Leçon 927 : Exemples de preuves d'algorithmes : correction et terminaison

Julie Parreaux

2018 - 2019

Références pour la leçon

[1] Carton, *Langages formels, calculabilité et complexité.*
[2] Cormen, *Algorithmique.*
[3] Nielson et Nielson, *Semantics with applications.*

Développements de la leçon

Correction de l'algorithme de Dijkstra — Complétude de la logique de Hoare

Plan de la leçon

Introduction **2**

1 Terminaison **2**
1.1 Ensemble bien fondé . . . 3
1.2 Terminaison des algorithmes récursifs . . . 3
1.3 Terminaison des algorithmes itératifs . . . 3

2 Correction **3**
2.1 Correction des algorithmes récursifs . . . 3
2.2 Correction des algorithmes itératifs . . . 4
2.3 Programmer c'est prouver . . . 4

3 Automatisation **4**
3.1 Langage IMP [3, p.7] . . . 5
3.2 Sémantique naturelle [3, p.20] . . . 5
3.3 Logique de Hoare [3, p.213] . . . 5

Motivation

Défense

Lors de l'écriture d'un programme, savoir s'il termine et s'il est correcte est une question légitime et difficile. C'est pourtant l'essence de l'informatique : l'étude de ces objets (comme

les nombres pour les mathématiques). En effet, par les théorèmes d'indécidabilité du problème de l'arrêt et de Rice nous savons que répondre à ces questions est indécidable. Leur automatisation ne peut être complète et pour la plus part des problèmes nous devons prouver au moins une partie de ces résultats à la main : ce qui peut être long, fastidieux et même délicat.

Ce qu'en dit le jury

Le jury attend du candidat qu'il traite des exemples d'algorithmes récursifs et des exemples d'algorithmes itératifs.

En particulier, le candidat doit présenter des exemples mettant en évidence l'intérêt de la notion d'invariant pour la correction partielle et celle de variant pour la terminaison des segments itératifs.

Une formalisation comme la logique de Hoare pourra utilement être introduite dans cette leçon, à condition toutefois que le candidat en maîtrise le langage. Des exemples non triviaux de correction d'algorithmes seront proposés. Un exemple de raisonnement type pour prouver la correction des algorithmes gloutons pourra éventuellement faire l'objet d'un développement.

Introduction

Motivation

- Étude des programmes fait l'essence de la science informatique.
- Savoir si un programme termine ou s'il est correct est une notion importante.
- Indécidabilité de l'automatisation.
- Automatisation de la vérification : vérifier pas simple (souvent) long et fastidieux.
- Pourtant programmer c'est prouver (correspondance de Curry–Howard)
- *Définition* : Spécification
- *Remarque* :Le plus difficile à donner

1 Terminaison

Le théorème de l'arrêt et son indécidabilité pose les limites de l'algorithmique : on ne peut obtenir un algorithme qui permettent de prouver la terminaison de tous les algorithmes. On étudie alors quelques techniques que nous utilisons à la main. Gardons en tête la preuve des programmes parallèles.

- *Définition* : Terminaison d'un algorithme
- *Remarque* : Boucle POUR préserve la terminaison
- *Exemple* : Terminaison et boucle TANT QUE
- *Définition* : Problème ARRÊT
- *Théorème* : ARRÊT est indécidable
- *Exemple* : Syracuse
- *Conséquence* : Pas de procédure effective mais heuristique pour certaine classe

1.1 Ensemble bien fondé

La notion d'ensemble bien fondé est un outil puissant pour montrer la terminaison.

— *Définition* : Ensemble bien fondé
— *Exemple* : $\mathbb{N}$
— *Contre-exemple* : $\mathbb{Z}$
— *Proposition* : $\mathbb{N}^2$ muni de l'ordre lexicographique est bien fondé
— *Proposition* : Caractérisation de l'ordre lexicographique

1.2 Terminaison des algorithmes récursifs

Dans le cas des algorithmes récursifs, l'ensemble bien fondé suffit pour montrer la terminaison de tels algorithmes.

— *Théorème* : Terminaison des algorithmes récursifs
— *Exemple* : Calcul du pgcd via l'algorithme d'Euclide (ordre bien fondé sur le premier argument)
— *Exemple* : Fonction d'Ackermann

1.3 Terminaison des algorithmes itératifs

Dans le cas des algorithmes itératifs, la notion d'ensemble bien fondé ne s'applique pas directement : on utilise la notion d'invariant de boucle.

