Calculabilité, décidabilité.

1 Calculabilité

Une fonction est dit **calculable** s'il existe un algorithme (une suite finie d'instructions) qui calcule cette fonction en un nombre fini d'étapes dans un ordinateur idéal, qui n'a pas de limite de temps et pas de limite d'espace.

Sinon, elle est dite incalculable.

Entre 1930 et 1936, plusieurs modèles de calculs pour formaliser les algorithmes des fonctions calculables sont inventés : les fonctions récursives de Gödel, le lamba-calcul de Church, les machines de Turing.

En 1936, la thèse de Church-Turing affirme que ces définitions sont équivalentes et une fonction calculable dans un modèle l'est dans un autre. On dit que ces modèles sont Turing-complet.

2 Fonctions incalculables

Théorème : il existe une infinité de fonction non calculable.

Démonstration

- Les programmes sont représentés en mémoire par une suite de bits. Il est donc possible d'associer à chaque programme un nombre binaire qu'il est possible de convertir en base 10.

  Ainsi pour chaque entier naturel il existe un unique programme.

- L'argument diagonale de Cantor (en détail à la page qui suit) nous dit que les nombres réels sont un ensemble plus grand que les nombres entiers, et qu'il n'est pas possible d'associer un nombre entier à chaque nombre réel.

  Les deux ensembles sont infinis mais l'ensemble des réels est un infini plus grand que l'ensemble des entiers naturels.

- Ainsi, il est donc impossible d'écrire pour chaque nombre réel un programme qui le calculerait.

Le **castor affairé** est un autre exemple de fonction incalculable, mais cette fois parce que cela prendrait trop de temps, même pour des valeurs d'entrée relativement faibles.

3 Argument diagonal de Cantor

L'argument diagonal de Cantor est une méthode utilisée pour démontrer que certains ensembles sont non dénombrables, c'est-à-dire qu'ils ne peuvent pas être mis en correspondance biunivoque avec l'ensemble des entiers naturels. Cette méthode a été introduite par le mathématicien Georg Cantor pour prouver que l'ensemble des nombres réels est plus grand que l'ensemble des entiers naturels.

1. **Hypothèse initiale** : On suppose, pour le besoin de la démonstration par l'absurde, que l'ensemble des réels dans l'intervalle [0,1] peut être énuméré, c'est-à-dire qu'on peut associer à chaque entier naturel un réel distinct de cet intervalle.
2. **Construction d'une liste hypothétique** : On imagine qu'on a une liste infinie de réels, où chaque réel est représenté sous forme décimale. Par exemple :
   - $r_1 = 0.a_{11}\ a_{12}\ a_{13}\ \ldots$
   - $r_2 = 0.a_{21}\ a_{22}\ a_{23}\ \ldots$
   - $r_3 = 0.a_{31}\ a_{32}\ a_{33}\ \ldots$
   - Et ainsi de suite.
3. **Construction du réel diagonal** : On crée maintenant un nouveau nombre réel en choisissant, pour chaque décimale de la n-ème position, un chiffre qui est différent de celui de la n-ème ligne. Autrement dit, si le n-ème chiffre de r_n est a_{nn}, on choisit un chiffre b_n qui est différent de a_{nn}.
4. **Contradiction** : Le nombre ainsi construit ne peut être dans la liste initiale, car il diffère de chaque r_n par au moins un chiffre (le n-ème chiffre). Cela signifie qu'il existe un réel qui ne figure pas dans la liste, ce qui contredit l'hypothèse initiale selon laquelle tous les réels de [0,1] étaient énumérés.
5. **Conclusion** : L'ensemble des réels n'est pas dénombrable, et donc il existe des ensembles dont la cardinalité est strictement supérieure à celle des entiers naturels.

$$
\begin{array}{ll}
s_1 & = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\ldots \\
s_2 & = 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\ldots \\
s_3 & = 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\ldots \\
s_4 & = 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\ldots \\
s_5 & = 1\,1\,0\,1\,0\,1\,1\,0\,1\,0\,1\ldots \\
s_6 & = 0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\ldots \\
s_7 & = 1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\ldots \\
s_8 & = 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\ldots \\
s_9 & = 1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\ldots \\
s_{10} & = 1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,1\ldots \\
s_{11} & = 1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\ldots \\
\vdots & \quad \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots
\end{array}
$$

$$
s = 1\,0\,1\,1\,1\,0\,1\,0\,0\,1\,1\ldots
$$

4 Problème de l'arrêt

Les fonctions dont la réponse est soit `VRAI` soit `FAUX` sont appelés **problèmes de décision**. Pour les problèmes de décision on utilise les termes **décidables** et **indécidables** à la place de calculable et incalculable.

