Funcţii primitiv recursive

Informal, o funcţie primitiv recursivă este o funcţie care poate fi procesată de un program care foloseşte doar bucle "for" (fără while), mai exact există o limită superioară a numărului de iteraţii şi poate fi calculată înainte de a intra în buclă.

O funcţie primitiv recursivă primeşte un număr de argumente (numere naturale) şi returnează un număr natural. Dacă primeşte n argumente, se numeşte funcţie de aritate n.

În continuare menţionăm nişte noţiuni care ne ajută să definim formal funcţiile primitiv recursive.

Funcţii de bază:

(sunt şi ele primitiv recursive ofc)

1. Funcţia constantă (aritate k)
   $C_n^k(x_1, \ldots, x_k) \stackrel{\text{def}}{=} n$

2. Funcţia succesor - aritate 1, returnează succesorul argumentului
   $S(x) \stackrel{\text{def}}{=} x + 1$

3. Funcţia proiecţie (aritate k)
   $P_i^k(x_1, \ldots, x_k) \stackrel{\text{def}}{=} x_i$

Operaţii cu funcţii de bază:

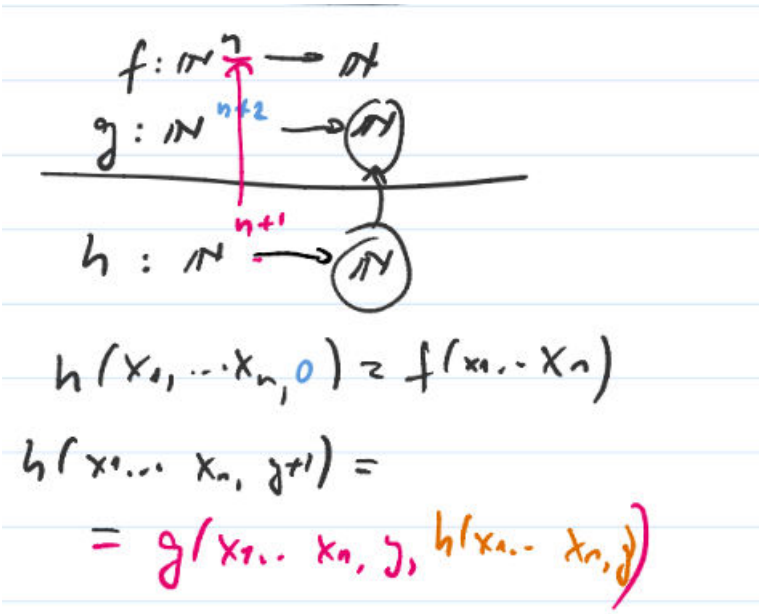
1. Compunere
   Dându-se o funcţie de aritate m $h(x_1,\ldots,x_m)$ şi m funcţii de aritate k $g_1(x_1,\ldots,x_k), \ldots, g_m(x_1,\ldots,x_k)$, avem:
   $h \circ (g_1, \ldots, g_m) \stackrel{\text{def}}{=} f$
   unde $f(x_1, \ldots, x_k) = h(g_1(x_1, \ldots, x_k), \ldots, g_m(x_1, \ldots, x_k))$

2. Recursie primitivă ρ
   Dându-se funcţia de aritate k $g_1(x_1,\ldots,x_k)$ şi funcţia de aritate (k+2) $h(y, z,$

$x_1,\ldots,x_k$), avem:

$$\rho(g,h) \stackrel{\text{def}}{=} f$$
$$f(0,x_1,\ldots,x_k) = g(x_1,\ldots,x_k)$$
$$f(S(y),x_1,\ldots,x_k) = h(y,f(y,x_1,\ldots,x_k),x_1,\ldots,x_k)$$

Pun şi poză din curs că e colorat şi se înţelege puţin (el schimbă puţin ordinea argumentelor/numele funcţiilor dar e la fel):



Definiţie: O funcţie este primitiv recursivă dacă este o funcţie de bază sau dacă se obţine din funcţii de bază aplicând aceste operaţii de un număr finit de ori.

Examples [edit]

- C_0^1 is a 1-ary function which returns 0 for every input: $C_0^1(x) = 0$.
- C_1^1 is a 1-ary function which returns 1 for every input: $C_1^1(x) = 1$.
- C_3^0 is a 0-ary function, i.e. a constant: $C_3^0 = 3$.
- P_1^1 is the identity function on the natural numbers: $P_1^1(x) = x$.
- P_1^2 and P_2^2 is the left and right projection on natural number pairs, respectively: $P_1^2(x,y) = x$ and $P_2^2(x,y) = y$.
- $S \circ S$ is a 1-ary function that adds 2 to its input, $(S \circ S)(x) = x + 2$.

