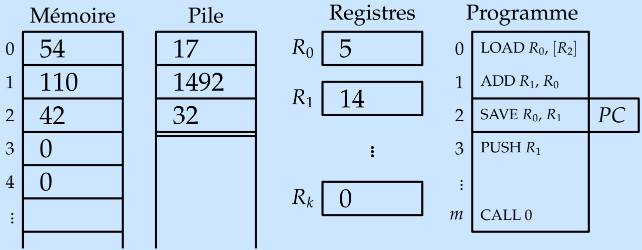
Corollaire La familles des langages **récursifs** est close par **union** et **intersection**.

Un modèle robuste

Le modèle des MT présenté semble très **ad hoc**.

Cependant, la classe des fonctions définies est **robuste** à tout un tas de **variations** :

- ruban mono-infini ;
- plusieurs rubans ;
- mémoire comme une file ;
- mémoire comme deux piles ;
- mémoire 2D ;
- mémoire collection de compteurs unaires ;
- ...



2. Thèse de Church-Turing

Thèse de Church, Turing, Kleene, Post *et al.*

Thèse de Church-Turing Toute fonction calculable par une méthode effective est Turing-calculable.

Théorème (Gandy 1980) Toute fonction **discrète** calculable par un dispositif **mécanique déterministe**, régit par les règles de la physique classique et qui satisfait les **hypothèses** suivantes, est Turing-calculable :

- homogénéité de l'espace ;
- homogénéité du temps ;
- densité bornée d'information dans l'espace ;
- vitesse bornée de propagation de l'information à travers l'espace ;
- quiescence initiale de l'espace de calcul sauf dans une zone bornée.

Turing-complétude

Un **modèle de calcul** décrit une **famille dénombrable** d'objets et la manière de les utiliser pour calculer.

Une **fonction de codage acceptable** décrit une transformation raisonnable entre modèles pour coder les entrées et les sorties.

Définition Deux modèles de calcul se **simulent** entre eux s'ils calculent les même fonctions à un codage acceptable près.

Définition Un modèle de calcul est **Turing-complet** s'il est équivalent au modèle des machines de Turing.

Codages raisonnables

Exemples de codages :

1. coder les **entiers** $\mathbb{N}, \mathbb{Z}$: en unaire, en base k (binaire, octal, hexadécimal)
2. codage de **tuples** $(u_1, \ldots, u_k)$: avec un délimiteur, avec un codage auto-délimité
3. codage de **matrices** ou de **graphes** : recoder les éléments, avec délimiteurs
4. changement d'**alphabet** : recodage des lettres en mots de taille fixe, en code préfixe

Un codage raisonnable ne doit pas permettre de calculer des nouvelles propriétés des objets...

Codage acceptable

Définition Un codage est **acceptable** si :

1. le langage des codages valides est **récursif** ;
2. on peut **calculer** (avec un MdC adapté) le codage d'une entrée ;
3. on peut **extraire** les informations nécessaires du mot codé :
   - entiers : incrémenter, décrémenter, test à zéro, ...
   - graphes : parcours des sommets, test d'adajacence, ...

Remarque Deux codages **acceptables** d'un même ensemble d'objets sont **récursivement équivalents**.

Quelques exemples de modèles Turing-complets

- le λ-calcul de Church ;
- le modèle RAM ;
- votre langage de programmation favori ;
- les fonctions récursives de Herbrand/Gödel/Kleene ;
- les systèmes canoniques de Post ;
- les automates cellulaires ;
- ...

