Leçon 913 : Machines de Turing. Applications.

Julie Parreaux

2018 - 2019

Références pour la leçon

[1] Lassaigne et Rougemont, *Logique et fondements de l'informatique. Logique du 1^{er} ordre, calculabilité et λ-calcul.*
[2] Sipser, *Introduction to the Theory of Computation.*
[3] Stern, *Fondements mathématiques de l'informatique.*
[4] Turing et Girard, *La machine de Turing.*
[5] Wolper, *Introduction à la calculabilité.*

Développements de la leçon

Turing-calculable $\Rightarrow$ μ-récursive — Théorème de Savitch

Plan de la leçon

Motivation

Défense

Les machines de Turing on été introduites par Turing en 1936.

Contexte historique : répondre à la question d'Hilbert : Qu'est-ce qui est calculable ? Elle s'inscrive dans la formalisation des modèles de calculs formels qui tendent à répondre à cette question. On peut également cité les fonctions récursives et le λ-calcul ou la logique car calculer c'est prouver (Gödel).

Les modèle de calcul formel permettent d'appréhender les limites non-physiques mais bien conceptuelles du calcul et de l'informatique. Les fonctions récursives s'inscrivent dans une démarche qui identifie les fonctions non-calculables. Turing se tourne quant-a lui vers le calcul et tendent à répondre à la question comment calcule-t-on ? Les machines de Turing sont alors une réponse de logicien à l'action du calcul tel que réalisé par un humain. Elles servent aujourd'hui d'étalon dans la théorie de la complexité. Les machines de Turing semblent "stable" : à vouloir améliorer les Machines de Turing, on retombe toujours sur des machines reconnaissant la même classe de langages : elles semblent englober toute idée de procédure effective.

Ce qu'en dit le jury

Il s'agit de présenter un modèle de calcul. Le candidat doit expliquer l'intérêt de disposer d'un modèle formel de calcul et discuter le choix des machines de Turing. La leçon ne peut se réduire à la leçon 914 ou à la leçon 915, même si, bien sûr, la complexité et l'indécidabilité sont des exemples d'applications. Plusieurs développements peuvent être communs avec une des leçons 914, 915, mais il est apprécié qu'un développement spécifique soit proposé, comme le lien avec d'autres modèles de calcul, ou le lien entre diverses variantes des machines de Turing.

1 Introduction

Motivations :

— Pourquoi avoir un modèle de calcul formel ? Répondre à la question de Hilbert : Qu'est-ce qui est calculable. Plusieurs modèles de calculs existent : les fonctions récursives, le λ-calcul, les machines de Turing, ... Ils tentent tous de répondre à la question.

— Pourquoi les machines de Turing ? Même si elles n'ont pas été le premier modèle de calcul formelle, elle sont aujourd'hui considéré comme le modèle abstrait des ordinateurs (modèle RAM). De plus, même si elles n'ont pas apporté de nouvelles réponses à la question de Hilbert, par son approche novatrice elle a permis de classifier les fonctions calculables (et donc les problèmes).

2 Les Machines de Turing : un modèle de calcul formel

Définition[1, p.115] : Un modèle de calcul.

2.1 Vocabulaire autours des machines de Turing [5, p.103]

Définition : Une machine de Turing déterministe : $(Q, \Gamma, \Sigma, \delta, s, \#, F)$.

Définition : Une configuration $c \in \Gamma^* \times Q \times (\epsilon \cup \Gamma^*(\Gamma \setminus \{\#\}))$. (Attention, la position de la tête de lecture est donnée implicitement par la taille de la première composition.)

Exemple : dessin d'une configuration illustrant cette position implicite

Application [1, p.132] : Les automates finies : elle parcours le ruban une seule fois sans écrire (machines de Turing sans mémoire).

Définition : Configurations suivantes

Définition : Une exécution : suite de configurations

Définition : Langage accepté / langage décidé (Notions clés des théories basés sur les Machines de Turing (décidabilité et complexité))

Exemple : $a^n b^n c^n$ est décidé par une machine de Turing.

2.2 Les machines de Turing calculent [3, p.59]

On veut pouvoir calculer avec les machines de Turing.

Exemples : Additionneur, soustracteur, multiplicateur (par 2, de deux entiers) Dessins ajoutés en annexe

Définition [5] : Fonction calculable par une machine de Turing.