Orice funcţie recursiv primitivă este intuitiv calculabilă!

În continuare vom demonstra că unele funcţii sunt primitiv recursive.

1. f(x,y) = x+y

Folosind recursie primitivă, știm $f(0,y) = g(y)$ din teorie.

$f(0,y) = 0 + y = y$, așadar obținem $g(y) = y = P_1^1$.

$f(S(y),\ z) = h(y,\ f(y,\ z),\ z)$ (din teorie) $\Leftrightarrow$

$\Leftrightarrow\ f(y+1,\ z)\ =\ h(y,\ y+z,\ z)$

$f(y+1,\ z)$ trebuie să fie $y+z+1$, așadar impunem condiția

$h(y,\ y+z,\ z)\ =\ y+z+1\ =\ S(P_2^3(y,\ y+z,\ z))\ \Rightarrow$

$\Rightarrow h = S \circ P_2^3\ \Rightarrow f = \rho(P_1^1,\ S \circ P_2^3)\ \Rightarrow f$ *recursiv primitivă*

2. f(x,y) = x*y

$f(0,y)\ =\ 0$

$f(0,y)\ =\ g(y)\ \Rightarrow\ g(y)\ =\ 0\ =\ C_0^1$

$f(S(y),\ z) = h(y,\ f(y,z),\ z)\ \Leftrightarrow$

$\Leftrightarrow f(y+1,\ z) = h(y,\ y*z,\ z)$

$h(y,\ y*z,\ z) = y*z + z = P_2^3 + P_3^3$

$P_2^3 + P_3^3$ *este primitiv recursivă deoarece am demonstrat mai sus că adunarea este primitiv recursivă.*

$\Rightarrow f = \rho(C_0^1,\ +\ \circ (P_2^3,\ P_3^3))\ \Rightarrow f$ *primitiv recursivă*

3. sign(x) = 0 dacă x=0, 1 altfel

$$\text{sign(x)} = \begin{cases} 0, \text{ dacă } \text{x} = 0 \\ 1, \text{ altfel} \end{cases}$$

$$\text{sign}(0) = 0 = \text{g} = \text{C}_0^1 \ (g \ este \ funcție \ de \ aritate \ 0 \Rightarrow constantă)$$

$$\text{sign}(\text{S(y)}) = \text{h(y, sign(y))} = 1 = \text{C}_1^2 \ (S(y) > 0 \ mereu \Rightarrow sign(S(y)) = 1)$$

$$\Rightarrow \text{sign} = \rho(\text{C}_0^1, \text{C}_1^2) \Rightarrow sign \ primitiv \ recursivă$$

Alte funcţii primitiv recursive: funcţia putere, factorial, predecesor, min, max, împărţire, modulo, împărţire întreagă, al n-lea număr prim, etc.

