École polytechnique — X2013 — INF412 Fondements de l'informatique

Fonctions Récursives Primitives

Sujet proposé par Bruno Salvy

(corrigé)

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 $n = 1$) ;

— $\mathsf{Proj}_n^i : (x_1, \dots, x_n) \mapsto x_i$ les fonctions de projection, pour $1 \leq i \leq n$;

— $\mathsf{Comp}_n(g, h_1, \dots, h_m) : (x_1, \dots, x_n) \mapsto g(h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n))$ la composition des fonctions récursives primitives $g, h_1, \dots, h_m$;

— $\mathsf{Rec}(g, h)$ la fonction définie par récurrence comme

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

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 $g, h, h_1, \dots, h_m$ éventuellement en jeu.*

Solution : La définition par récurrence (c'est-à-dire avec Rec) de $+$ est la suivante :

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

Dans le premier cas, la fonction g utilisée est la projection sur son argument. Dans le second, la fonction h est définie par $h : (x_1, x_2) \mapsto x_1 + 1$. Il s'agit d'une composition, avec h_1 la projection sur le premier argument, et g le successeur. En résumé :

$$+ = \mathsf{Rec}(\mathsf{Proj}_1^1, \mathsf{Comp}_3(\mathsf{Succ}, \mathsf{Proj}_3^1)).$$

$\square$

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

Solution : Les définitions récursives sont simplement

$$*(0, y) = 0, \quad *(x+1, y) = +(*(x, y), y), \quad \hat{}(x, 0) = 1, \quad \hat{}(x, y+1) = *(\hat{}(x, y), x).$$

La première est une application immédiate de la définition par récurrence :

$$* = \mathsf{Rec}(\mathsf{Zero}, \mathsf{Comp}_3(+, \mathsf{Proj}_3^1, \mathsf{Proj}_3^3)).$$

Pour la puissance, la récurrence agit sur le second argument. Il est plus commode d'écrire d'abord une fonction $(x, y) \mapsto y^x$, puis d'intervertir les arguments. On commence donc par cette fonction auxiliaire qui ressemble beaucoup au produit :

$$\hat{}_{\text{aux}} = \mathsf{Rec}(\mathsf{One}, \mathsf{Comp}_3(*, \mathsf{Proj}_3^1, \mathsf{Proj}_3^3)),$$

où $\mathsf{One} = \mathsf{Comp}_2(\mathsf{Succ}, \mathsf{Zero})$. Ensuite, on définit une construction qui intervertit les arguments :

$$\mathsf{Swap}(f) = \mathsf{Comp}_2(f, \mathsf{Proj}_2^2, \mathsf{Proj}_2^1).$$

La puissance est donc finalement donnée par

$$\hat{} = \mathsf{Swap}(\hat{}_{\text{aux}}).$$

$\square$

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 $f(x_1, \ldots, x_n)$ et $g(x_1, \ldots, x_n)$ sont des fonctions récursives primitives, alors les prédicats

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

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.

Solution : La définition est une simple récurrence :

$$x \dot{-} 0 = x, \quad x \dot{-} (y+1) = \mathsf{Pred}(x \dot{-} y).$$

Comme pour la puissance, il est donc plus commode d'introduire une fonction auxiliaire, puis d'intervertir les arguments. La fonction est donc obtenue par

$$\dot{-} = \mathsf{Swap}(\mathsf{Rec}(\mathsf{Proj}_1^1, \mathsf{Comp}_3(\mathsf{Pred}, \mathsf{Proj}_3^1))).$$

□

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.]*

Solution : Suivant l'indication, on peut commencer par définir un prédicat qui teste si un entier est nul :

$$\mathsf{IsZero} := \mathsf{Rec}(\mathsf{Succ}(0), \mathsf{Comp}_2(0)).$$

D'autres variantes exploitent la puissance (avec nos définitions, $0^x = 0$ si et seulement si $x \neq 0$), ou la fonction $\dot{-}$ (puisque $1 \dot{-} x$ reconnaît 0).

D'après la définition de la soustraction tronquée, $x \leq y$ si et seulement si $x \dot{-} y = 0$. À partir de là, on déduit alors successivement

$$\begin{aligned} \leq &= \mathsf{Comp}_2(\mathsf{IsZero}, \dot{-}); \\ \geq &= \mathsf{Swap}(\leq); \\ < &= \mathsf{Comp}_2(\mathsf{IsZero}, \geq); \\ = &= \mathsf{Comp}_2(*, \leq, \geq). \end{aligned}$$

Là encore une autre variante possible, utilisant le fait que les fonctions n'ont que des entiers comme arguments, repose sur l'équivalence entre $x \leq y$ et $x < y + 1$.

