Indécidabilité du problème de validité en logique du premier ordre

Julie Parreaux

2018-2019

Référence du développement : Logic in computer science [2, p.131]

Leçons où on présente le développement : 914 (Décidabilité et indécidabilité) ; 924 (Modèles et théorie FO).

1 Introduction

La logique du premier ordre est une logique expressive. En effet, savoir si une formule logique est valide (au sens où pour tout modèle elle est satisfiable) est indécidable. Cette indécidabilité donne de nombreuses applications : la complexité descriptive (à la place des machines de Turing, on considère des fragment de la logique du premier ordre), les théorème d'indécidabilité en base de données, ...

Ce résultat nous permet d'utiliser la notion de réduction calculable (méthode pour prouver l'indécidabilité) à un problème indécidable bien connu POST. On pratique également le jeu de la syntaxe et de la sémantique dans la logique du premier ordre.

Remarques sur le développement

Ce développement met en œuvre une réduction pour prouver l'indécidabilité d'un problème : il faut la faire proprement.

1. Description de l'instance de Post.
2. Description de la traduction.
3. Preuve de l'indécidabilité (pour le sens direct de l'implication aller vite pour se laisser du temps pour la réciproque qui est plus intéressante).

2 Le problème de la validité en logique du premier ordre

Théorème. *Le problème* VALIDE *est indécidable.*

Démonstration. On réduit le problème POST au problème VALIDE.

Par indécidabilité du problème POST, la réduction donne l'indécidabilité du problème VALIDE (se fait par un raisonnement par l'absurde).

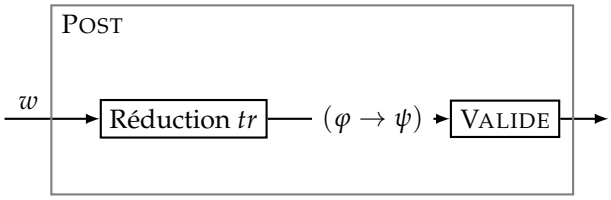

FIGURE 1 – Schéma du principe de réduction du problème POST au problème VALIDE.

Description de l'instance de POST que nous considérons On se place sur l'alphabet $\Sigma = \{a, b\}$. Soit w une instance de POST, autrement dit, un ensemble de N tuiles de la forme $\boxed{\begin{array}{c} u_i \\ \hline v_i \end{array}}$ pour $i \in [\![1, N]\!]$ avec u_i et v_i des mots sur Σ tel que u_i et v_i ne soit pas tous deux le mot vide (ϵ) (cette tuile ne sert à rien et cela nous permet de ne pas la confondre avec le cas où aucune tuile n'est encore posée).

On rappelle que le problème POST est de savoir s'il existe ou non une suite de tuiles telle qu'en les concaténant, les mots du haut et du bas (bornés par la concaténation des tuiles) sont les deux mêmes.

Description de la traduction On pose la traduction suivante : $tr(w) = (\varphi \to \psi)$ avec

— $\varphi = \underbrace{p(\epsilon, \epsilon)}_{\text{constante}} \wedge \bigwedge_{i=1}^{N} (\underbrace{\forall x \forall y}_{\text{pour tout mot}} , (\underbrace{p(x, y) \to p(u_i(x), v_i(y))}_{\text{Si } x=y \text{ alors } u_i x = v_i x \text{ (après concaténation des tuiles)}}))$

— $\psi = \exists x (p(a(x), a(x)) \vee p(b(x), b(x)))$

Posons quelques notations :

— $m = m_1 \dots m_n$ un mot sera noté $m(.) = m_1 \dots m_n(.) = m_1(\dots m_n(.) \dots)$ (justification de $u_i(x)$

— $\boxed{\begin{array}{c} x \\ \hline y \end{array}}$ une succession de tuiles qui donne le mot x en haut et y en bas.

Preuve de l'indécidabilité Pour montrer l'indécidabilité, il nous faut montrer deux propriétés : tr est une fonction calculable (lemme 1) et tr est correcte (lemme 2).