3 Justification de la thèse de Church.

Thèse de Church [5, p.109] : Les fonctions calculables par un algorithme sont les fonctions calculables par une machine de Turing.

C'est une thèse (ni une hypothèse, ni un théorème). Nous ne pouvons pas le montrer formellement (facilement) car la notion d'algorithme ne possède pas de définition formelle. Cependant nous allons mettre en avant deux arguments qui nous conforte cette thèse

— Pas d'extension qui améliore le langage décidé (toute équivalente à une machine de Turing) (sous-section A) ;

— Équivalence avec les autres modèles de calculs évoqué plus tard (sous-section B).

3.1 Les extensions d'une machine de Turing décident les mêmes langages.

Cas des machines à plusieurs rubans [5, p.112]

— *Fonction transition* : $\delta : \mathcal{Q} \times \Gamma^k \to \mathcal{Q} \times \Gamma^k \times \{\leftarrow; \rightarrow\}^k$;

— *Configuration* : $c \in (\Gamma^*)^k \times \mathcal{Q} \times (\epsilon \cup \Gamma^*(\Gamma \setminus \{\#\}))^k$;

— *Simulation par une machine à un ruban* : les cases du i^{ieme} ruban sont représentées par les cases de positions i modulo k.

Cas des machines à ruban bi-infini [5, p.110]

— *Configuration* : $c \in ((\epsilon \cup (\Gamma \setminus \{\#\})\Gamma^*) \times \mathcal{Q} \times (\epsilon \cup \Gamma^*(\Gamma \setminus \{\#\}))$;

— *Simulation par une machine à deux rubans* : on fixe une case d'indice 0. Le premier ruban représente la partie droite du ruban bi-infini et le second la partie gauche.

Cas des machines non-déterministes [5, p.114]

— *Relation du transition* : $\delta \subseteq (\mathcal{Q} \times \Gamma) \times (\mathcal{Q} \times \Gamma \times \{\leftarrow; \rightarrow\})$;

— *Configuration suivante* : la machine choisie parmi l'ensemble des triplets obtenus par la relation de transition ;

— *Langage accepté* : un mot w est accepté si il existe une suite de choix définissant une exécution acceptante. Le langage accepté est l'ensemble des mots acceptés ;

— *Simulation par une machine déterministe à trois rubans* : le premier ruban contient l'entrée en lecture seule. Le second permet de lister les différentes suites des choix possibles (finies). Le troisième simule la machine non-déterministe en effectuant les choix indiqués par le second ruban.

— *Exemple d'utilisation* : satisfiabilité d'une formule du langage propositionnel ; langage pour lequel la déterminisation d'un de ces automate explose (On a bien l'équivalence mais on met en avant que la taille de la machine déterministe explose ce qui a un impact sur la complexité.).

3.2 Les autres modèles de calcul décident les mêmes langages.

Cas des fonctions récursives [5, p.131] :

— *Définition* : Syntaxe des expressions des fonctions μ-récursive.

— *Définition* : Sémantique des expressions des fonctions μ-récursive.

— *Définition* : Fonction μ-récursive.

— *Exemple* : Fonction primitive récursive / Fonction μ-récursive

— *Théorème* : Les fonctions μ-récursives sont exactement les fonctions Turing-calculable. DEV : Turing $\Rightarrow$ μ-récursive.

— *Remarque* : On peut réécrire la thèse de Church avec des fonctions

Cas du λ-calcul [1, p.185] :

— *Définition* : Termes.

— *Exemple* : Codage des entiers avec des termes du λ-calcul.

— *Théorème* : Les fonctions μ-récursives sont exactement représentable par les termes du λ-calcul.

— *Corollaire* : Les fonctions Turing calculable sont exactement représentable par les termes du λ-calcul.

D'autres modèles de calcul pourrait être exploités : les circuits booléens par exemple

3.3 La théorie de la calculabilité. [5, p.139]

La calculabilité se fonde sur la thèse de Church-Turing ; elle permet aussi de montrer l'existence de fonctions non-calculables, par un simple argument de diagonalisation.

Définition [5, p.116] : Machines universelles.

