Programació funcional

Albert Rubio

Especialitat de Computació
Grau en Enginyeria Informàtica

FIB

Pla de la sessió

- Introducció Sistemes de tipus.
- Type safety
- Tipat fort/feble
- Comprovació Estàtica/Dinàmica
- Sistema de tipus del Haskell
- Sistema de tipus Java
- Inferència de tipus
- Inferència de tipus a C++
- Inferència de tipus a Haskell
- Algorisme de Milner.

Introducció

Usos principals dels tipus en els llenguatges de programació:

- Donar nom i organitzar conceptes.
- Assegurar-se que els bits de memòria s'interpreten consistentment.
- Donar informació al compilador sobre les dades manipulades pel programa.

Introducció

És important teniren compte que:

- No hi ha llenguatges de programació sense tipus
- Sempre hi ha alguna forma de tipus.
  No necessàriament explícits.

El fet d'exitir tipus implica l'aparició dels "errors de tipus"

Type safety

Un llenguatge és “type safe” si cap programa pot donar errors de tipus en temps d’execució.

Alguns errors típics en execució:

- Type Cast: usar un valor d’un tipus en un altre tipus. Per exemple, en C un enter pot ser usat com a funció (com a adreça), però pot saltar on no hi ha una funció (generant un error).
- Aritmètica de punters.
  Per exemple, a C si tenim
  `A* p;`
  llavors `*(p+i)` té tipus `A`, però el que hi ha a `p+i` pot ser qualsevol cosa i pot causar un error de tipus.

Type safety

- Alliberament explícit de memòria (deallocate/delete). Per exemple, a Pascal usar un apuntador alliberat pot donar errors de tipus.
- En llenguatges OO (antics). No existència d'un mètode (degut a l'herència).

- "Type safe": Haskell, Java, ...
- "Type unsafe": C, C++, ...

Tipat fort/feble

Els llenguatges amb tipat fort són els que imposen restriccions que eviten barrejar valors de diferents tipus. Per exemple, amb un tipat feble podem tenir:

```
a = 2
b = "2"

a + b       # JavaScript retorna "22"
a + b       # Visual Basic retorna 4
```

- Tipat fort: C++, Java, Python, Haskell, ...
- Tipat feble: Basic, JavaScript, Perl, ...

Comprovació Estàtica/Dinàmica

- Comprovació estàtica: en temps de compilació.

  Conservadora.
  Codi més eficient.

- Comprovació dinàmica: en temps d'execució.

En llenguatges OO antics (sense comprovació estàtica), es podien donar excepcions d'execució: "Method undefined".

Pot implicar, restringir construccions com l'herència, per garantir que sempre tot serà tiplable (type safe).

Sistema de tipus del Haskell

Usarem un estil de regles d'inferència (com al λ-calcul):

Variable:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

Declaració:

$$\frac{f :: \tau \in \Gamma}{\Gamma \vdash f : \tau\xi}$$

Abstracció:

$$\frac{\Gamma \cdot \{x : \sigma\} \vdash t : \tau}{\Gamma \vdash (\backslash x.t) : \sigma \to \tau}$$

Aplicació:

$$\frac{\Gamma \vdash s : \sigma \to \tau \quad \Gamma \vdash t : \sigma}{\Gamma \vdash (s\ \ t) : \tau}$$

Let:

$$\frac{\Gamma \vdash s : \sigma \quad \Gamma \cdot \{x : \sigma\} \vdash t : \tau}{\Gamma \vdash \text{let } x = s \text{ in } t : \tau}$$

Context declaration:

$$\frac{C[\alpha] => f :: \tau[\alpha] \in \Gamma \quad C[\sigma] \text{ es satisfà a } \Gamma}{\Gamma \vdash f : \tau[\sigma]\xi}$$

on ξ es una substitució de tipus

Sistema de tipus del Java

Usarem un estil de regles d'inferència (com al λ-calcul):

Constants:

$$\frac{}{\vdash 1 : \texttt{Int}} \qquad \ldots$$

Variables:

$$\frac{x : T \in E}{\vdash x : T}$$

Aplicació de funció:

$$\frac{E \vdash e_1 : T_1 \quad \ldots \quad E \vdash e_n : T_n}{E \vdash obj.f(e_1, \ldots, e_n) : T}$$