— *Définition* : Variant de boucle
— *Théorème* : Terminaison des algorithmes itératifs
— *Exemple* : Division euclidienne

2 Correction

Le théorème de Rice et son résultat d'indécidabilité pose les limites de l'algorithmique : on ne peut obtenir un algorithme qui permettent de prouver la correction de tous les algorithmes. On étudie alors quelques techniques que nous utilisons à la main. De plus, via la correspondance de Curry–Howard, on remarque que programmer c'est prouver et donc programmer c'est prouver la correction.

— *Idée* : Le programme (algorithme) vérifie se spécification
— *Définition* : Correction (totale et partielle)
— *Exemple* : Syracuse est partiellement correct
— *Exemple* : Tri topologique est totalement correct

2.1 Correction des algorithmes récursifs

Commençons par l'étude de la correction des algorithmes récursifs.

— *Proposition* : Vérification par les prédicats
— *Théorème* : Correction des algorithmes récursifs
— *Exemple* : l'algorithme du pgcd est correct
— *Exemple* : Correction du tri par insertion

2.2 Correction des algorithmes itératifs

Pour les algorithmes itératifs, nous pouvons utiliser la notion d'invariant de boucle. Cet invariant permet de prouver la correction des boucles : ce sont des étapes vers la correction de l'algorithme.

- *Définition* : Invariant de boucle
- *Théorème* : Utilisation des invariants de boucles
- *Exemple* : Correction de Dijkstra DEV

2.3 Programmer c'est prouver

La correspondance de Curry–Howard, notamment, nous dit que programmer c'est prouver et que cette preuve est en réalité la correction d'un énoncé logique (comme une spécification peut être vue comme une formule logique, on prouve le programme).

Preuve avant l'algorithme Dans certains cas la preuve de correction d'un algorithme vient par l'analyse de ce problème et de ces propriétés. Plus exactement, on en tire un algorithme. Dans d'autres cas, les structures de données que nous utilisons donne la correction de l'algorithme.

- *Méthode* : Analyse du problème
- *Application* : Programmation dynamique
- *Exemple* : Alignement optimaux
- *Méthode* : Structure de données
- *Exemple* : Tri par tas

La correspondance de Curry–Howard Curry et Howard ont prouvé grâce au λ-calcul simplement typé et la logique intuitionniste, ils prouvent que programmer c'est prouver. Ici, on ne évoque simplement quelques faits culturels autours de ce paradigme.

- *Théorème* : Correspondance de Curry–Howard
  - type = proposition (spécification)
  - terme = démonstration (programme)
  - réduction = normalisation (exécution)
- *Application* : Requête SQL dans une base de données (une requête peut être vue comme une formule de la logique du premier ordre et donc leur évaluation sur une base est donc l'exécution de sa preuve de satisfiabilité).

3 Automatisation

Même si l'automatisation de telles preuves est indécidable, nous souhaitons en automatiser une partie. Pour la correction la logique de Hoare s'avère être un outils puissant. Nous allons la définir pour un langage jouet IMP muni de sa sémantique naturelle (nous savons que la sémantique à petits pas et la sémantique dénotationnelles sont équivalentes à celle-ci pour ce langage). Nous pouvons donc choisir n'importe laquelle mais la naturelle est la plus simple pour faire ce que l'on souhaite.

3.1 Langage IMP [3, p.7]

— *Définition* : Langage IMP
— *Exemple* : Factorielle

3.2 Sémantique naturelle [3, p.20]

— *Définition* : Règle de la sémantique naturelle
— *Théorème* : Cette sémantique est déterministe
— *Définition* : Fonction déterministe
— *Exemple* : Arbre de dérivation

3.3 Logique de Hoare [3, p.213]

— *Définition* : Triplet de Hoare
— *Définition* : Règle de la logique de Hoare
— *Définition* : Conséquence sémantique et conséquence de la preuve
— *Théorème* : Correction de la logique
— *Définition* : wlp
— *Théorème* : Complétude de la logique DEV
— *Exemple* :Factorielle par la logique de Hoare
— *Remarque* : Rôle du wlp dans l'automatisation

Références

[1] O. Carton. *Langages formels, calculabilité et complexité*. Vuibert, 2008.
[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Algorithmique, 3ème édition*. Dunod, 2010.
[3] H. R. Nielson and F. Nielson. *Semantics with application*. Springer, 2007.