Le **problème de l'arrêt** est le problème de décision qui détermine, à partir d'une description d'un programme informatique, et d'une entrée, si :

- le programme s'arrête avec cette entrée : il renvoie un résultat
- le programme ne s'arrête pas avec cette entrée : il boucle indéfiniment ou il plante.

Alan Turing a montré en 1936 que le **problème de l'arrêt est indécidable**, c'est-à-dire qu'il n'existe pas de programme informatique qui prend comme entrée une description d'un programme informatique et un paramètre et qui, grâce à la seule analyse de ce code, répond `VRAI` si le programme s'arrête sur son paramètre et `FAUX` sinon. Une partie importante de la démonstration a été la formalisation du concept de programmes informatiques : les machines de Turing.

La preuve de ce résultat est donnée par Turing dans son article fondateur de 1936. Elle repose sur un argument diagonal, tout comme la preuve d'indénombrabilité des réels de Cantor (1891) et celle du théorème d'incomplétude de Gödel (1931). Elle utilise le concept de machine de Turing.

Démontrons que le problème de l'arrêt est indécidable :

1. **Hypothèse initiale** : Supposons qu'il existe un programme `HALT` qui décide du problème de l'arrêt. C'est-à-dire, pour un programme quelconque `PROG` et une entrée `M` :

   Si `PROG(M)` s'arrête, alors `HALT(PROG, M)` renvoie `VRAI` en un temps fini ;

   Si `PROG(M)` ne s'arrête pas, alors `HALT(PROG, M)` renvoie `FAUX` en un temps fini.

   Nous supposons que HALT fonctionne correctement pour tous les programmes et toutes les entrées.

2. **Construction du programme diagonal** : On construit le programme diagonale suivant :

```
1  FONCTION diagonale(x):
2       si HALT(x, x)
3            on boucle à l'infini
4       sinon
5            renvoie VRAI
```

3. **Contradiction :** On obtient des contradictions à l'exécution de `diagonale(diagonale)`.

   Cas 1 : `diagonale(diagonale)` boucle à l'infini (ligne 3), donc `HALT(diagonale, diagonale)` renvoie `VRAI` (ligne 2) et `diagonale(diagonale)` termine. Contradiction.

   Cas 2 : `diagonale(diagonale)` termine (ligne 7), donc `HALT(diagonale, diagonale)` renvoie `FAUX` (ligne 2), et `diagonale(diagonale)` ne se termine pas. Contradiction.

4. **Conclusion** : Cela prouve donc par l'absurde que `HALT` ne peut pas exister.

Les Machines de Turing

Un peu d'histoire

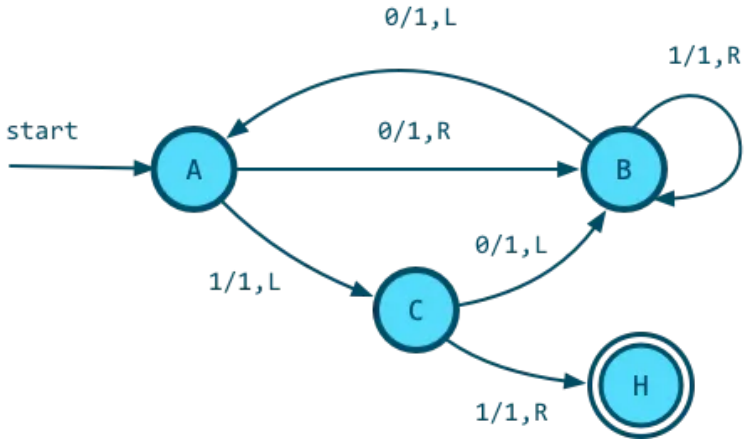
Vous avez peut-être déjà entendu parler d'Alan Turing, à la tête de l'équipe qui a réussi à déchiffrer des messages de la machine Enigma pendant la deuxième guerre mondiale, donnant ainsi un net avantage aux alliés ce qui a nettement réduit la durée du conflit.
Tout juste avant l'arrivée des premiers ordinateurs, ce même Alan Turing a eu l'idée en 1936 d'une machine abstraite (donc pas une vraie que l'on peut toucher, mais juste une description théorique), assez rudimentaire, composée d'un état interne, d'un ruban sur lequel on peut écrire/lire des lettres et de règles (le programme) disant comment faire évoluer cette machine.
A priori rien de bien révolutionnaire, sauf qu'en fait si ! Les fondements que cette machine de Turing a contribué à mettre en place ont pu aboutir quelques années plus tard au concept d'ordinateurs comme vous les connaissez.