Définition : Les langages R et RE + les langages décidables Donner "l'équivalence" entre langages et programmes

Problème : ARRÊT

entrée une machine de Turing déterministe M ; un mot w

sortie oui si $M(w)$ s'arrête ; non sinon

Théorème : Le problème de l'arrêt est indécidable et dans RE.

Définition : La réduction Dessin.

Théorème : Utilisation des réductions

Problème : PAVAGE DE WANG

entrée une famille finie de tuiles

sortie oui si le jeu de tuiles permet de paver le plan ; non sinon

Application : Le pavage de Wang est indécidable

Théorème : Théorème de Rice

Application : Lien avec sémantique : correction de programmes

4 Les machines de Turing classent les fonctions calculables

Les machines de Turing servent d'étalon pour définir les classes de complexité.

Dans cette partie, nous différencions les machines de Turing déterministes et non-déterministe (le facteur exponentielle existant dans la déterminisation ne nous permet plus d'exploité leur équivalence). Nous avons une première application des différentes variantes de nos machines de Turing.

Définition : Arbre de calcul

Définition : Machine décide un langage en temps/espace f (Attention, en espace : petit piège sur les variantes des machines de Turing : si $f(n) < n$ alors il nous faut une machine à plusieurs rubans).

Définition : classes (N)TIME et (N)SPACE

Définition : classes P -> EXPSPACE

Théorème : Savitch DEV Plusieurs versions du théorème, bien être au claire sur celle que nous développons

Corollaire NPSAPCE = PSPACE

Problème UNIVERSALITÉ

entrée une expression rationnelle E

sortie oui si $L(E) = \Sigma^*$; non sinon

Proposition Le problème UNIVERSALITÉ est dans PSPACE

Définition : NP-complétude

Problème : SAT

entrée une formule ϕ de la logique propositionnelle

sortie oui si ϕ est satisfiable ; non sinon

Théorème : Cook

Définition : classes (N)L avec des machines à plusieurs rubans Nécessité d'avoir plusieurs types de Machines de Turing.

Problème ACCESSIBILITÉ

entrée Un graphe orienté G est deux sommets s et t

sortie Oui si il existe un chemin de s à t dans G

Proposition : Le problème ACCESSIBILITÉ est dans NL.

Remarque : Si G est un graphe non-orienté alors le problème arrive dans L.

Ouverture

On peut parler de la relation grammaire - machine de Turing via la hiérarchie de Chomsky.

Autres modèles de calculs : les circuits booléens.

Quelques notions importantes

Les machines de Turing calculent

Pour exprimer les opérations arithmétiques de base nous exploitons l'équivalence entre une machine de Turing pour laquelle la fonction transition fait toujours avancée et celle où la machine peut rester sur place.

Les machines de Turing sont des modèles de calcul capable d'effectuer ses calculs. La figure 1 montre la multiplication d'un nombre binaire où la fonction de transition est marqué

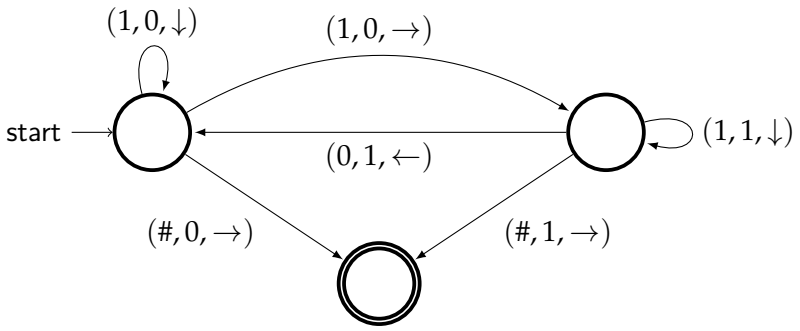

FIGURE 1 – Multiplication par 2

sur les arcs : (a, b, c) où a est la lettre lu sur le ruban, b est la lettre écrite à la place de a et $c \in \{\leftarrow; \downarrow; \rightarrow\}$. (Remarque le nombre sur le ruban est écrit à l'envers.)

Une machine de Turing peut alors additionner deux nombres (Figure 1). Pour cela $\Gamma = \{0; 1\} \cup \# \cup \{\alpha; \beta\} \cup \{C; \$\}$. Elle prend en entré les deux nombres binaires n et m écrit en miroir tel que $\#mCn$. Il commence par remplacer les 1 et les 0 dans les deux nombres par α et β (il fait l'addition bit à bit avec propagation de retenu). Puis il remplace dans le résultat les α et les β par des 0 et des 1 respectivement.

