

Soit $<_{\text{tuple}}$ une fonction sur deux n-uplets de naturels de même longueur définie par:

$$\begin{aligned} <_{\text{tuple}} ([], []) &= \text{false} \\ <_{\text{tuple}} (x : xs, y : ys) &= \begin{cases} \text{true} & \text{si } x < y \\ <_{\text{tuple}} (xs, ys) & \text{si } x = y \\ \text{false} & \text{si } x > y \end{cases} \end{aligned}$$

On peut montrer par induction sur la longueur des n-uplets que cette relation est bien-fondée (puisque que $<$ sur les naturels l'est). Il ne peut donc y avoir de chaîne infinie de n-uplets X_i telle que $\forall i \in \mathbb{N}, X_{i+1} <_{\text{tuple}} X_i$, et ce, quel que soit X_0 . Intuitivement, pour assurer la termination d'une récurrence, il suffit donc d'assurer que les arguments à toute récursion réponde à ce critère (pour tout appel avec pour argument le n-uplet X_i , il faut s'assurer que les appels récursifs se fassent sur des n-uplets $X_n <_{\text{tuple}} X_i$). Ceci est un cas spécial du *size change termination principle* (Neil D. Jones, Amir Ben-Amram et Chin Soon Lee @ TOPPS, DIKU).

Le but est donc d'obtenir un modèle de calcul tel que tout programme termine, quelles que soit les entrées, en exploitant la relation bien-fondée ci-haut. Pour fins de simplifications, prenons pour acquis que le programme est constitué d'une seule fonction récursive, f , qui prend comme seul argument un n-uplet, et a accès à un ensemble arbitraire de primitives dont la termination est elle garantie (e.g. `inc`).

Afin d'obtenir un critère syntaxique de terminaison, exposons comme seul moyen de récursion une primitive, `selfk`, ne permettant d'exprimer que des appels répondant au critère ci-haut. Dans la fonction f qui prend comme seul argument un n-uplet $x = [x_1, x_2, \dots, x_n]$, `selfm` est une primitive d'arité $m < n$. `selfm` utilisée avec m arguments y_1, y_1, \dots, y_m , réduit à $f[x_1, x_2, \dots, x_{n-m-1}, y_1, \dots, y_m]$. Ainsi, les m derniers éléments du n-uplet argument prennent des valeurs arbitraires (y_1, \dots, y_m), mais, puisque le $m + 1$ dernier élément est décrémenté, la descente lexicographique est garantie (l'argument à l'appel récursif est $<_{\text{tuple}}$ l'argument original).

Cela ne garanti toutefois toujours pas la totalité de f : rien n'empêche d'écrire un programme qui décrémente un élément qui est déjà nul. Pour éviter ce problème, il suffit de définir f par cas, selon la non-nullité de chacun des éléments du n-uplet. Il est alors trivial d'assurer que `self` ne sera pas utilisé pour décrémenter un élément nul (si le cas ne garanti pas que $x_i > 0$, `selfn-i` ne peut être utilisé). Cela donne, pour `Ackermann` : $\mathbb{N}^2 \rightarrow \mathbb{N}$, la définition suivante:

$$\begin{aligned} A[m = 0, n \geq 0] &= n + 1 \\ A[m > 0, n = 0] &= \text{self}_1(1) \\ A[m > 0, n > 0] &= \text{self}_1(\text{self}_0()) \end{aligned}$$

On peut assurer que tous les cas sont couverts en énumérant les $2^n = 4$ cas, au lieu de permettre des 'jokers', et arriver à cette définition:

$$\begin{aligned}A[m = 0, n = 0] &= 1 \\A[m = 0, n > 0] &= n + 1 \\A[m > 0, n = 0] &= \mathbf{self}_1(1) \\A[m > 0, n > 0] &= \mathbf{self}_1(\mathbf{self}_0())\end{aligned}$$

Notons que la définition avec 'joker' est facilement macroexpressible en terme de celle avec tous les 2^n cas explicitement énumérés. Il serait aussi possible d'ajouter un argument (constant) à **self**, qui permettrait de décrémenter par plus de $n > 1$ à la fois. Il faudrait alors garantir que l'argument décrémenté $x_i \geq n$ dans le cas. Il est aussi possible de simplifier plus encore le critère de terminaison, en fixant l'ordre dans lesquels les cas doivent être définis. On pourrait par exemple forcer l'ordre utilisé ci-haut. Dans le 0^e cas ($x_1 = 0, x_2 = 0$), aucun **self** ne peut être utilisé; dans le 1^{er} ($x_1 = 0, x_2 > 0$), seul **self**₀; dans le 2^e ($x_1 > 0, x_2 = 0$), seul **self**₁; et dans le 3^e ($x_1 > 0, x_2 > 0$), **self**₀ et **self**₁. La généralisation à plus de 2 arguments est évidente; il suffit d'observer la représentation binaire de la position de chaque cas.