Llenguatges de Programació, FIB, 16 de juny de 2021

Possibles solucions

1. Pragmàtica dels LPs

Pascal fou un LP dissenyat per a l'educació que encoratjava bones pràctiques de programació basades en la programació estructurada. Tenir un sol punt de sortida és un avantatge perquè fa els procediments més entenedors. Tanmateix, algunes situacions habituals són més clares de programar amb **break**s i **continue**s (no cal tant d'aniuament de construccions i s'eviten variables booleanes per controlar el flux d'execució).

[Error comú: Perquè tinc un 8 enlloc d'un 10? Probablament has donat raons a favor i en contra correctes, però has oblidat de posar en context el disseny del llenguatge: Pascal era un LP per a l'ensenyança de la programació estructurada.]

2. Haskell

1. Amb la suma:

   instance *Monoide* **Int where**
   $\quad neutre = 0$
   $\quad (\diamond) = (+)$

   Amb el producte:

   instance *Monoide* **Int where**
   $\quad neutre = 1$
   $\quad (\diamond) = (*)$

2. Instanciació:

   instance *Monoide* $[t]$ **where**
   $\quad neutre = []$
   $\quad (\diamond) = (++)$

   Lleis:

   (1) $neutre \diamond x = [] ++ x = x$ ✓
   (2) $x \diamond neutre = x ++ [] = x$ ✓
   (3) $x \diamond (y \diamond z) = x ++ (y ++ z) = (x ++ y) ++ z = (x \diamond y) \diamond z$ ✓

3. Instanciació:

   instance *Monoid* $t \Rightarrow$ *Monoid* (**Maybe** t) **where**
   $\quad neutre = \mathbf{Nothing}$
   $\quad \mathbf{Nothing} \diamond a = a$
   $\quad a \diamond \mathbf{Nothing} = a$
   $\quad \mathbf{Just}\ a \diamond \mathbf{Just}\ b = \mathbf{Just}\ (a \diamond b)$

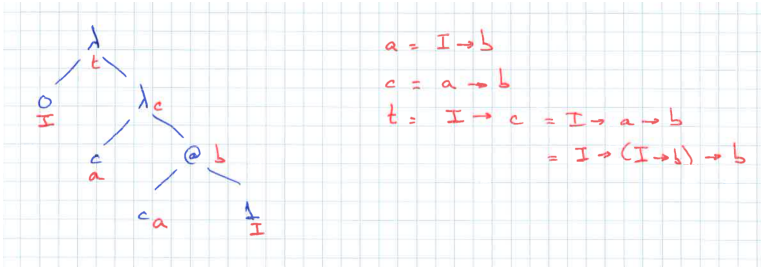
   Per brevetat, utilitzarem $N = \mathbf{Nothing}$ i $J = \mathbf{Just}$.

   Lleis: La (1) i la (2) són evidents. Per a la (3):

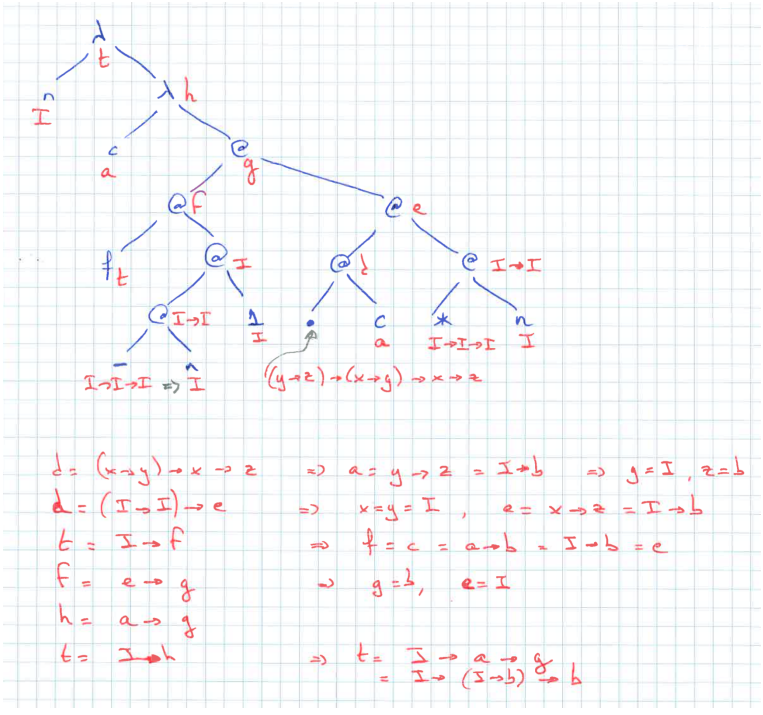
- Si x, y o z són N, es fàcil de veure. Per exemple, quan $x = N$:
  $x \diamond (y \diamond z) = N \diamond (y \diamond z) = y \diamond z = (N \diamond y) \diamond z = (x \diamond y) \diamond z$ ✓
  (Aquí hem usat la propietat de l'element neutre.)
- Si $x = J\ a$, $y = J\ b$ i $z = J\ c$:
  $x \diamond (y \diamond z) = J\ a \diamond (J\ b \diamond J\ c) = J\ a \diamond J\ (b \diamond c) = J\ (a \diamond (b \diamond c))$
  $= J\ ((a \diamond b) \diamond c) = J\ (a \diamond b) \diamond J\ c = (J\ a \diamond J\ b) \diamond J\ c = (x \diamond y) \diamond z$ ✓
  (Aquí hem usat el fet que t és un monoide.)

