Table des matières

Leçon 1

Décidabilité et indécidabilité. Exemples

Auteur·e·s: Emile Martinez
Niveau : MPI
Pré-requis : Langages
Références :

1 Cadre théorique

I Problèmes de décision

Définition 1.1: Un problème de décision Π est la donnée d'un ensemble E d'instance et d'un ensemble $E^+ \subset E$ d'instances positives.

Exemple 1.2:

- $E = \mathbb{N}$, $E^+ = \{n \in \mathbb{N} / n \text{ est premier}\}$
- $E = \{\text{formules du premier ordre}\}$, $E^+ = \{\text{formules universellement vraies}\}$

notation 1.3: Pour Π un problème de décision (E, E^+), on identife parfois Π et E^+

Exemple 1.4: $SAT = \Big(\{\text{formules propositionnelles, } \{\text{formules propositionnelles satisfiables}\}\}\Big)$. On peut alors noter $x \wedge y \in SAT$ mais $x \wedge \neg x \notin SAT$.

2 Modèle de calcul

Définition 1.5: Un modèle de calcul est la donnée de :

— un ensemble de ≪machines≫
— le comportement d'une machine sur une entrée (comment elle ≪s'exécute≫)
— si l'exécution renvoie une réponse, quelle est-elle ?

Remarque 1.6: Cette définition est volontairement large et flou pour pouvoir accueillir un grand nombre de concept différents

Remarque 1.7: L'exécution peut mener à un arrêt prématuré (ou erreur) ou ne jamais terminer

Exemple 1.8: Les automates finis déterministes, qui prennent en entrée $w \in \Sigma^*$ et répondent un booléen ($w \in \mathcal{L}(A)$)

Définition 1.9: Une machine résout un problème si pour toute instance du problème, la machine retourne une sortie correspondant à l'entrée.

Exemple 1.10: Un automate $\mathcal{A}$ sur $\Sigma = \{0,1\}^3$ qui détermine si la somme de deux entiers binaires donne bien le troisième.
$\mathcal{L}(A) = \{w_1w_2w_3 / w_i \in \{0,1\}^3 \text{ et } w_1 + w_2 = w_3\}$

Remarque 1.11: Certains problèmes de décision ne sont pas résolvables par des automates déterministes : l'appartenance d'un mot à $\{a^n b^n / n \in N\}$

Définition 1.12: Autres modèles de calcul : les grammaires algébriques, les fonctions OCamL, les automates infinis

Exercice 1.13: Classer ces modèles par puissance de calcul

3 Notions de calculabilité et décidabilité

I Sérialisation

Définition 1.14: 0n appelle sérialisation le processus consistant à transformer une donnée en chaînes de caractères, sans perte d'informations.

Remarque 1.15: Sans perte d'information veut dire que la fonction de conversion est injective.

Exemple 1.16: Pour un arbre binaire, la chaîne de caractère composée de son parcours préfixe, en ajoutant un symbole pour les feuilles.

Syntaxe 1.17: On notera $< D >$ la sérialisation d'une donnée D

Principe 1.18: Toutes les données manipulés par un ordinateur sont sérialisables

Démonstration. La donnée est stockée dans des cases mémoires, que l'on peut interpréter comme des caractères. On concatène alors tous les couples (adresse, caractère) (couple qui prend un nombre fixe de caractères) des cases concernées. □

Exemple 1.19: Pour stocker un couple de chaines de caratères, on peut stocker la taille de la premiere en caractère, puis #, puis la premiere chaine sans le caractère de fin de chaine, puis la seconde.

On se limitera donc ici à E comme un ensemble de chaine de caractère.

II Décidabilité

On se fixe alors comme modèle de calcul, les fonctions OCaml avec une mémoire infinie. Notons $\mathcal{F}$ l'ensemble de ces fonctions, que l'on appellera machine.

Remarque 1.20: Avec mémoire infinie revient en OCamL que l'on a jamais de débordement de pile, et que l'on a toujours de la place pour créer de nouvelles données. Par contre le code des fonctions doit être fini.

Remarque 1.21: Des cadres théoriques plus formels existent (comme les machines de Turing)

Définition 1.22 (décidable): On dit alors qu'un problème (E, E^+) est décidable s'il existe une machine de type : `string` $\to$ `bool` terminant toujours et renvoyant `true` sur E^+ et `false` sur $E \backslash E^+$.

Principe 1.23 (Thèse de Church): La notion de décidabilité ne dépend pas du modèle de calcul, pour peu qu'il soit raisonnable.

Remarque 1.24: ≪Raisonnable≫

Commentaire 1.1 *Ici, les modèles de calcul ne prennet pas forcément en entrée les mêmes types de données, mais il suffit qu'ils implement les chaines de caractères (qu'on puisse les injecter dans le modèle) pour avoir une notion qui a du sens (pour pouvoir comparer des modèles entre eux qui n'ont à priori rien à voir).*

Commentaire 1.2 *On peut insister ici sur le fait que ce n'est qu'une thèse, et que ca dépend de ce qu'on entend par raisonnable, mais que néanmoins, cela semble quelque chose de très puissant et très clairement non trivial qui se vérifie toujours.*

Idée 1.25: Tous les langages de programmation ont la même puissance de calcul théoriques (peuvent calculer les mêmes choses)

Commentaire 1.3 *Ici le théorique vient car c'est vrai dans des modèles à mémoire infini, etc.*

Équivalence entre C sans fonction et OCaml sans impératif

Corollaire 1.26: On s'autorisera à écrire du pseudo-code et à décrire les programmes.

