École polytechnique — X2013 — INF412 Fondements de l'informatique

Fonctions Récursives Primitives

Sujet proposé par Bruno Salvy

Les fonctions récursives primitives ont été introduites par Gödel dans son travail sur l'incomplétude. Elles permettent de décrire des fonctions dont il est clair que le calcul termine toujours. Elles correspondent ainsi aux fonctions qui peuvent être calculées sans l'instruction "while" dans un langage typé comme Pascal, c'est-à-dire avec des "if-then-else" et des "for i :=1 to n". L'objectif de cette petite classe est de se convaincre de cette expressivité et d'observer une partie de ses limitations.

Informellement, ces fonctions sont celles que l'on peut définir sur l'ensemble $\mathbb{N}$ des entiers naturels par récurrence. L'ensemble de ces fonctions est défini par induction.

Définition. Une fonction $f : \mathbb{N}^n \to \mathbb{N}$ est *récursive primitive* si elle est soit la constante 0 (alors $n = 0$), soit l'une des fonctions :

- $\mathsf{Zero} : x \mapsto 0$ la fonction 0 (alors $n = 1$) ;
- $\mathsf{Succ} : x \mapsto x + 1$ la fonction successeur (alors $n = 1$) ;
- $\mathsf{Proj}_n^i : (x_1, \ldots, x_n) \mapsto x_i$ les fonctions de projection, pour $1 \leq i \leq n$;
- $\mathsf{Comp}_n(g, h_1, \ldots, h_m) : (x_1, \ldots, x_n) \mapsto g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$ la composition des fonctions récursives primitives $g, h_1, \ldots, h_m$;
- $\mathsf{Rec}(g, h)$ la fonction définie par récurrence comme

$$\begin{cases} f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n), \\ f(x_1 + 1, x_2, \ldots, x_n) = h(f(x_1, \ldots, x_n), x_1, \ldots, x_n), \end{cases}$$

  où g et h sont récursives primitives.

1 Puissance de calcul

Ces règles simples permettent de définir des fonctions nouvelles. Par exemple, on peut définir une fonction Pred (prédecesseur) par la règle de récurrence :

$$\mathsf{Pred}(0) = 0, \quad \mathsf{Pred}(x + 1) = x,$$

ce qui s'exprime plus formellement par $\mathsf{Pred} = \mathsf{Rec}(0, \mathsf{Proj}_2^2)$. (Noter qu'on emploie ici la *constante* 0 et non la fonction Zero).

1.1 Premiers exemples

Question 1.1. *Montrer que l'addition $+ : \mathbb{N}^2 \to \mathbb{N}$ est récursive primitive. Pour ce premier exemple, bien détailler les règles employées en explicitant les fonctions $g, h, h_1, \ldots, h_m$ éventuellement en jeu.*

Question 1.2. *Montrer que la multiplication et la puissance sont récursives primitives.*

1.2 If-Then-Else

Les prédicats, vus comme des fonctions à valeurs dans $\{0,1\}$, rentrent naturellement dans le cadre des objets que l'on souhaite voir un programme manipuler. Cependant, au-delà des exemples ci-dessus, il faut souvent un peu d'astuce pour définir des opérations, et même pour la simple égalité. Les deux questions qui suivent vont montrer que si $f(x_1,\dots,x_n)$ et $g(x_1,\dots,x_n)$ sont des fonctions récursives primitives, alors les prédicats

$$f(x_1,\dots,x_n) < g(x_1,\dots,x_n), \quad f(x_1,\dots,x_n) \leq g(x_1,\dots,x_n), \quad f(x_1,\dots,x_n) = g(x_1,\dots,x_n) \tag{1}$$

le sont aussi. Une fois ceci établi, les questions suivantes montreront comment les fonctions définies par des programmes contenant des if-then-else ou des boucles simples sont aussi récursives primitives.

Question 1.3. *Montrer que la soustraction tronquée, définie par*

$$x \dot{-} y = \begin{cases} x - y & \text{si } x \geq y, \\ 0 & \text{si } x < y, \end{cases}$$

est récursive primitive.

Question 1.4. *Montrer que les prédicats $x < y$, $x \leq y$ et $x = y$ sont récursifs primitifs. Conclure pour les prédicats de (1). [Indication : commencer par écrire une fonction* IsZero *qui teste si son argument est nul ou non.]*

Question 1.5. *Soient R et S des prédicats récursifs primitifs. Montrer que le sont aussi les prédicats :*

$$\neg R, \quad R \wedge S, \quad R \vee S.$$

La programmation par if-then-else est capturée par la question suivante :

Question 1.6. *Soient f, g et R deux fonctions et un prédicat, tous trois récursifs primitifs à k arguments, et soit h la fonction définie par*

$$h(x_1,\dots,x_k) = \begin{cases} f(x_1,\dots,x_k) & \text{si } R(x_1,\dots,x_k) = 1, \\ g(x_1,\dots,x_k) & \text{sinon.} \end{cases}$$

Montrer que h est récursive primitive.

1.3 Boucles simples

La question suivante montre qu'une certaine forme de "boucle for" est également possible.

