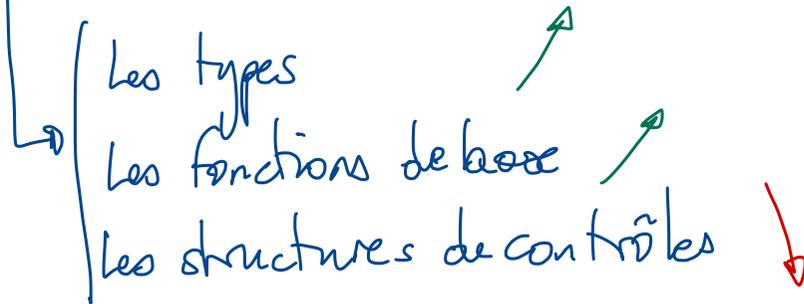


Fonctions vs Algorithmes

Revisiter les classes de fonctions (FP, par exemple)

Regarder les algorithmes / programmes écrivables pour une fonction de la classe



Le langage LOOP

Lo avec -1 +

Lo avec break +

le schéma de la récursion primitive

↳ Avec liste d'entiers +

↳ récursion mutuelle

↳ récursion avec paramètre variable

En fonction du type d'appel
valeur, nom, ...

$$\text{PR} \left[\begin{array}{l} f(0, y) = g(y) \\ f(x+1, y) = h(x, f(x, y), y) \end{array} \right.$$

$$\text{PRV} \left[\begin{array}{l} f(0, y) = g(y) \\ f(x+1, y) = h(x, f(x, j(x, y)), y) \end{array} \right.$$

- X = 0
- X = 2
- X = X + 1
- LOOP X DO

DONE

-
- X = X - 1
 - IF X = 0 THEN

ELSE

FI

- BREAK

$$\text{PRL} \left[\begin{array}{l} f([], y) = g(y) \\ f(x::xs, y) = h(x, xs, f(xs, y), y) \end{array} \right.$$

Notion de complétude algorithmique

Regarder le pouvoir
algorithmique des ODE discrètes

Classes d'algorithmes

Point de départ : Thèse de Gurevich

Tout algorithme séquentiel est représentable/implémentable
pos à pos par un programme d'algèbre évolutive (ASM/EA)

instructions $\left[\begin{array}{l} \bullet \text{ if } C \text{ then } f(t_1, \dots, t_n) := t_0 \quad [\text{Test and Set}] \\ \bullet i_1 \parallel i_2 \quad [\text{Simultanéité}] \end{array} \right.$

programme $\left[\text{itération de l'instruction jusqu'à point fixe} \right.$

t_i termes

f symbole de fonction

$:=$ mise à jour en 1 point

\parallel en même temps

ex if $x \neq 0$ then $x := x - 1$

\parallel if $y \neq 0$ then $y := y - 1$

ex if not NF(t) then $t := \text{Reduce}(t)$

⇒ Définir des classes d'algorithmes à partir de cette formalisation et thèse

⇒ Etant donnée une classe d'algorithmes C .
Est ce que le langage L implémente les algo de C ?

.....