Lemme 1. *La fonction tr est calculable.*

Démonstration. La fonction tr s'exécute en temps fini car pour toute instance w du problème POST, il n'y a qu'un nombre fini de tuiles. La formule se calcul alors en temps fini. □

Lemme 2. *Pour toute instance w de* POST, *$tr(w)$ est une formule valide si et seulement si w est ne instance positive.*

Démonstration. $\Rightarrow$ La formule $\varphi \to \psi$ est valide, donc pour tout modèle $\mathcal{M}$, $\mathcal{M} \models (\varphi \to \psi)$.

Choisir le modèle On choisit le modèle $\mathcal{M}$ suivant (possible car la formule est conséquence sémantique de tous modèle).

- $D_{\mathcal{M}} = \Sigma^*$
- $a^{\mathcal{M}}(.) : \begin{array}{ccc} \Sigma^* & \to & \Sigma^* \\ x & \mapsto & xa \end{array}$
- $e^{\mathcal{M}}$ correspond à e_{Σ^*} (constante : pas de tuiles)
- $b^{\mathcal{M}}(.) : \begin{array}{ccc} \Sigma^* & \to & \Sigma^* \\ x & \mapsto & xb \end{array}$
- $p^{\mathcal{M}} = \{(x, y) \mid \boxed{\begin{array}{c} x \\ \hline y \end{array}}\} \cup \{\epsilon, \epsilon\}$

Montrer que $\mathcal{M} \models \varphi$ On a $\mathcal{M} \models p(e, e)$ (par hypothèse, correspond au cas où on n'a pas mis de tuiles sur la table). Montrons que $\mathcal{M} \models \forall x \forall y, (p(x, y) \to p(u_i(x), v_i(y)))$. Comme $\forall u, v \in \Sigma^*$, si $\mathcal{M} \left[\begin{array}{l} x := u \\ y := v \end{array} \right] \models p(x, y)$, alors en interprétant p, $\boxed{\begin{array}{c} u \\ \hline v \end{array}}$ existe. Pour tout $i \in [\![1, N]\!]$, par concaténation de $\boxed{\begin{array}{c} u \\ \hline v \end{array}}$ et de $\boxed{\begin{array}{c} u_i \\ \hline v_i \end{array}}$, alors il existe $\boxed{\begin{array}{c} uu_i \\ \hline vv_i \end{array}}$. Donc $\forall i \in [\![1, N]\!]$, $\mathcal{M} \models \forall x \forall y, (p(x, y) \to p(u_i(x), v_i(y)))$ et $\mathcal{M} \models \varphi$.

Montrer que w est une instance positive du problème POST Comme $\mathcal{M} \models (\varphi \to \psi)$ (la formule est valide) et $\mathcal{M} \models \varphi$, $\mathcal{M} \models \psi$. Sans perte de généralité, on suppose que $\mathcal{M} \models \exists x, p(a(x), a(x))$ (le cas b est similaire). Donc, il existe $\alpha \in \Sigma^*$ tel que $\mathcal{M}[x := \alpha] \models p(a(x), a(x))$. Par la définition de p, la tuile $\boxed{\begin{array}{c} a\alpha \\ \hline a\alpha \end{array}}$ existe. L'instance w est donc positive (car il existe une suite de tuiles telle que le mot de haut et le mot du bas sont le même).

$\Leftarrow$ On suppose que w soit une instance positive du problème POST. Il existe donc $m \geq 0$ tel que $\exists i_1, \ldots, i_m \in [\![1, N]\!]$ avec $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$. Soit $\mathcal{M}$ un modèle tel que $\mathcal{M} \models \varphi$. Montrons que $\mathcal{M} \models \psi$.

Montrons que $\mathcal{M} \left[\begin{array}{l} x := u_{i_1} \ldots u_{i_k} \\ y := v_{i_1} \ldots v_{i_k} \end{array} \right] p(x, y)$ en raisonnant par récurrence sur $k \in \mathbb{N}$.

Initialisation Pour $k = 0$, on a $\mathcal{M} \models p(e, e)$ car $\mathcal{M} \models \varphi$.

Hérédité Soit $k \in \mathbb{N}$ tel que $\mathcal{M} \left[\begin{array}{l} x := u_{i_1} \ldots u_{i_k} \\ y := v_{i_1} \ldots v_{i_k} \end{array} \right] \models p(x, y)$ et montrons le pour $k + 1$.

