Fonctions vs Algorithmes

Revisiter les classes de fonctions (FP, par exemple)

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

- Les types
- Les fonctions de base
- Les structures de contrôles

Le langage LOOP

- avec -1 +
- 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, ...

PR $\begin{cases} f(0,y) = g(y) \\ f(x+1,y) = h(x, f(x,y), y) \end{cases}$

PRV $\begin{cases} f(0,y) = g(y) \\ f(x+1,y) = h(x, f(x, j(x,y)), y) \end{cases}$

PRL $\begin{cases} f([\,],y) = g(y) \\ f(x::xs,y) = h(x, xs, f(xs,y), y) \end{cases}$

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

  DONE

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

  ELSE

  FI
- BREAK

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 pas à pas par un programme d'algèbre evolutive (ASM/EA)

instructions
- if C then $f(t_1, \dots, t_n) := t_0$ [Test and Set]
- $i_1 \parallel i_2$ [Simultanéité]

programme [itération de l'instruction jusqu'à point fixe

t_i termes
f symbole de fonction
$:=$ mise à jour en 1 point
$\parallel$ en même temps

ex
```
if x != 0 then x := x-1
|| if y != 0 then y := y-1
```

ex
```
if not NF(t) then t := Reduct(t)
```

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

⟹ Etant donnée une classe d'algorithme C.
Est ce que le langage L implémente les algos de C ?

.....