De ces deux particules, on peut obtenir un multiplicateur.

Autres variantes des machines de Turing

Cas des machines à trois actions

— *Fonction transition* : $\delta : \mathcal{Q} \times \Gamma \to \mathcal{Q} \times \Gamma \times \{\leftarrow; \downarrow; \rightarrow\}^k$;

— *Simulation par une machine à deux actions* : lorsque l'action $\downarrow$ se présente, on la décompose en deux actions : une première qui se déplace vers la droite en récrivant le caractère comme dans l'action du $\downarrow$ et la deuxième se déplaçant vers la gauche en laissant le caractère ne place.

Cas des machines RAM [5, p.113][1, p.134] Une machine RAM est une machine semblable à un ordinateur réel. Elle comporte une mémoire à accès direct, un certain nombre de registre et un compteur de programme.

Simulation par une machine de Turing à plusieurs rubans : on munit la machine de Turing d'un ruban pour la mémoire à accès direct (où les informations sont stocké sous la forme (adresse $*$ variable) où l'adresse est le numéro du ruban, séparé par des #), d'un ruban par registre et d'un ruban pour le compteur de programme. La machine va alors lire dans la RAM l'emplacement du compteur de programme qui indique la prochaine instruction à exécuter. Elle va alors exécuter cette instruction (en changeant si besoin les registre et la RAM) et incrémente le compteur de programme de 1 et recommence.

Cas des machines à oracles [1, p.130] Une machine à oracle est une machine à plusieurs rubans et muni d'une fonction $g : \mathbb{N} \mapsto \mathbb{N}$ (un oracle) calculé à l'extérieure de la machine.

Fonctionnement de la machine : un état particulier de la machine q_g est l'état de l'oracle et le premier ruban de travaille est le ruban oracle. Lorsque M est dans q_g et que x est sur le ruban oracle, alors en une unique transition M passe dans l'état q_i et $g(x)$ est inscrit sur le ruban oracle.