IMP : l'archétype du langage impératif

$$
\begin{array}{llcll}
\text{Variables} & \ni & X, Y & & \\
\text{Expressions} & \ni & E, F & ::= & X \\
 & & & | & \text{Constante entière} \\
 & & & | & E \odot F \text{ avec } \odot \in \{+, -, *, \mathtt{DIV}, \mathtt{MOD}\} \\
 & & & | & E \odot F \text{ avec } \odot \in \{=, <, >, \mathtt{AND}, \mathtt{OR}\} \\
 & & & | & \odot E \text{ avec } \odot \in \{\mathtt{NOT}, -\} \\
\text{Instructions} & \ni & S, T & ::= & X = E \\
 & & & | & S \;\mathtt{;}\; T \\
 & & & | & \mathtt{IF}\ E\ \mathtt{THEN}\ S\ \mathtt{ELSE}\ T \\
 & & & | & \mathtt{WHILE}\ E\ \mathtt{DO}\ S \\
\text{Programme} & \ni & P & ::= & \mathtt{READ}\ X\mathtt{;}\ S\mathtt{;}\ \mathtt{WRITE}\ Y
\end{array}
$$

Calcule des fonctions partielles $f : \mathbb{N} \to \mathbb{N}$.

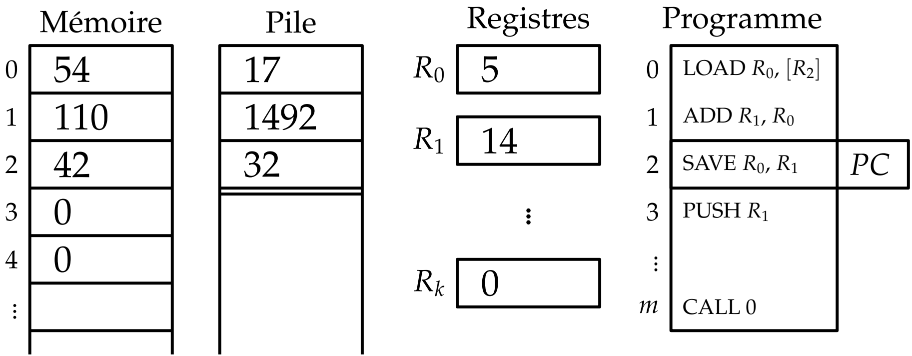
I : forme minimale de IMP

Pour simplifier les simulations entre modèles, il est souvent utile d'avoir une sorte de forme normale minimale du modèle cible.

Théorème Toute fonction calculée par un programme IMP est calculée par un programme I : un programme IMP utilisant au plus **une boucle** et **deux variables**.

Remarque La version à deux variables est un peu technique mais on peut se convaincre facilement qu'on sait le faire avec un nombre constant de variables.

RAM : l'archétype du processeur moderne



Les registres stockent des entiers de taille arbitraire, le jeu d'instructions est choisit suffisament expressif.

Calcule des fonctions partielles $f : \mathbb{N} \to \mathbb{N}$.

REG : forme minimale de RAM

Théorème Toute fonction calculée par un programme RAM est calculée par un programme REG : un programme RAM utilisant seulement **3 registres** et **sans pile ni mémoire**.

Remarque La mémoire, les registres et la pile stockent à chaque instant un tuple d'entiers. Avec un codage astucieux, on peut les coder dans un unique entier. On se garde deux autres registres pour manipuler ce codage.

Turing-équivalence

Théorème Les modèles de calcul **MT**, **IMP** et **RAM** calculent les mêmes fonctions (à un **codage acceptable** près).

Démonstration

1. **MT++** simule **REG** *au programme du TD !*
2. **REG** est équivalent à **RAM**
3. **RAM** simule **I** *par compilation du langage !*
4. **I** est équivalent à **IMP**
5. **IMP** simule **MT** *par simulation des MT !*
6. **MT** est équivalent à **MT++**

où **MT++** désigne les MT enrichies avec rubans mutiples, multi-têtes, semi-infinis, *etc*. ■

Dans le prochain épisode

L'**automate** au cœur de nos **ordinateurs** est fixé *in silico*.

Contrairement aux calculatrices de notre enfance à 4, 10, 100 fonctions, on ne change plus de machine pour avoir de nouvelles possibilités de calcul.

Les **programmes** sont des **données** comme les autres, **chargés** et **exécutés** et même **compilés** sur place.

Cette possibilité existe dans la plupart des modèles de calcul !