Calculabilité
Cours 5 : universalité et thèse de Church-Turing

Kévin Perrot – Aix Marseille Université – printemps 2022-23

Table des matières

5 Universalité et complétude

Revenons sur le langage $L_u = \{(\langle M \rangle, w) \mid M \text{ accepte l'entrée } w\}$. Nous avons vu que L_u n'est pas décidable. Cependant, L_u est semi-décidable.

Théorème 1. *Le langage L_u est semi-décidable.*

Démonstration. Plutôt que de décrire une machine de Turing qui reconnait L_u, nous nous contenterons de décrire informellement un semi-algorithme [1] pour déterminer si un mot est dans L_u. Le semi-algorithme commence par vérifier si l'entrée a une forme correcte : un couple composé du code $\langle M \rangle$ d'une machine sur l'alphabet d'entrée Σ, et un mot $w \in \Sigma^*$. Cette vérification est décidable. Ensuite, le semi-algorithme simule la machine M sur l'entrée w jusqu'à ce que (le cas échéant) la machine M s'arrête. Une telle simulation pas à pas peut être effectuée car le semi-algorithme a connaissance du code de M. Le semi-algorithme retourne alors la réponse « oui » si et seulement si M s'arrête dans son état final. □

Nous pouvons en déduire les corollaires suivants.

Théorème 2. *Il existe des langages semi-décidables qui ne sont pas décidables, et la famille des langages semi-décidables n'est pas close par complémentation.*

1. c'est-à-dire un algorithme qui répond oui pour les mots appartenant au langage, mais qui peut ne pas répondre (ou répondre non) pour les mots n'appartenant pas au langage.

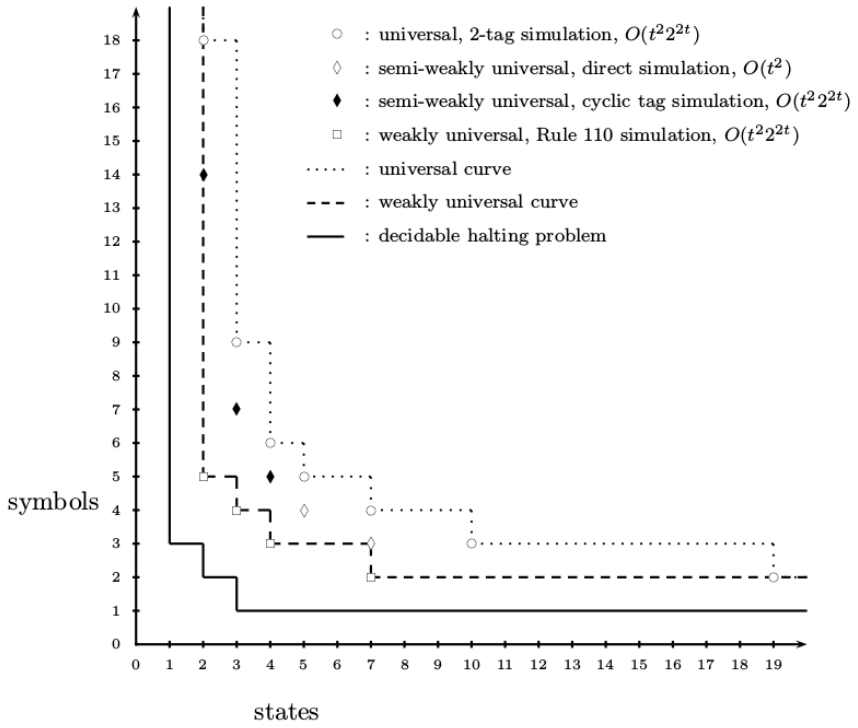

FIGURE 1 – Existence ou non de machines de Turing universelles pour un nombre d'états et de symboles de ruban donné.

5.1 Universalité

Le théorème 1 nous dit qu'il existe une machine de Turing M_u capable de reconnaitre L_u (c'est-à-dire capable de répondre « oui » pour tout mot $w \in L_u$). Une telle machine M_u est appelée une **machine de Turing universelle** car elle peut simuler n'importe quelle machine sur n'importe quelle entrée, si on lui donne une description de la machine à simuler et de l'entrée sur laquelle la lancer :

$$M_u(\langle M \rangle, w) = M(w).$$

M_u est un ordinateur programmable : plutôt que de construire une nouvelle machine de Turing pour tout nouveau langage, on peut utiliser la même machine M_u et changer le **programme** $\langle M \rangle$ qui décrit quelle machine de Turing on souhaite simuler.