Si $obj : Class\{\ldots T\ f(T_1, \ldots, T_n) \ldots\} \in E$

Sistema de tipus del Java

Exemple de derivació de tipus

$$\frac{\dfrac{x : Object\{\ldots String\ toString()\ldots\} \in E}{E \vdash x.toString() : String}}{E \vdash System.out.println(x.toString()) : void}$$

ja que

$$System.out : PrintStream\{\ldots\ void\ println(String)\ldots\} \in E$$

és a dir, *System.out* és un objecte de la classe *PrintStream*

- Cal afegir la noció de subtipus (herència)
- Cal extendre les regles y afegir-ne de noves.

Sistema de tipus del Java amb herència

Aplicació de funció:

$$\frac{E \vdash e_1 : U_1 \quad \ldots \quad E \vdash e_n : U_n \quad E \vdash U_1 \leq T_1 \quad \ldots \quad E \vdash U_n \leq T_n}{E \vdash obj.f(e_1, \ldots, e_n) : T}$$

Si $obj : Class\{\ldots T\ f(T_1, \ldots, T_n) \ldots\} \in E$

Regles pel subtipat:

Reflexivity:

$$\frac{}{\vdash T \leq T}$$

Object:

$$\frac{T \text{ no és un tipus primitiu}}{\vdash T \leq Object}$$

Implements:

$$\frac{\texttt{class}\ U\ \texttt{implements}\ T \in E}{E \vdash U \leq T}$$

En general, si $U \leq T$ apareix directament a E amb un `implements` o un `extends`

Sistema de tipus del Java amb herència

Més regles pel subtyping

Arrays:

$$\frac{E \vdash U \leq T}{E \vdash U[\,] \leq T[\,]}$$

Transitivitat:

$$\frac{E \vdash U \leq T \qquad E \vdash T \leq S}{E \vdash U \leq S}$$

Cal definir de qui s'hereten els mètodes quan hi ha herència i redefinició de mètodes.

Solució: del més proper!

Problemes amb l'herència múltiple.

Sistema de tipus del Java amb herència

Suposeu que tenim

$$D \leq B \leq A$$
$$D \leq C \leq A$$

Aquí D presenta herència multiple de B i C.

Si D crida a un mètode d'A que ha estat redefinit per B i per C de maneres diferents.

De qui ha d'heretar el mètode?

És un problema de tipus!

Java no admet herència múltiple.

Per exemple, C++ i Python sí ho admeten però de formes diferents.

Inferència de tipus

Mecanisme (re)inventat per Robin Milner

(Curry i Hindley havien desenvolupat idees similars independentment en el context del lambda-calcul)

Donat un programa en un llenguatge de programació trobar el tipus més general pel programa (y totes les seves expressions) dins del sistema de tipus del llenguatge.

L'algorisme és similar a la "unificació".

- Sempre present als llenguatges funcionals.
- S'ha estès a altres llenguatges.
  Per exemple Visual Basic, C#, C++, ...

Inferència de tipus a C++

A la versió més recent de l'estàndar de C++ (C++0x/C++11) apareix la inferència de tipus.

Paraula clau "auto"

```
auto x = 0;
```

També hi ha la Paraula clau decltype

Obté el tipus d'una expressió.

```
decltype(u+1) m =0;
```

No necessariament genera el mateix tipus que auto. Vegeu l'exemple (`exam.cpp`)

Inferència de tipus a Haskell

- En la majoria de casos no cal definir res.
- Algunes situacions estranyes.
- Problemes amb l'anomenada "Monomorphism restriction"
  Sovint no es pot sobrecarregar una funció si no es dona una declaració explicita de tipus (veure exemples).

Es poden demanar els tipus inferits.

L'algorisme de Milner

1. S'assigna un tipus a l'expressió i a cada subexpressió.
   - Si el tipus conegut se li assigna aquest tipus.
   - Sinó se li assigna una variable de tipus.

   Recordeu que les funcions són expressions.
2. Es genera un conjunt de restriccions (d'igualtat principalment) a partir l'arbre de l'expressió.
   - Aplicació.
   - Abstracció.
   - Let
   - ...
3. Es resolen les restriccions usant unificació.