Question 1.7. *Si $g(x,y)$ est récursive primitive, alors le sont aussi*

$$S_g(z,y) = \sum_{x=0}^{z} g(x,y), \qquad P_g(z,y) = \prod_{x=0}^{z} g(x,y).$$

Question 1.8. *Si R est un prédicat récursif primitif, alors le sont aussi*

$$\exists x \leq z, R(x,y) \quad \text{et} \quad \forall x \leq z, R(x,y).$$

Question 1.9. *Montrer que le prédicat "x est un nombre premier" est récursif primitif.*

1.4 Exploration bornée

Soit $R(y, x_1, \ldots, x_n)$ un prédicat. On définit un opérateur Min de "minimisation bornée" par :

$$\min_{y=0}^{x} R(y, x_1, \ldots, x_n) : (x, x_1, \ldots, x_n) \mapsto \begin{cases} \text{le plus petit } y \leq x \text{ tel que } R(y, x_1, \ldots, x_n), \text{ s'il en existe un} \\ x \text{ sinon.} \end{cases}$$

Question 1.10. *Montrer que si R est récursive primitive, alors $\mathrm{Min}_{y=0}^{x} R(y, x_1, \ldots, x_n)$ aussi.*

Cette opération permet de définir des fonctions de manière implicite par exploration bornée. En voici deux exemples.

Question 1.11. *Montrer que la fonction $n \mapsto$ nième nombre premier est récursive primitive.*

Question 1.12. *La fonction de Cantor $\langle x, y\rangle : (x, y) \mapsto ((x+y)^2+3x+y)/2$ envoie bijectivement $\mathbb{N}^2$ sur $\mathbb{N}$. Montrer que $\langle x, y\rangle$ est primitive récursive, ainsi que les deux fonctions K et L telles que $K(\langle x, y\rangle) = x$ et $L(\langle x, y\rangle) = y$.*

1.5 Récurrences d'ordre supérieur

Question 1.13. *En utilisant la question précédente, montrer que la suite de Fibonacci, définie par récurrence, est récursive primitive :*

$$f(0) = 1, \quad f(1) = 1, \quad f(n+2) = f(n+1) + f(n).$$

Conclusion

Malgré leur puissance d'expressivité, les fonctions primitives récursives ne capturent pas toute la richesse des fonctions que peut calculer une machine de Turing (qui seront présentées plus tard dans le cours et que l'on appelle simplement *récursives*). Il leur manque peu : si on rajoute à leur définition l'opération de *minimisation non-bornée* (correspondant au "while"), définie comme la minimisation bornée en Section 1.4, sauf qu'on ne restreint pas à $y \leq x$, alors on obtient des fonctions partielles (il est possible que la condition ne soit jamais satisfaite) appelées *fonctions récursives partielles*. Celles des fonctions récursives partielles qui sont totales (définies pour toute valeur de leur argument) sont exactement les fonctions récursives, donc toutes les fonctions calculables par une machine de Turing, présentées dans quelques séances.

Complément : la fonction d'Ackermann n'est pas récursive primitive

Pour montrer qu'une fonction n'est pas récursive primitive, il ne suffit pas de disposer d'une définition de cette fonction qui viole une des contraintes.

Question 1.14. *La fonction* min *peut être définie par*

$$\min(0, y) = 0, \quad \min(x+1, 0) = 0, \quad \min(x+1, y+1) = \min(x, y) + 1.$$

qui n'est pas une définition de fonction récursive. Donner une preuve que min *est quand même récursive primitive.*

La fonction d'Ackermann est un des premiers exemples historiques de fonction "calculable" qui ne soit pas récursive primitive. Les questions qui suivent prouvent ce résultat.
Définition. Soit A_n la suite de fonctions définie par

$$A_0(m) = m + 1, \quad A_{n+1}(0) = A_n(1), \quad A_{n+1}(m+1) = A_n(A_{n+1}(m)).$$

La *fonction d'Ackermann* est définie [1] par $A(n, m) = A_n(m)$.

1. Il s'agit de la définition moderne de cette fonction. Ackermann avait défini une variante assez semblable, mais à trois arguments. Sa construction a été simplifiée plus tard.

Question 1.15. *Montrer que pour n fixé, la fonction A_n est récursive primitive.*

Comme on l'a vu avec min, il faut une idée supplémentaire pour prouver qu'une fonction n'est pas récursive primitive. La construction d'Ackermann vise à produire une fonction à croissance "trop rapide" pour être récursive primitive. Par exemple, pour les premières valeurs de l'indice, on observe facilement par récurrence que

$$A_1(m) = m + 2, \quad A_2(m) = 2m + 3, \quad A_3(m) = 8 \cdot 2^m - 3, \quad A_4(m) = 2^{2^{\cdot^{\cdot^{2}}}} - 3.$$

La question suivante mesure cette croissance ; dans un premier temps, elle peut être sautée et son résultat admis.

Question 1.16. *Montrer que pour tous n, m, $A_n(m) > m$, que les fonctions A_n sont strictement croissantes, et que pour tous n, m, $A_n(m) \leq A_{n+1}(m)$. Enfin, montrer que $A_k(A_m(n)) \leq A_{2+\max(k,m)}(n)$.*

Le point clé est le suivant.

Question 1.17. *Pour toute fonction récursive primitive $f(x_1, \ldots, x_k)$, montrer qu'il existe n tel que*

$$\forall (x_1, \ldots, x_k), \quad f(x_1, \ldots, x_k) \leq A_n(x_1 + \cdots + x_k).$$

Question 1.18. *Conclure que la fonction d'Ackermann n'est pas récursive primitive.*