Ce concept est très important : imaginez si nous devions construire un nouvel ordinateur pour chaque algorithme que nous souhaiterions exécuter !

MT universelle	$\iff$	ordinateur (système d'exploitation / interpréteur)
ruban	$\iff$	disque dur
$\langle M \rangle$	$\iff$	programme

Il est clair qu'une machine de Turing à 2 états et 2 symboles de ruban ne peut pas être universelle (pensez à son score au busy beaver). Trouver le plus petit nombre d'états et de symboles de ruban nécessaires à la construction d'une machine de Turing universelle est un problème compliqué, auquel ont travaillé Damien Woods et Turlough Neary [3]. La figure 1 est issue de la thèse de doctorat de ce dernier, soutenue en 2008.

Comment construire une machine de Turing universelle ?

Pour prouver leur existence, il suffit de donner un exemple de machine de Turing universelle. On pourrait le faire sans trop de problème à partir de notre encodage $\langle M \rangle$. Ce serait long à décrire entièrement, mais l'idée est simple : la machine universelle M_u prend en entrée un couple $(\langle M \rangle, w)$ avec $w = w_1 w_2 \dots w_n$ codé par

$$\underbrace{0\dots0}_{w_1}1\underbrace{0\dots0}_{w_2}1\dots1\underbrace{0\dots0}_{w_n}.$$

Le couple $(\langle M \rangle, w)$ peut être codé en binaire grâce aux techniques vues en TD, ou plus simplement en utilisant un nouveau symbole séparateur (par exemple #). Pour simuler M sur l'entrée w, elle doit retenir l'état courant de M (en l'écrivant tout à gauche de son ruban par exemple, initiallement q_0 (encodé par 0) de la machine M) et la position courante (l'alphabet de ruban de M_u peut inclure un marqueur sur la case courante), et à chaque étape de M, la machine M_u doit :

1. chercher dans la liste des transitions de $\langle M \rangle$ si il y a une transition définie pour l'état courant et le symbole courant sous la tête de lecture (en comparant une à une avec toutes les transitions),
2. si une transition existe, alors changer l'état, le symbole et la position tel qu'indiqué dans cette transition, et recommender,
3. sinon s'arrêter, dans son état final ou non suivant si M s'arrête dans son état final ou non.

La difficulté principale tient au fait qu'une même machine M_u, qui a un nombre d'états et de symboles de ruban fixé, doit pouvoir simuler toute machine M, quels que soient son nombre d'états et de symboles de ruban. Pour cela M_u écrit ces informations en unaire sur le ruban, et compare en faisant des aller-retours. Ainsi M_u ne retient dans son état qu'une quantité finie d'information, et est bien une machine de Turing.

5.2 Turing-complétude

On appelle **modèle de calcul** la définition mathématique d'un ensemble d'opérations utilisables pour réaliser un calcul (une syntaxe et des règles décrivant la sémantique de la syntaxe). Exemple : les machines de Turing.

Définition 3. *Un modèle de calcul capable de calculer toutes les fonctions calculables par des machines de Turing est appelé* **Turing-complet**.

Remarque 4. *Un modèle de calcul capable de simuler une machine de Turing universelle est Turing-complet.*

Tous les langages de programmation que vous utilisez couramment (C, Haskell, Java, OCaml, Python…) sont bien entendu Turing-complets : si vous pouvez implémenter un simulateur de machines de Turing, alors le langages est capable de calculer toutes les fonctions calculables par des machine de Turing !

Petite liste de modèles, jeux et langages Turing-complets (parfois accidentellement) : `https://en.wikipedia.org/wiki/Turing_completeness#Unintentional_Turing_completeness`

— le λ-calcul (très minimaliste),

— les fonctions μ-récursives (façon calculatoire de définir des fonctions),

— les pavages (un type de puzzles),
— les automates cellulaires (vit-on dans un AC ?),
— les jeux Minecraft et Pokemon jaune (et de nombreux autres),
— Brainf*ck (un langage extrêmement simpliste qui comporte 8 instructions).
— Les briques de Lego™mécaniques (avec engrenages et pistons) :
`http://www.dailymotion.com/video/xrmfie/`.

6 Thèse de Church Turing

Nous avons vu que les machines de Turing ne peuvent pas calculer toutes les fonctions, et même qu'elles ne sont capables d'en calculer qu'une infime partie. Il est légitime de se poser les questions suivantes : est-ce un bon modèle de calcul ? Ne pourrions nous pas définir un modèle de calcul qui puisse calculer un plus grand ensemble de fonctions ?