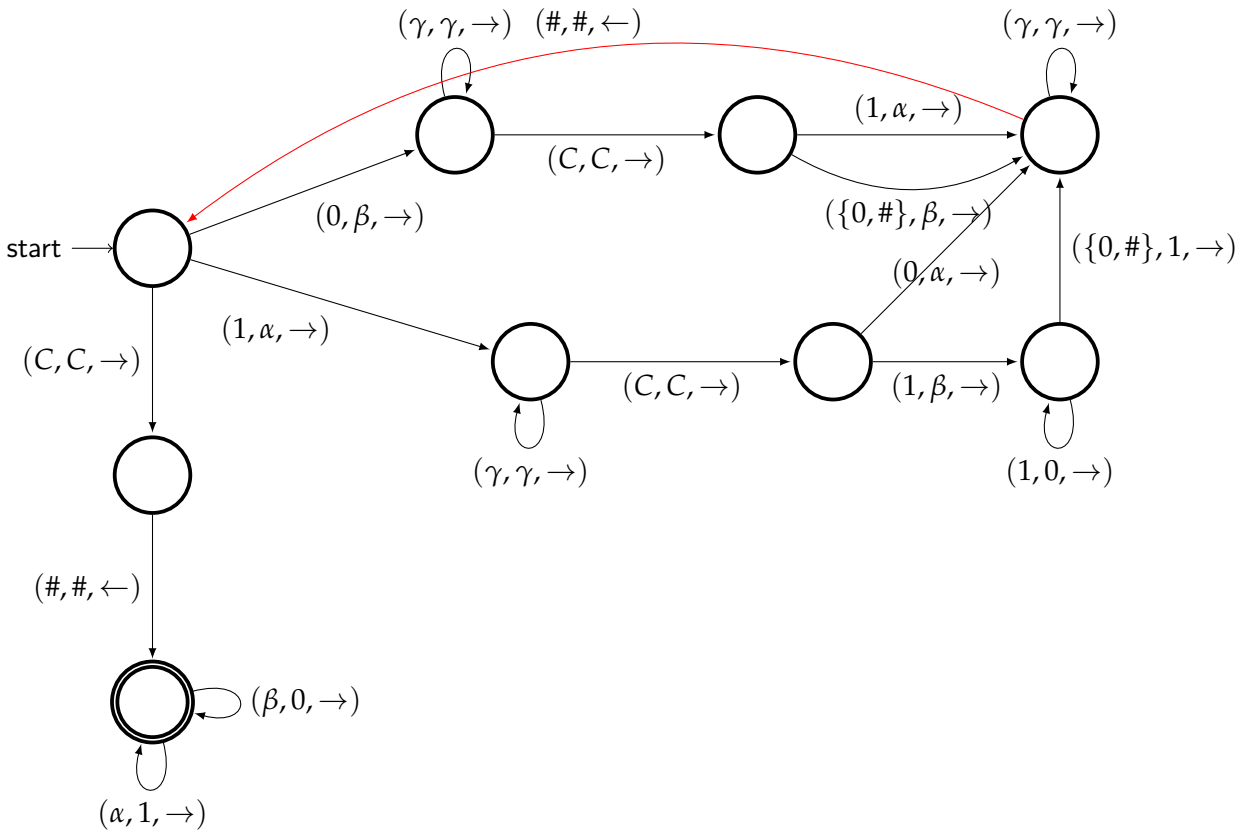

FIGURE 2 – Additionner

Simulation par une machine sans oracle ; Si l'oracle est calculable par une machine de Turing (sinon, ce n'est pas vrai et on rentre dans de nouvelles classes), la machine sans oracle est donc la machine M à qui on substitue la transition issue de q_g par la machine de l'oracle qui rendra la main à M en q_i une fois son calcul effectué.

Cas des machines alternantes [1, p.131] Une machine alternante est une machine qui fait tantôt des choix existentiel tantôt des choix universel (on peut voir ça comme un jeu à deux joueurs dont l'un des joueurs cherche une stratégie gagnante : pour tout choix de son adversaire, il doit trouver un choix tel qu'il gagne).

Application : permet de définir de nouvelles classes de complexité.

Technique de preuve : la réduction

Pour montrer qu'un problème apparaît dans une certaine classe de complexité (et même sa dureté) ou qu'il est indécidable, nous utilisons une technique de preuve : la réduction. Pour appliquer le principe de réduction il nous faut connaître un premier problème possédant les propriétés que l'on souhaite montrer sur le deuxième.

Nous présentons ici le principe de la réduction dans sa généralité puis nous verrons comment le spécialiser pour en faire ce que nous souhaitons.

Définition. Une réduction d'un problème A à un problème B (Figure 3) est une fonction tr calculable telle que pour tout w instance de A, w est une instance positive de A si et seulement si $tr(w)$ est une instance positive de B. On note $A \leq B$.

On dit que A se réduit à B s'il existe une réduction de A à B (intuitivement, A est plus facile que B).

Remarque : En fonction des propriétés sur la fonction tr, on obtient différentes réductions qui vont nous permettre de spécialiser la réduction au résultat que nous souhaitons montrer.

Théorème (Principe de la réduction)**.** *Si A se réduit à B, alors si P est une propriété sur B alors P est une propriété sur A (dans notre cas, P peut être l'appartenance à une classe de complexité ou être le caractère indécidable d'un problème, ...).*

Démonstration. On raisonne par l'absurde et grâce à la fonction de traduction, on obtient une contradiction. □

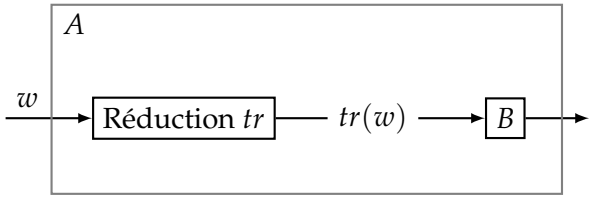

FIGURE 3 – Schéma du principe de réduction du problème A au problème B.

Nous allons donner quelques réductions de A à B et leurs propriétés. On note C une classe de complexité telle que $P \subseteq C$.