On sait que $\forall i \in [\![1, N]\!], \mathcal{M} \models \forall x \forall y, (p(x, y) \to p(u_i(x), v_i(y)))$.
En particulier, pour $i = i_{k+1}$, on a $\mathcal{M} \models \forall x \forall y, (p(x, y) \to p(u_{i_{k+1}}(x), v_{i_{k+1}}(y)))$.

Donc, $\mathcal{M} \left[\begin{array}{l} x := u_{i_1} \ldots u_{i_k} \\ y := v_{i_1} \ldots v_{i_k} \end{array} \right] \models (p(x, y) \to p(u_{i_{k+1}}(x), v_{i_{k+1}}(y)))$.

Ainsi, $\mathcal{M} \left[\begin{array}{l} x := u_{i_1} \ldots u_{i_{k+1}} \\ y := v_{i_1} \ldots v_{i_{k+1}} \end{array} \right] \models p(x, y)$.

Conclusion On a $\forall k \in [\![1, m]\!], \mathcal{M} \left[\begin{array}{l} x := u_{i_1} \ldots u_{i_k} \\ y := v_{i_1} \ldots v_{i_k} \end{array} \right] \models p(x, y)$.

Sans perte de généralité, on suppose que $u_{i_1} \ldots u_{i_m}$ finit par un a. On écrit $u_{i_1} \ldots u_{i_m} = \tilde{u}a$ et $v_{i_1} \ldots v_{i_m} = \tilde{v}a$. On veut prouver que $\mathcal{M} \models \psi$ et on a $\mathcal{M} \left[\begin{array}{l} x := u_{i_1} \ldots u_{i_m} \\ y := v_{i_1} \ldots v_{i_m} \end{array} \right] \models p(x, y)$ donc $\mathcal{M} \left[\begin{array}{l} x := \tilde{u}a \\ y := \tilde{v}a \end{array} \right] \models p(x, y)$ d'où $\mathcal{M} \left[\begin{array}{l} x := \tilde{u}a \\ y := \tilde{u}a \end{array} \right] \models p(x, y)$. De fait, il existe $\tilde{u}$ tel que $\mathcal{M}[x := a] \models p(a(x), a(x))$. Donc $\mathcal{M} \models \psi$. Ainsi $\mathcal{M} \models (\varphi \to \psi)$. Ce qui prouve que $(\varphi \to \psi)$ est valide. □

On a donc montre l'indécidabilité du problème VALIDE. □

3 Validité et satisfiabilité

3.1 Modèles en logique du premier ordre

Les formules de la logique du premier ordre ne signifie rien en particulier : il nous faut donc leur donner un sens [1, p.398]. C'est le rôle du modèle qui va alors interpréter les symboles de constantes (à l'aide d'un domaine), les symboles de fonctions (fonctions sur ce domaine) et de prédicat (prédicat de ce domaine dans les valeurs vrai, faux). Contrairement au cas du calcul porpositionnel où le modèle est en réalité une valuation (dont le domaine porte sur $\{\mathsf{vrai}, \mathsf{faux}\}$), le domaine d'un modèle en logique du premier ordre peut contenir un élément comme une infinité.

Définition. Une signature Σ est un ensemble dénombrable de symbole de fonctions et de prédicats munis de leurs arités (dans ce cas, les fonctions d'arité 0 sont les constantes de notre modèle, sinon il faut les rajouter à la signature).

Définition. Une Σ-structure (ou un Σ-modèle) est la donnée de $\mathcal{M} = \langle M, .^{\mathcal{M}} \rangle$ où

— M est un ensemble non-vide appelé domaine.
— $.^{\mathcal{M}}$ est une fonction appelée interprétation telle que :
  — à tout symbole de fonction f d'arité n dans Σ associe la fonction $f : M^n \to M$;
  — à tout symbole de prédicat p d'arité n dans Σ associe le prédicat $p : M^n \to \{0, 1\}$.

Dans le cas où on sépare les symboles de fonction et de constante, la fonction d'interprétation doit également interpréter ces symboles de constantes.

Remarque. Le domaine d'un modèle est supposé non vide [3, p.68]. Cette hypothèse est nécessaire (même si un modèle avec un domaine non vide est inintéressant) pour le théorème de complétude. En effet, si on veut qu'une formule démontrée soit vraie dans tout modèle, il faut (par la règle du $\exists_i$) que le domaine soit non vide.