Définition 5. *Deux modèles de calcul sont* **équivalents** *s'ils sont capables de se simuler mutuellement (donc ils calculent exactement le même ensemble de fonctions).*

Remarque 6. *Les MT non déterministes et multi-rubans sont équivalentes aux MT.*

Il se trouve que tous les modèles de calcul « réalistes » qui ont été définis jusqu'à aujourd'hui sont équivalents aux machines de Turing : ils permettent de calculer exactement le même ensemble de fonctions. *Exactement* le même ensemble de fonctions !

Historiquement, en 1933 Kurt Gödel et Jacques Herbrand définissent le modèle des fonctions μ-récursives. En 1936, Alonzo Church définit le λ-calcul. En 1936 (sans avoir connaissance des travaux de Church), Alan Turing propose sa définition de machine. Church et Turing démontrent alors que ces trois modèles de calcul sont équivalents !

La thèse de Church-Turing, ou plutôt ses deux versions [4], sont des énoncés que la communauté (dans sa majorité) pense vrais, mais qu'il n'est pas possible (du moins c'est le point de vue jusqu'à maintenant) de prouver. Ils énoncent que les machines de Turing capturent « correctement » la notion intuitive de calcul : toute autre façon de calculer, ou de définir le calcul, reviendrait au même [2].

6.1 Thèse de Church-Turing version physique

Thèse 7. *Toute fonction physiquement calculable est calculable par une MT.*

Autrement dit tout modèle de machine, qui peut effectivement exister selon les lois de la physique, sera au mieux équivalent aux machines de Turing, sinon moins puissant (en terme d'ensemble de fonctions calculables).

Robin Gandy (qui a effectué son doctorat sous la direction d'Alan Turing) a travaillé sur cette question et démontré un résultat que nous reproduisons ci-dessous dans une formulation simplifiée [2].

Théorème 8. *Toute fonction calculée par une machine respectant les lois physiques :*

1. *homogénéité de l'espace (partout les mêmes lois),*
2. *homogénéité du temps (toujours les mêmes lois),*

2. les machines quantiques n'échappent pas à la thèse de Church-Turing [1].

3. densité d'information bornée (pas plus de n bits au m^2),

4. vitesse de propagation de l'information bornée (pas plus de c $m.s^{-1}$),

5. quiescence (configuration initiale finie et état de repos tout autour),

est calculable par une machine de Turing.

Est-ce satisfaisant ? Une version quantique de ce théorème a été démontrée [1].

6.2 Thèse de Church-Turing version algorithmique

Thèse 9. *Toute fonction calculée par un algorithme est calculable par une MT.*

Pas facile de définir ce qu'est, ou plutôt ce qu'en toute généralité pourrait être, un **algorithme**. On peut dire qu'un algorithme est exprimable par un programme rédigé dans un certain langage de programmation, qu'il décrit des instructions pouvant être suivies sans faire appel à une quelconque « réflexion ». Définir un modèle de calcul revient à définir une façon d'écrire des algorithmes.

Cette version de la thèse de Church-Turing est parfois appelée version symbolique, car elle exprime ce qu'il est possible de calculer à l'aide de symboles mathématiques auxquels on donne un sens calculatoire.

Rappelons qu'Alan Turing est parti de l'idée d'un calculateur humain avec son stylo devant une feuille de papier, pour construire le modèle des machines qui portent son nom. Les machines de Turing sont-elles assez générales ? ou bien peut-on imaginer une autre façon non-équivalente de décrire les algorithmes, mais qui soit en accord avec notre intuition ? La Thèse 9 postule que les machines de Turing capturent en toute généralité la notion d'algorithme, et que l'on peut s'en contenter.

Malgré de nombreux efforts, la thèse de Church-Turing n'a jusqu'ici jamais été contredite !

`https://fr.wikipedia.org/wiki/Th%C3%A8se_de_Church`

Références

[1] P. Arrighi and G. Dowek. The physical Church-Turing thesis and the principles of quantum theory. *Int. J. of Found. of C.S.*, 23 :1131–1145, 2012. arXiv:1102.1612.

[2] R. Gandy. Church's Thesis and Principles for Mechanisms. *The Kleene Symposium*, 101 :123–148, 1980.

[3] T. Neary and D. Woods. The complexity of small universal turing machines. In *proceedings of CiE'2007, volume 4497 of LNCS*, pages 791–798, 2007. arXiv:1110.2230.

[4] M. Pégny. Les deux formes de la thèse de Church-Turing et l'épistémologie du calcul. *Philosophia Scientiae*, 16(3) :39–67, 2012.