Composition de la machine de Turing

La machine de Turing est composée des éléments suivants :

- d'un ruban infini divisé en cases, dans lesquelles la machine peut écrire des symboles d'un alphabet.
- d'une tête de lecture et d'écriture
- d'une table de transition, dont chaque ligne :
  - est associée à un état
  - spécifie les actions à effectuer quand la machine est dans cet état en fonction du symbole lu par la tête de lecture ayant comme actions possibles :
    - Écrire un symbole (choisi dans un alphabet, 0 ou 1 souvent)
    - Se déplacer d'une case sur la gauche ou sur la droite
    - Changer d'état
- S'arrêtant quand on atteint un état désigné comme « terminal »

Une table de transition peut être représentée par un **automate fini déterministe** :



Ici, la table de transition du **castor affairé à trois états** (l'état initial A, le B, le C et l'état terminal H n'étant pas comptabilisé) L'alphabet est compris des caractères 0 et 1.
Chaque flèche est une transition sur laquelle est indiqué :
- le symbole lu sur le ruban,
- le symbole écrit sur le ruban à la place du symbole lu
- la direction vers laquelle on se déplace sur le ruban (L for Left/Gauche et R for Right/Droite)
Par exemple, « 0/1, R » indique que le symbole 0 est lu, que 1 est écrit et qu'on se déplace à droite.

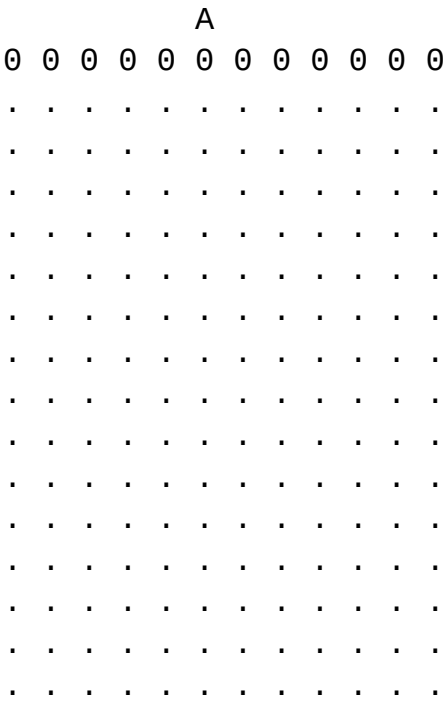
Définition formelle

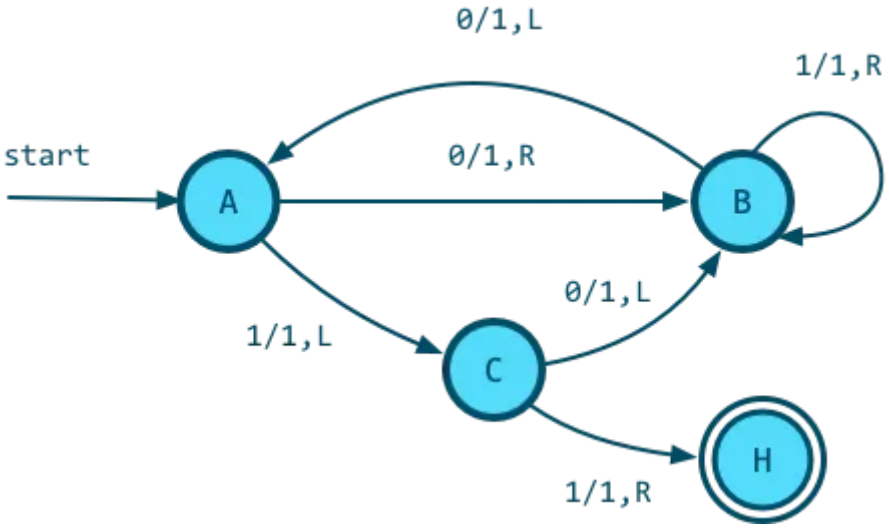
Plusieurs définitions formelles proches les unes des autres peuvent être données d'une machine de Turing. L'une d'elles , relativement courante, est choisie ici. Une machine de Turing est un quintuplet $(Q, \Gamma, q_0, \delta, F)$ où :

- Q est un ensemble fini d'*états* ;
- Γ est l'*alphabet de travail* des symboles de la bande, contenant B un symbole particulier (dit *blanc*), $B \in \Gamma$;
- q_0 est l'*état initial*, $q_0 \in Q$;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la *fonction de transition* ;
- F est l'ensemble des *états acceptants* (ou finals , terminaux), $F \subseteq Q$.

Il s'agit d'un modèle de machine de Turing complète et déterministe ; i.e $\forall q \in Q, \forall \gamma \in \Gamma, \delta(q, \gamma)$ est définie et unique .

Exercice 1 : Exécuter la machine de Turing suivante avec un ruban contenant uniquement des 0.