Réduction	Propriétés	Conséquences
Calculable	tr est calculable par une machine de Turing (variante de MT : équivalence)	A indécidable $\Rightarrow$ B indécidable B décidable $\Rightarrow$ A décidable
Temps polynomial	tr est calculable par une machine de Turing déterministe en temps polynomial	A est C-dur $\Rightarrow$ B est C-dur ($C \neq P$) $B \in C \Rightarrow A \in C$
Espace logarithmique	tr est calculable par une machine de Turing déterministe en espace logarithmique (la MT a trois rubans)	A est D-dur $\Rightarrow$ B est D-dur $B \in D \Rightarrow A \in D$ ($D \neq P$) avec $D \in \{L, NL, co-NL, P\}$

Remarque : La réduction en espace logarithmique est une réduction très spéciale car une machine qui travail en espace logarithmique a au moins deux rubans (il ne faut pas que l'entrée rentre dans la calcul). Mais pour la réduction, la sortie n'est pas non plus dans la borne de la mémoire utilisé (notons qu'elle est polynomiale (car

une réduction en espace logarithmique s'exécute en temps polynomial)), il nous faut donc un troisième ruban. La machine de Turing effectuant la réduction a donc trois rubans : un ruban contenant l'entrée en lecture seule et en une seule passe ; un ruban de travail logarithmique et un ruban de sortie polynomial en écriture seule et en une seule passe.

Proposition. *Une réduction en espace logarithmique est une réduction en temps polynomial.*

Indécidabilité sur les machines de Turing

Nous allons présenter un résumé sur les problèmes de décisions pour les machines de Turing. La majorité (sinon la totalité) de ces problèmes sont indécidables. Nous donnerons alors quelques éléments de leur preuve d'indécidabilité et de leurs applications.

Définition. Le *problème de l'*ARRÊT sur une machine de Turing déterministe.

Problème ARRÊT
entrée : Une machine de Turing déterministe M ; un mot w
sortie Oui si $M(w)$ s'arrête ; non sinon

Théorème (Un premier problème indécidable [5])**.** *Le problème de l'*ARRÊT *est indécidable et dans RE.*

Idée de la preuve. On donne une machine de Turing universelle $\mathcal{U}$ qui accepte l'entrée $\mathcal{M}$,w si et seulement si $\mathcal{M}$ s'arrête sur w. Ceci montre que le problème de l'arrêt est récursivement énumérable.

Pour montrer que le problème de l'ARRÊT est indécidable, on raisonne par l'absurde. Il existe alors une machine de Turing $\mathcal{A}$ telle que elle s'arrête pour tout entrée $\mathcal{M}$, w et accepte une telle entrée si et seulement si $\mathcal{M}(w)$ termine. On construit une machine PARADOXE (algorithme 1). On remarque que PARADOXE(PARADOXE) ne termine pas si et seulement si $\mathcal{A}$ accepte (PARADOXE, $\langle$PARADOXE$\rangle$) où $\langle$PARADOXE$\rangle$ est le codage de la machine de Turing PARADOXE. Soit PARADOXE(PARADOXE) ne termine pas si et seulement si PARADOXE(PARADOXE) termine. Contradiction. □

Algorithm 1 La procédure PARADOXE de la preuve de l'indécidabilité du problème de l'ARRÊT.

```
1: procedure PARADOXE(M) X
2:     if A accepte (M, ⟨M⟩) then
3:         Boucler
4:     else
5:         Accepter
6:     end if
7: end procedure
```

Algorithm 2 La procédure $\mathcal{N}_{\mathcal{M},w}$ de la preuve du théorème de Rice.

```
1: procedure N_{M,w}(x)
2:     M(w).
3:     if G accepte x then
4:         accepter
5:     else
6:         rejeter
7:     end if
8: end procedure
```

Applications :

— Le problème de l'ARRÊT est le premier problème que nous avons montrer qu'il était indécidable, nous allons donc l'utiliser pour des réductions afin de montrer qu'il n'est pas le seul problème qui est indécidable.

— Le théorème de Rice et ses applications.

— Si on identifie une machine de Turing à un langage de programmation (qui contient une boucle while, par exemple IMP), on vient de montrer que déterminer si un programme d'un tel langage est termine indécidable.

Théorème (Rice [5])**.** *Pour toute propriété non triviale $\mathcal{P}$ sur les langages récursivement énumérables, le problème de savoir si le langage $L(\mathcal{M})$ d'une machine de Turing $\mathcal{M}$ vérifie $\mathcal{P}$ est indécidable.*

Démonstration. Sans perte de généralité, on suppose que $\emptyset \in \mathcal{P}$. On définit le problème $P_{\mathcal{P}}$.