La conclusion générale s'obtient alors par composition. □

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

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

Solution : Les idées sont déjà apparues dans la question précédente. La fonction IsZero agit comme une négation, que l'on peut utiliser par composition :

$$\mathsf{Not} = \mathsf{IsZero}, \quad \neg R = \mathsf{Comp}_n(\mathsf{Not}, R).$$

Ensuite, il suffit de multiplier ou d'additionner :

$$\wedge(R, S) = \mathsf{Comp}_n(*, R, S), \quad \vee(R, S) = \mathsf{Comp}_1(\mathsf{Not}, \mathsf{Comp}_1(\mathsf{IsZero}, \mathsf{Comp}_n(`+`, R, S))).$$

Pour le second, on peut aussi utiliser $R \vee S = \neg(\neg R \wedge \neg S)$. □

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*

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

Montrer que h est récursive primitive.

Solution : Il suffit d'additionner et de multiplier : $h = +(*(f, R), *(g, \neg R))$. L'écriture avec Comp est claire. □

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*

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

Solution : C'est un cas typique de programmation récursive : si $z = 0$ le résultat est $g(0, y)$ alors qu'en $z + 1$ il faut ajouter (ou multiplier) $g(z + 1, y)$ au résultat en z. Formellement, on obtient donc pour la somme

$$\mathsf{Rec}(\mathsf{Comp}_1(g, \mathsf{Zero}, \mathsf{Proj}_1^1), \mathsf{Comp}_2(+, \mathsf{Proj}_2^1, \mathsf{Comp}_3(g, \mathsf{Comp}_3(\mathsf{Succ}, \mathsf{Proj}_3^2), \mathsf{Proj}_3^3)))$$

et une formule analogue pour le produit en remplaçant $+$ par $*$. $\square$

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

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

Solution : C'est un cas particulier du précédent, en composant par $\neg\mathsf{IsZero}$ dans le premier cas pour ramener le résultat dans $\{0, 1\}$. $\square$

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

Solution : Il suffit maintenant d'utiliser les relations précédentes. On obtient successivement les prédicats :
- Divisibilité : $\mathsf{div}(x, z) = \exists y \leq x, yz = x$.
- Nombre composite : $\mathsf{Composite}(x) = \exists y \leq x, y > 1 \wedge y < x \wedge \mathsf{div}(x, y)$.
- Nombre premier : $\mathsf{Prime}(x) = \neg\mathsf{Composite}(x)$.

$\square$

1.4 Exploration bornée

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

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

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

Solution : L'idée de base est d'obtenir y en additionnant 1 pour chaque entier inférieur à y. On définit d'abord une fonction auxiliaire par if-then-else :

$$f_1(x, x_1, \ldots, x_n) = \begin{cases} 0 & \text{si } \exists y \leq x, R(y, x_1, \ldots, x_n), \\ 1 & \text{sinon.} \end{cases}$$

Ensuite, le résultat est obtenu par $\sum_{y \leq x} f_1(y, x_1, \ldots, x_n)$. $\square$

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.*

Solution : Dès lors que l'on dispose d'une fonction récursive primitive $B(n)$ telle que le n^{e} nombre premier est inférieur ou égal à $B(n)$, il suffit de définir par récurrence

$$p_0 = 2 \quad \text{et} \quad p_{n+1} = \operatorname*{Min}_{x \leq B(n)} [p_n < x \wedge \mathsf{Prime}(x)].$$

Plusieurs possibilités sont offertes pour le choix de cette borne. Si on admet le postulat de Bertrand, prouvé par Tchebychev [1], qui dit que pour tout entier $n \geq 2$, il existe un nombre premier p tel que $n < p < 2n$, on en déduit par récurrence la borne $B(n) = 2^{n+1}$. Sinon, le produit $\prod_{k \leq n} p_k$, plus 1, donne une borne supérieure ; c'est la base de la preuve d'Archimède de l'infinité des nombres premiers. Cette borne dépend des p_k précédents, mais on peut prendre la borne plus simple $p_{n+1} \leq p_n! + 1$ et définir avec Rec une fonction $B(n)$ qui vaut 1 pour $n = 0$ et $B(n+1) = B(n)! + 1$ au-delà. □

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* $L(\langle x, y \rangle) = y$*.*

Solution : Pour coder la division par 2, on peut à nouveau utiliser l'opérateur Min :

$$\langle x, y \rangle = \operatorname*{Min}_{z \leq (x+y)^2 + 3x + y} [2z = (x+y)^2 + 3x + y].$$

Les équations d'inversion sont résolues à l'aide de Min également :

$$K(z) = \operatorname*{Min}_{x \leq z} [\exists y \leq z, \langle x, y \rangle = z], \quad L(z) = \operatorname*{Min}_{y \leq z} [\langle K(z), y \rangle = z].$$