Exercice 2 : Représenter l'automate correspondant aux transitions suivantes puis tester la machine sur un ruban contenant uniquement des 0.

Pour une machine à deux états (**A** et **B**), le castor affairé correspond à la table de transition suivante

État	Symbole	
	0	1
A	• Écrire le symbole **1** • Déplacer le ruban à **droite** • Passer à l'état **B**	• Écrire le symbole **1** • Déplacer le ruban à **gauche** • Passer à l'état **B**
B	• Écrire le symbole **1** • Déplacer le ruban à **gauche** • Passer à l'état **A**	• Écrire le symbole **1** • Déplacer le ruban à **droite** • Passer à l'état **STOP**

```
          A
0 0 0 0 0 0 0 0 0 0
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
```

Exercice 3 : Dessiner l'automate correspondant aux transitions suivantes puis tester la machine sur un ruban avec des 0.

Pour une machine à quatre états, le castor affairé correspond à la table de transition suivante :

État	Symbole	
	0	1
A	• Écrire le symbole **1** • Déplacer le ruban à **droite** • Passer à l'état **B**	• Écrire le symbole **1** • Déplacer le ruban à **gauche** • Passer à l'état **B**
B	• Écrire le symbole **1** • Déplacer le ruban à **gauche** • Passer à l'état **A**	• Écrire le symbole **0** • Déplacer le ruban à **gauche** • Passer à l'état **C**
C	• Écrire le symbole **1** • Déplacer le ruban à **droite** • Passer à l'état **STOP**	• Écrire le symbole **1** • Déplacer le ruban à **gauche** • Passer à l'état **D**
D	• Écrire le symbole **1** • Déplacer le ruban à **droite** • Passer à l'état **D**	• Écrire le symbole **0** • Déplacer le ruban à **droite** • Passer à l'état **A**

Tester cette automate sur http://turingmachine.vassar.edu/

Combien de 1 apparaissent sur le ruban ?

Castor Affairé

Le castor affairé à n état est la machine de Turing à (n+1) états qui en prenant en entrée un ruban contenant uniquement des 0 doit écrire le maximum de 1 possible sur le ruban avant de s'arrêter.
Vous avez vu dans les question précédentes les castors affairés de 2, 3 et 4 états.

A partir de 6 états, le castor affairé est **incalculable**. Le nombre d'étapes pour que l'algorithme finisse est au moins supérieur à $10 \uparrow\uparrow 15$ (nous ne sommes même pas sûr du nombre d'étapes) , autrement dit le nombre d'étapes qu'elle effectue avant de s'arrêter est au moins :

$$10 \uparrow\uparrow 15 = 10^{10^{\cdots^{10}}}$$ avec 15 itérations de puissance de 10.

On a déjà $10 \uparrow\uparrow 2 = 10^{10} = 10\,000\,000\,000$ et $10 \uparrow\uparrow 3 = 10^{10^{10}} = 10^{10000000000}$, soit bien plus que l'age de l'univers en milliseconde $\approx 4{,}4 \times 10^{20}$, soit bien plus que le nombre d'atomes dans l'univers $\approx 10^{80}$…

Conclusion

Avec cette activité le lien avec l'informatique est clair : on exécute et on écrit des programmes, OK.

Mais la question qui reste est de savoir quel est l'intérêt de continuer à utiliser ce modèle rudimentaire inventé au milieu des années 30. Pourquoi s'en souvenir plus de 80 ans plus tard alors que nous avons des ordinateurs très puissants que nous programmons dans des langages dits "évolués", bien plus proche du langage naturel ? Parce que ce modèle, si rudimentaire soit-il, a la même capacité de calcul que n'importe quel ordinateur.

Oui nos ordinateurs modernes sont bien plus rapides, efficaces, agréables à programmer, mais ce qui est incroyable c'est que pour tout problème qu'un ordinateur sait résoudre, si complexe soit-il, on peut concevoir une machine de Turing (sans doute énorme et peu efficace) qui fait la même chose. Si vous ne voyez toujours pas l'intérêt cela va venir.

Imaginez maintenant que vous trouvez un problème pour lequel vous réussissez à prouver qu'aucune machine de Turing n'est capable de le résoudre. Avec cette preuve et le fait que la machine de Turing sait faire la même chose que les ordinateurs, on obtient automatiquement le résultat suivant : aucun ordinateur, même si dans 100 ans ils sont bien plus puissants, ne pourra résoudre ce problème.

Et faire cette preuve sur les machines de Turing est rendu plus simple car le modèle de ces machines est très précis et les "instructions" ou règles sont d'un type très restreint.

Bonus : application à la recherche textuelle

Représenter une machine de Turing sur un alphabet {0;1} à 4 états qui arrive a un état terminal quand elle à lu un mot 101 sur le ruban.