3.2 Sémantique de la logique du premier ordre

Nous allons maintenant étudier comment on donne un sens à nos formules à l'aide d'un modèle [1, p.408]. Soit φ une formule de la logique du premier ordre, $\mathcal{M}$ un modèle pour cette logique et $\mathcal{V}$ un ensemble de dénombrable de variables (que nous utilisons dans notre formule) sur notre domaine.

Évaluation sous forme classique L'évaluation permet de faire un pont entre la syntaxe et la sémantique. Elle se définit naturellement de manière inductive sur la hauteur de notre formule. Les variables ne sont pas comme les constantes, si elle ne sont pas liée par un quantificateur, elles sont libre comme l'air. Dans la suite, on utilisera des formules closes (toutes les variables sont liées) mais nous devons définir l'évaluation en présence de variables libres car toutes sous formules d'une formule close ne sont pas nécessairement closes.

Définition. Une assignation des variable v est une fonction de $\mathcal{V} \to M$.

Définition. Une assignation des termes est définie par induction comme suit :

— $v^{\mathcal{M}}(x) = v(x)$ si $x \in \mathcal{V}$;

— $v^{\mathcal{M}}(f(t_1, \ldots, t_n)) = f^{\mathcal{M}}\left(v^{\mathcal{M}}(t_1), \ldots, v^{\mathcal{M}}(t_n)\right)$ sinon.

Définition. On définit (par induction sur la hauteur de la formule) $\mathcal{M}, v \models \varphi$ qui signifie que φ est vraie dans le modèle $\mathcal{M}$ sous l'assignation des variables v (que nous appelons également condition de vérités).

— $\mathcal{M}, v \models p(t_1, \ldots, t_n)$ si $p^{\mathcal{M}}\left(v^{\mathcal{M}}(t_1), \ldots, v^{\mathcal{M}}(t_n)\right) = 1$

— $\mathcal{M}, v \models \neg\varphi$ si $\mathcal{M}, v \not\models \varphi$

— $\mathcal{M}, v \models (\varphi \vee \psi)$ si $\mathcal{M}, v \models \varphi$ ou $\mathcal{M}, v \models \psi$

— $\mathcal{M}, v \models (\varphi\wedge)$ si $\mathcal{M}, v \models \varphi$ et $\mathcal{M}, v \models \psi$

— $\mathcal{M}, v \models \exists x\varphi$ s'il existe $m \in D$ tel que $\mathcal{M}, v[x := m] \models \varphi$

— $\mathcal{M}, v \models \forall x\varphi$ pour tout $m \in D$ tel que $\mathcal{M}, v[x := m] \models \varphi$

Remarque. Dans certain ouvrage, une Σ-structure est appelée modèle que si cette structure est telles que $\mathcal{M}, v \models \varphi$.

Définition. Deux formules de la logique du premier ordre φ et ψ sont dites équivalentes si et seulement si pour tout modèle $\mathcal{M}$, ($\mathcal{M}, v \models \varphi$ si et seulement si $\mathcal{M}, v \models \psi$). On note alors $\varphi \equiv \psi$.

Évaluation par le jeu Évaluer une formule peut se présenter comme un jeu à deux joueurs : le premier cherchant à montrer que la formule est satisfiable et le second à prouver que ce n'est pas le cas. Pour définir ce jeu, on suppose que les seuls connecteurs logique que nous disposons sont $\neg, \vee, \wedge$.

Définition. Soit φ une formule du premier ordre et $\mathcal{M}$ un modèle. Le jeu d'évaluation $\mathbb{E}v(\mathcal{M}, \varphi)$ est défini comme suit :

1. Il comprend deux joueurs : Vérifieur (prouve que $\mathcal{M}, v \models \varphi$) et le Falsificateur (prouve que $\mathcal{M}, v \not\models \varphi$). Les coûts des joueurs consiste à choisir des sous-formules via les règles suivantes :
   $\varphi_0 \vee \varphi_1$ V choisit $j \in \{0, 1\}$ et le jeu continue avec φ_j.
   $\varphi_0 \wedge \varphi_1$ F choisit $j \in \{0, 1\}$ et le jeu continue avec φ_j.
   $\neg\varphi$ V et F échange leur rôle et le jeu continue avec φ.
   $\exists x\varphi$ V choisit $a \in M$ et le jeu continue avec $v[x := a]$ et φ.
   $\forall x\varphi$ F choisit $a \in M$ et le jeu continue avec $v[x := a]$ et φ.
   $R(t_1, \ldots, t_n)$ le jeu s'arrête et V gagne si et seulement si $p^{\mathcal{M}}\left(v^{\mathcal{M}}(t_1), \ldots, v^{\mathcal{M}}(t_n)\right) = 1$
2. La condition de victoire n'apparaît que si le jeu s'arête. Il s'arête quand toutes les variables ont été assignées (si la formule est close) et que nous obtenons une formule atomique.