□

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 :*

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

Solution : La difficulté provient de ce que la récurrence a besoin de deux termes précédents. On va donc écrire une récurrence d'ordre 1 sur les paires $(f(n), f(n+1))$, d'abord codées sur un seul entier par la question précédente.

$$\mathsf{fib1}(0) = \langle 1, 1 \rangle, \quad \mathsf{fib1}(n+1) = \langle L(\mathsf{fib1}(n)), R(\mathsf{fib1}(n)) + L(\mathsf{fib1}(n)) \rangle.$$

La suite elle-même est obtenue par composition avec K. □

1. Pour une preuve élémentaire due à Paul Erdös, on pourra consulter le réjouissant livre *Raisonnements divins : Quelques démonstrations mathématiques particulièrement élégantes.* par Martin Aigner et Gunter M. Ziegler. Le point de départ est l'observation simple que le produit des nombres premiers entre n et $2n$ divise le coefficient binomial $\binom{2n}{n}$.

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.*

Solution : Une solution simple consiste à le définir par un if-then-else : $\min(x, y) = x$ si $x \leq y$, y sinon. □

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 [2] par $A(n, m) = A_n(m)$.

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

Solution : La preuve est par récurrence sur n. Pour $n = 0$ on a $A_0 = \mathsf{Succ}$. Pour $n + 1$ on a $A_{n+1} = \mathsf{Rec}(A_n(1), \mathsf{Comp}_2(A_n, \mathsf{Proj}_2^1))$. □

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 $A_k(A_m(n)) \leq A_{2+\max(k,m)}(n)$.*

2. 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.

Solution : Les deux premières propriétés s'obtiennent par récurrence immédiate. Pour $n = 0$ elles découlent de la forme de la solution A_0, ensuite on a $A_{n+1}(0) = A_n(1) > 1 > 0$ et $A_{n+1}(m+1) = A_n(A_{n+1}(m)) > A_{n+1}(m)$ donne la croissance stricte, dont la première inégalité est une conséquence.

La troisième propriété s'obtient par récurrence sur n. Pour $n = 0$, elle s'observe sur les formes de A_0 et A_1. Ensuite si la propriété est vraie pour n on l'obtient pour $n+1$ par récurrence sur m. Pour $m = 0$, $A_{n+1}(0) = A_n(1) > A_n(0)$ par croissance de A_n. Puis $A_{n+1}(m) = A_n(A_{n+1}(m-1)) \geq A_n(m)$ puisque $A_{n+1}(m-1) > m-1$ et que A_n est croissante.

Enfin, si $s = \max(k, m)$, les inégalités précédentes entraînent

$$A_k(A_m(n)) \leq A_s(A_{s+1}(n)) = A_{s+1}(n+1) = A_{s+1}(A_0(n)) \leq A_{s+2}(n).$$

□

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*

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

Solution : On repart de la définition inductive des fonctions récursives primitives. Pour Zero, Succ et Proj, la propriété est obtenue avec $n = 0$. Si g et $h_1, \ldots, h_m$ sont des fonctions récursives primitives, bornées par A_n et $A_{n_1}, \ldots, A_{n_m}$ alors leur composition est bornée par $A_{2m+\max(n, n_1, \ldots, n_m)}$.

Reste le cœur de l'idée : la récurrence. On fait ici la preuve pour le cas d'une fonction f à deux variables définie à partir de g et h à une et trois variables, bornées par A_m et A_k. Pour le cas de base, $f(0, y) = g(y) \leq A_m(y)$. Ensuite si $f(x, y) \leq A_\ell(x+y)$, alors

$$\begin{aligned} f(x+1, y) &= h(f(x,y), x, y) \leq A_k(f(x,y) + x + y) \leq A_k(A_\ell(x+y) + x + y) \\ &\leq A_k(A_\ell(x+y) + A_0(x+y)) \leq A_k(2A_\ell(x+y)) \leq A_k(A_2(A_\ell(x+y))) \leq A_{k+4}(A_\ell(x+y)). \end{aligned}$$

Ainsi, si $\ell = \max(k+5, m)$ on conclut cette suite d'inégalités par

$$f(x+1, y) \leq A_{\ell-1}(A_\ell(x+y)) = A_\ell(x+y+1),$$

ce qui termine la preuve. □

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

Solution : On considère maintenant la fonction $f(n) = A(n, n)$. Si la fonction d'Ackermann était récursive primitive, alors f le serait aussi par composition et projection. D'après le résultat précédent, il existerait alors un m tel que $A_n(n) \leq A_m(n)$ pour tout n. Ceci contredit la croissance par rapport à l'indice établie en question 1.16. □