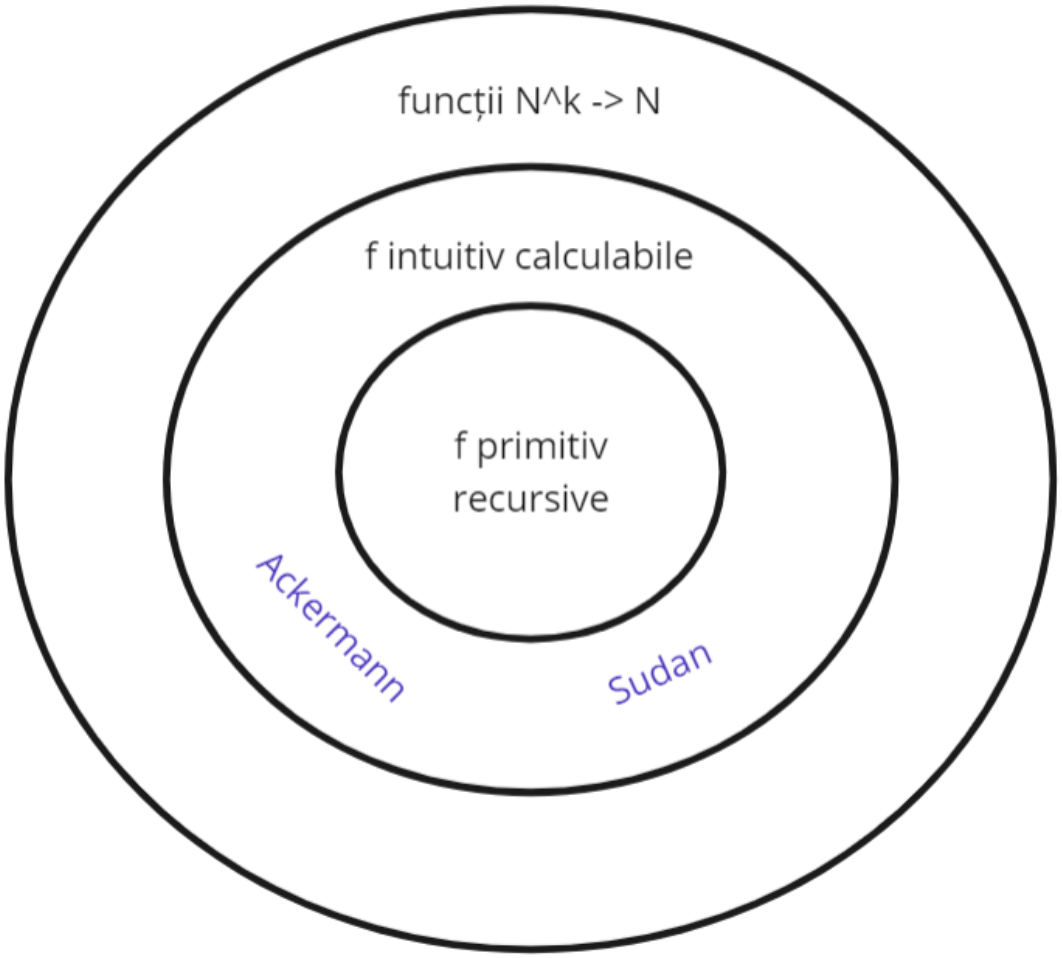
Funcţie totală = funcţie care este definită pe întreg domeniul (de exemplu, pe toată mulţimea N, nu doar pe o parte a ei)

Funcţia lui Ackermann:

$$\begin{aligned} \text{A}(0, n) &= n + 1 \\ \text{A}(m + 1, 0) &= \text{A}(m, 1) \\ \text{A}(m + 1, n + 1) &= \text{A}(m, A(m + 1, n)) \end{aligned}$$

Funcţia lui Ackermann **NU** este primitiv recursivă! Este o funcţie totală calculabilă, dar nu este primitiv recursivă. Este considerată prima de acest tip care a fost descoperită. Se cunoaşte în schimb că orice funcţie primitiv recursivă este totală şi calculabilă.

Funcţia lui Ackermann nu este tocmai prima descoperită, însă este considerată astfel datorită popularităţii ei. Atât Ackermann cât şi Gabriel Sudan (român) erau studenţi ai lui Hilbert, iar Gabriel Sudan a descoperit o astfel de funcţie la scurt timp înainte de Ackermann (care a descoperit-o independent).



Funcţii parţiale

Trebuie să admitem noţiunea de funcţie care uneori nu ia valori. Spre exemplu:

$$f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$$
$$f(x, y) = x - y$$

Funcţia de mai sus ia valori doar când x>=y.

Minimizarea/operatorul μ:

Operatorul μ ne ajută să găsim cel mai mic număr natural cu o anumită proprietate.

$g(x_1,...,x_n) = \mu y \{f(x_1,...,x_n, y)=0\}$, unde f este o funcţie totală.

Partea din dreapta se citeşte "cel mai mic y cu proprietatea că **$f(x_1,...,x_n, y)=0$**".

Definiţie: O funcţie parţială se numeşte **parţial recursivă** dacă şi numai dacă poate fi obţinută din funcţii de bază, compunere, recursie primitivă **şi minimizare**.

O mulţime S se numeşte **recursiv enumerabilă** dacă există o funcţie parţial recursivă al cărei domeniu este exact S.

Teza Church-Turing

Godel a creat o definiţie formală a unei clase numite funcţii general recursive (~acelaşi lucru cu cele parţial recursive) şi a formulat că o funcţie este calculabilă dacă şi numai dacă este o funcţie general recursivă.

Church a creat o metodă de definire a funcţiilor, numită λ-calcul şi o codificare a numerelor naturale (numeralii Church). Acesta a formulat că o funcţie este calculabilă dacă şi numai dacă poate fi reprezentată ca un λ-termen.

Turing a creat maşina Turing şi a formulat că o funcţie este calculabilă dacă şi numai dacă poate fi calculată de maşina Turing.

Ulterior, Church şi Turing au demonstrat că aceste trei clase definite formal de funcţii calculabile coincid: o funcţie este λ-calculabilă dacă şi numai dacă este calculabilă Turing şi dacă şi numai dacă este general recursivă.