On peut maintenant se demander si le Vérifieur a une stratégie gagnante dans notre jeu. En déroulant la partie sous la forme d'un arbre, on voit que cette stratégie (si elle existe) peut être calculant en remontant l'arbre à partir d'une solution gagnante pour celui-ci.

Théorème. *$\mathcal{M}, v \models \varphi$ si et seulement si V a une stratégie gagnante dans $\mathbb{E}v(\mathcal{M}, \varphi)$.*

Idée de la preuve. Se fait par induction sur la hauteur de la formule pour montrer que si $\mathcal{M}, v \models \varphi$ alors V a une stratégie gagnante dans $\mathbb{E}v(\mathcal{M}, \varphi)$ et si si $\mathcal{M}, v \models \neg\varphi$ alors F a une stratégie gagnante dans $\mathbb{E}v(\mathcal{M}, \varphi)$. □

Remarque. Le jeu $\mathbb{E}v(\mathcal{M}, \varphi)$ est un jeu fini (la formule est de taille finie) mais l'arbre du jeu peut être infini (si le domaine est infini).

3.3 Les problèmes de décision sur les logiques

Nous allons maintenant présenter trois problème de décision légitime lorsque nous étudions une logique : le problème de la satisfiabilité, de la validité et du model checking. Nous rappelons également leur complexité (si elle existe).

Définition. Une formule (close) φ est satisfiable s'il existe un modèle $\mathcal{M}$ tel que $\mathcal{M}, v \models \varphi$.

Définition. Une formule (close) φ est valide (ou tautologique) si pour tout modèle $\mathcal{M}$, $\mathcal{M}, v \models \varphi$.

Définition. On définit le problème SAT sur les formules d'une logique.
Problème : SAT
entrée : ϕ une formule (close)
sortie : Oui si ϕ est satisfiable ; non sinon

Définition. On définit le problème VALIDE sur les formules d'une logique.
Problème : VALIDE
entrée : ϕ une formule (close)
sortie : Oui si ϕ est valide ; non sinon

Définition. On définit le problème MODEL-CHECKING sur les formules d'une logique.
Problème : MODEL-CHECKING
entrée : ϕ une formule (close), $\mathcal{M}$ un modèle fini
sortie : Oui si $\mathcal{M} \models \phi$; non sinon

Présentons quelques résultats sur ces problèmes de décisions.

Problème	Propositionnelle	FO
SAT	NP-complet (Cook)	Indécidable
VALIDE	NP-complet	Indécidable
MODEL-CHECKING	P	PSPACE-complet

4 Technique de 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 2) 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. □

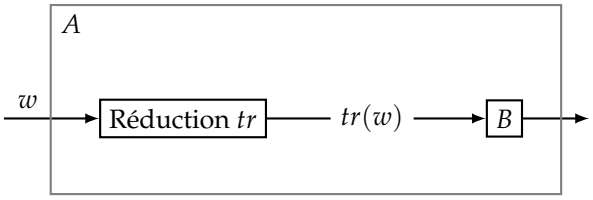

FIGURE 2 – Schéma du principe de réduction du problème A au problème B.

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.*

5 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 [4]). *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 $\langle$PARADOXE, $\langle$PARADOXE$\rangle\rangle$ 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 [4])**.** *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] J. Duparc. *La logique pas à pas*. Presse polytechnicienne et université romandes, 2015.

[2] Michael Huth and Mark Ryan. *Logic in Computer Science : Modelling and Reasoning About Systems*. Cambridge University Press, New York, NY, USA, 2004.

[3] Christophe Raffalli, René David, and Karim Nour. *Introduction à la Logique, Théorie de la démonstration (2nd édition)*. Sciences Sup. Dunod, 2004.

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