Commentaire 1.4 *Ce corollaire est important, et donne une raison supplémentaire de parler de la thèse de Church Turing, et de justifier pourquoi on s'abstrait aussi vite du modèle de calcul que l'on a décrit.*

III Calculabilité

On considère des ensembles A et B sérialisables (et dont on peut vérifier qu'une chaine de caractère est un codage valide)

Définition 1.27: On dit qu'une fonction $f : A \to B$ est calculable si il existe une machine g : `string` $\to$ `string` tel que $\forall x \in A, g$ termine sur $< x >$ et $g < x >=< f(x) >$

Principe 1.28: Les notions de calculabilité et de décidabilité sont équivalentes.

Démonstration. $\boxed{\Rightarrow}$ Soit $\Pi = (E, E^+)$. Si $f_\Pi : \begin{array}{ccc} E & \to & \mathbb{B} \\ e & \mapsto & \texttt{true ssi } e \in E^+ \end{array}$ est calculable, alors Π est décidable

$\boxed{\Rightarrow}$ Soit $f : A \to B$. On pose $\Pi_f = (E_f, E_f^+)$ où $E_f =< A \times B >$ et $E_f^+ = \{< x, f(x) > / x \in A\}$. Alors si Π_f est décidable, f est décidable. □

Propriété 1.29: $\mathcal{F}(\Sigma^*, \Sigma^*)$ est indénombrable.

Corollaire 1.30: Il existe des fonctions non calculables et donc des problèmes indécidables.

Démonstration. Les fonctions OCaml sont dénombrables et donc les fonctions calculables aussi (car on a plus de fonctions OCaml que de fonctions calculables). □

Exemple 1.31: Numérotons $f_0, f_1, \ldots$ les fonctions calculables de $\mathbb{N}$ dans $\mathbb{N}$. On créer alors la fonction $f(n) = f_n(n) + 1$. (On peut les numéroter car il y en a un nombre dénombrables, car il y a)

4 Exemples

La plupart des problèmes que vous connaissez sont décidables. Dès qu'il existe un algorithme le décidant, peu importe sa complexité, le problème est décidable.

Exemple 1.32: satisfiabilité d'une formule propositionnnelle, coloration d'un graphe, plus grande clique dans un graphe, présence d'un élément dans une liste, etc.

I Sur les programmes

Définition 1.33 (Problème de l'arrêt): Le problème de l'arrêt est le problème de savoir si une machine terminera sur une entrée.

Formalisation : $E = \Sigma^*$, $\Sigma^+ = \{< f, w > / f \in \mathcal{F}, w \in \Sigma^*, f$ termine sur $w\}$

Théorème 1.34: Le problème de l'arrêt est indécidable.

Démonstration. Par l'absurde. Soit `A` décidant l'arrêt. Soit `M` le programme suivant :

```
let M x = if A x x then (while true do () done; true)
                   else true;;
```

On obtient alors une contradiction en regardant la terminaison de `M <(M, <M>)>` □

Théorème 1.35 (Théorème de Rice): Soit $\mathcal{P}$ une propriété non triviale sur les langages. Alors savoir si le langage reconnu par une machine f vérifie $\mathcal{P}$ est indécidable

Exemple 1.36: Savoir si f ε renvoie true est indécidable.

Remarque 1.37: Même la correction partielle est indécidable

Développement : Preuve du théorème de Rice et de l'indécidabilité de la correction partielle.

Propriété 1.38: Le problème $E^+ = \{< f, n, w > / f$ termine sur w en moins de n étapes$\}$ est décidable.

Propriété 1.39: Le problème $E^+ = \{< L, n > / L$ est un langage finit tel que il existe une machine décidant L terminant toujours en moins de n étapes$\}$ est décidable.

Démonstration. Il suffit de tester tous les n premières étapes possibles pour un programme (y en a beaucoup, mais un nombre fini) et donnant toutes les valeurs possibles aux cases visitées, et voir si cela décide L. □

Remarque 1.40: Avec cette preuve on se rend bien compte que la complexité ne rentre pas en compte dans la décidabilité.

II Sur les langages

Décidable	Indécidable
• Un automate fini reconnaît un mot donné • Deux automates reconnaissent le même langage • Deux expressions régulières sont égales • Une grammaire accepte un mot donné	• Une grammaire ne reconnaît aucun mot • Une grammaire reconnait Σ^* • Deux grammaires ont le même langage • Une grammaire est ambigüe

III Autres domaines

Propriété 1.41: Déterminer si une théorie prouve un formule du premier ordre en déduction naturelle $(T \vdash F)$ est indécidable.

Propriété 1.42: Déterminer si un jeu de tuiles fini pave le plan ($\mathbb{Z} \times \mathbb{Z}$) est indécidable

Remarque 1.43: Certains sous problèmes sont décidables :

- Déterminer si une théorie prouve une formule propositionnelle
- Paver $\{1, \ldots, n\} \times \{1, \ldots, n\}$
- Déterminer si une théorie complète (toute formule y est soit vraie soit fausse) prouve une formule du premier ordre en logique propositionnelle.

Leçon 2

Formules du calcul propositionnel : représentation, formes normales, satisfiabilité. Applications.