3. Inferència de tipus

1. Comencem amb l'arbre de la primera definició i apliquem l'algorisme:



Ens surt que $f :: t$ amb $t = I \to (I \to b) \to b$. Per tal de veure si podem trobar què és b, continuem amb l'arbre de la segona definició:



Vaja, no hem guanyat gaire... b és encara una variable de tipus.

2. Amb $f :: I \to (I \to b) \to b$ i $\mathbf{id} :: s \to s$, ara podem inferir que $f\ n\ \mathbf{id} :: I$.

3. $f\ n\ \mathbf{id}$ calcula el factorial d'un enter n (per a valors negatius es penja).

4. Lambda càlcul

1. $(\lambda x\,.\,\lambda\,y\,.\,(\text{ADD}\ y\ ((\lambda\ z\,.\,(\text{MUL}\ x\ z))\ 3)))\ 7\ 5$

   - Ordre normal:
     $(\lambda x\,.\,\lambda\,y\,.\,(\text{ADD}\ y\ ((\lambda\ z\,.\,(\text{MUL}\ x\ z))\ 3)))\ 7\ 5$
     $\to_\beta\ \lambda\,y\,.\,(\text{ADD}\ y\ ((\lambda\ z\,.\,(\text{MUL}\ 7\ z))\ 3))\ 5$
     $\to_\beta\ \text{ADD}\ 5\ ((\lambda\ z\,.\,(\text{MUL}\ 7\ z))\ 3$
     $\to_\beta\ \text{ADD}\ 5\ (\text{MUL}\ 7\ 3)$
     $\to_\beta\ \text{ADD}\ 5\ 21 \to_\beta 26.$
   - Ordre aplicatiu:
     $(\lambda x\,.\,\lambda\,y\,.\,(\text{ADD}\ y\ ((\lambda\ z\,.\,(\text{MUL}\ x\ z))\ 3)))\ 7\ 5$
     $\to_\beta\ (\lambda x\,.\,\lambda\,y\,.\,(\text{ADD}\ y\ (\text{MUL}\ x\ 3)))\ 7\ 5$
     $\to_\beta\ \ \lambda\,y\,.\,(\text{ADD}\ y\ (\text{MUL}\ 7\ 3))\ 5$
     $\to_\beta\ \lambda\,y\,.\,(\text{ADD}\ y\ 21)\ 5$
     $\to_\beta\ \text{ADD}\ 5\ 21$
     $\to_\beta\ 26.$

2. $(\lambda\,y\,.\,5)((\lambda x\,.\,x\,x)(\lambda x\,.\,x\,x))$

   - Ordre normal:
     $(\lambda\,y\,.\,5)((\lambda x\,.\,x\,x)(\lambda x\,.\,x\,x)) \to_\beta 5.$
   - Ordre aplicatiu:
     $(\lambda\,y\,.\,5)((\lambda x\,.\,x\,x)(\lambda x\,.\,x\,x))$
     $\to_\beta\ (\lambda\,y\,.\,5)((\lambda x\,.\,x\,x)(\lambda x\,.\,x\,x))$
     $\to_\beta\ (\lambda\,y\,.\,5)((\lambda x\,.\,x\,x)(\lambda x\,.\,x\,x))$
     $\to_\beta\ \cdots$

5. Compilació

```
grammar BenPosats;

root : par root
     |
     ;

par  : '(' root ')'
     | '[' root ']'
     ;
```

La gramàtica és LL(1) perquè a `root` es pot triar entre les dues produccions en funció de si el caràcter següent és un parèntesi o claudàtor obert (primera) o no (segona) i perquè a `par` es pot triar entre les dues produccions en funció de si el caràcter següent és un parèntesi obert (primera) o un claudàtor obert (segona).

6. Subtipos y herencia

(1) Sí, serien legals.

(2) No, perquè *m2* retorna un *A* i no es pot assignar un *A* a un *B*.

(3) No, tot i que l'assignació sí és legal. La comprovació de tipus en compilar farà que la invocació a *m3* doni un error.

(4) No, per les mateixes raons que l'apartat 3.

(5) Sí, seria legal.

(6) No, no es pot assignar un punter a un B a un $A*$.

(7) No, per les mateixes raons que l'apartat 3 i 4.

(8) L'*m1* d'A.

(9) L'*m1* de B (compila bé perquè A té *m1*, però l'objecte apuntat és instància de B).

(10) Al convertir el b en a, queda un A, és a dir que es perd tota la informació de B. Per tant, es crida al *m1* de A.