Leçon 912 - Fonctions récursives primitives et non primitives. Exemples

15 juillet 2019

1 Extraits du Rapport

Rapport de jury 2018

Il s'agit de présenter un modèle de calcul : les fonctions récursives. S'il est bien sûr important de faire le lien avec d'autres modèles de calcul, par exemple les machines de Turing, la leçon doit traiter des spécificités de l'approche. Le candidat doit motiver l'intérêt de ces classes de fonctions sur les entiers et pourra aborder la hiérarchie des fonctions récursives primitives. Enfin, la variété des exemples proposés sera appréciée.

2 Cœur de la leçon

— Des **exemples**, partout, pour tout. (somme finie, produit fini, factorielle, conditionnelles...)

— Définitions des fonctions primitives récursives de bases, primitives récursives et non-primitives récursives (μ-récursives, i.e avec minimisation non bornée).

— Prédicats primitifs et fonctions caractéristiques. Minimisation bornée.

3 À savoir

— Fonction d'Ackermann.

— Lien avec les machines de Turing, ensembles récursifs, récursivement énumérables et fonctions partielles.

4 Ouvertures possibles

— Lien avec le λ-calcul.

— Hiérarchie de Grzegorczyk (si vous savez le motiver de manière convaincante)

— Théorème d'itération (s_{mn}), borderline.

5 Conseils au candidat

— Bien réfléchir et insister sur les motivations.

— Mettre des exemples intéressants.

— Faire des liens avec les constructions des langages de programmation.

— Ne pas définir les fonctions récursives primitives autrement que à partir de celle de bases.

— Ne pas se tromper sur la définition de la minimisation. Le bug standard est de seulement dire que sur f, le but est de retourner le premier n tel que $f(n) \neq 0$. Avec cette définition, on peux résoudre le problème de l'arrêt... Il faut aussi préciser que l'on renvoi ce n uniquement si la fonction répond (et donc termine) sur les entrées précédentes.

— Utiliser des exemples pour souligner le pouvoir expressif, et son évolution.

6 Questions classiques

— Pourquoi considérer les fonctions récursive primitives ?

— Différence entre la récursion définit ici et celle des langages de programmation.

— Quelle construction capture la boucle FOR ? La boucle WHILE ?

— L'encodage d'un problème est-il toujours calculable ? Pourquoi sont définis ainsi les fonctions récursives primitives de bases ?

— Exemple d'argument diagonal.

— Peut-on décider si une fonction récursive est primitive ?

— Montrer que telle fonction est primitive récursive.

— Quelles est la classe de machines de TURING équivalent aux fonctions récursives primitives ?

— Pour tous problèmes ayant une complexité raisonnable, est-ce que le FOR suffit pour les décider ?

— Pouvez-vous calculer Ack(1,x), Ack(2,x) et Ack(3,x) ?

— À quoi sert la fonction d'ACKERMANN ? Quelle partie de sa construction la rend non primitive ?

— Connaissez-vous les fonctions récursives élémentaires ? La classe de complexité associée ? Est-ce aussi expressif que les primitives récursives ? Savez-vous définir la minimisation borné à partir de produit, différence et somme bornée ?

7 Références

— [*Wol*] , Introduction à la calculabilité : cours et exercices corrigés - WOLPER - à la BU/LSV
Appréciable pour sa pédagogie, et la compréhension des concepts majeurs.

— [*Car*] Langages formels, calculabilité et complexité - CARTON - à la BU/LSV
Très bonne référence couvrant beaucoup de bases. Se méfier de certaines preuves faites un peu rapidement.

8 Dev

++ Montre que X est ou n'est pas récursive primitive - ([Cori],[Car],) - 912
X pour ACKERMANN, avec une preuve un peu compliqué dans le Cori, ou X = isPrime dans le Carton. Attention, ACKERMANN est casse-geule, il y a plein de lemmes auxilliaires souvent oubliés (e.g, monotonicté de la fonction considérée).

\+ Une fonction TURING calculable est μ-recursive. - ([Wol],) - 912,913
Preuve précise mais non pédagogique dans le Cori, claire mais non précise dans le Wolper...

\+ Equivalence entre fonctions récursives et fonctions représentables - ([Kri], *p. 24*) - 912,929
Il suffit de faire le sens recursif est représentable, mais il faut savoir justifier facilement de l'autre direction. Être bien au clair sur le problème des fonctions partielles et des problèmes de terminaison. Savoir exprimer le lambda terme correspondant à une fonction recursive, e.g l'addition, à partir du résultat.