Problème $P_{\mathcal{P}}$
entrée : Une machine de Turing $\mathcal{M}$
sortie : Oui si $L(\mathcal{M}) \in \mathcal{P}$; non sinon

Réduisons ARRÊT à $P_{\mathcal{P}}$ (voir Figure **??**). Soit $G \in \mathcal{P}$. Comme $G \in RE$, il existe une machine G qui accepte G. La réduction tr est définie par $tr(\mathcal{M}, w) = \mathcal{N}_{\mathcal{M},w}$ où $\mathcal{N}_{\mathcal{M},w}$ est la machine décrite dans l'algorithme 2.

1. tr est une fonction calculable : on construit effectivement $\mathcal{N}_{\mathcal{M},w}$ à partir de $\mathcal{M}$ et w ;
2. $(\mathcal{M}, w)$ instance positive de ARRÊT si et seulement si $\mathcal{M}$ s'arrête sur w. Comme

$$L(\mathcal{N}_{\mathcal{M},w}) = \begin{cases} G & \text{si } \mathcal{M} \text{ s'arrête sur } w \\ & \text{sinon} \end{cases}$$

$(\mathcal{M}, w)$ instance positive de ARRÊT si et seulement si $L(\mathcal{N}_{\mathcal{M},w}) \in \mathcal{P}$. Donc, $(\mathcal{M}, w)$ instance positive de ARRÊT si et seulement si $tr(\mathcal{M}, w) = \mathcal{N}_{\mathcal{M},w}$ est instance positive de $\mathcal{P}$.

□

Applications :

— Si on considère $\mathcal{P} = \emptyset$, le problème LANGAGEVIDE est indécidable.
Problème LANGAGEVIDE
entrée : Une machine de Turing $\mathcal{M}$
sortie : Oui si $L(\mathcal{M}) = \emptyset$; non sinon

— Si on identifie une machine de Turing à un langage de programmation, on vient de montrer que la correction d'un programme est un problème indécidable.

Définition. Le *problème de l'*ACCEPTATION sur une machine de Turing déterministe.
Problème ACCEPTATION
entrée : Une machine de Turing déterministe M ; un mot w
sortie Oui si M accepte w ; non sinon

Théorème. *Le problème de l'*ACCEPTATION *est indécidable.*

Idée de la démonstration. On réduit le problème de l'ARRÊT au problème de l'ACCEPTATION : on construit une machine de Turing qui accepte w si et seulement si $\mathcal{M}$ s'arrête sur w où $\mathcal{M}, w$ est une instance de ARRÊT. □

Application : Nous permet de montrer que le problème POST est indécidable.

Définition. Le *problème de* POST sur une famille de tuile.
Problème POST
entrée : Un alphabet fini, Σ et une famille finie de couples de mots $((h_i, bi)_{i=1\ldots n})$ sur Σ
sortie Oui s'il existe $i_1, \ldots i_k \in \{1, \ldots n\}$ tels que $h_{i_1} \ldots h_{i_k} = b_{i_1} \ldots b_{i_k}$; non sinon

Théorème. *Le problème de* POST *est indécidable.*

Idée de la démonstration. On réduit le problème de POST MARQUÉ au problème de POST où POST MARQUÉ est un problème analogue à POST mais dont la première tuile est définie. On montre l'indécidabilité de POST MARQUÉ par réduction du problème de l'ACCEPTATION dans POST MARQUÉ. □

Application : En utilisant le même principe de réduction, on montre par exemple que le problème de validité de la logique du premier ordre est indécidable.

Références

[1] R. Lassaigne and M. de Rougemont. *Logique et fondements de l'informatique. Logique du 1er ordre, calculabilité et λ-calcul.* Hermes, 1993.

[2] M. Sipser. *Introduction to the Theory of Computation*. Cengage learning, 1133187811.

[3] J. Stern. *Fondements mathématiques de l'informatique*. Ediscience internationnal, 1990.

[4] A. Turing and J.-Y. Girard. *La machine de Turing*. Source du savoir, Seuil, 1991.

[5] P. Wolper. *Introduction à la calculabilité*. Dunod, 2006.