Indice

Capitolo 1

Introduzione

Il lavoro da me svolto consiste nell'implementazione dell'algoritmo di type inference di Haskell in ELPI.

1.1 Stato dell'arte

1.1.1 Premesse

Iniziamo questa trattazione ponendo alcune basi. Esse faciliteranno la comprensione degli argomenti successivi.

Haskell Haskell è un linguaggio di programmazione che adotta il paradigma di programmazione funzionale. Al suo interno è presente il lambda calcolo e il meccanismo delle type class; è presente inoltre una parte più ampia che corrisponde alle librerie.

Type inference La type inference è il rilevamento automatico del tipo di dato di un'espressione in un linguaggio di programmazione. La capacità di dedurre i tipi automaticamente semplifica molte attività di

programmazione, lasciando il programmatore libero di omettere le annotazioni sui tipi pur consentendo il type check.

ELPI ELPI è un linguaggio di programmazione logico. Esso è un'estensione con vincoli del linguaggio λProlog, il quale a sua volta è un'estensione di Prolog a una logica di ordine superiore.

Prolog Prolog è un linguaggio di programmazione che adotta il paradigma di programmazione logica. Si basa sul calcolo dei predicati (logica del prim'ordine); la sintassi è composta da formule dette clausole che sono disgiunzioni di letterali del prim'ordine. L'esecuzione di un programma Prolog è comparabile alla dimostrazione di un teorema mediante la regola di inferenza detta risoluzione (essa permette di passare da un numero finito di proposizioni assunte come premesse a una proposizione che funge da conclusione). I concetti fondamentali di questo linguaggio sono l'unificazione, la ricorsione in coda e il backtracking.

λProlog λProlog è, come già detto, un'estensione di Prolog. Le caratteristiche principali in aggiunta, rispetto a Prolog, sono il polimorfismo, la programmazione di ordine superiore e il lambda calcolo tipato.

1.1.2 Limiti di λProlog

Non essendo λProlog un linguaggio di programmazione con vincoli risulta impossibile implementare la type inference mediante esso. I limiti dell'utilizzo di tale linguaggio si riscontrano in particolare nel tentativo di codificare il tipaggio per i costrutti del let-in e della type class.
L'unica strategia percorribile sarebbe quella di codificare interamente il sistema punto per punto. In tal caso però si creerebbe un sistema estremamente

rigido, il che rende tale strategia impraticabile. Ad esempio, infatti, si potrebbe codificare tutto nelle stringhe e svolgere ogni operazione attraverso di esse; a quel punto ogni cosa sarebbe implementabile, il sistema in questione sarebbe infatti Turing completo. Il problema però è il fatto che in tal caso si dovrebbero abbandonare tutte le features del linguaggio λProlog; infatti la rigidezza del sistema creato non consentirebbe l'utilizzo delle stesse, poiché si risulterebbe vincolati alla nuova struttura.
Si è reso dunque necessario l'utilizzo di ELPI, la cui maggiore espressività (non nel senso della Turing completezza) permette di svolgere operazioni impossibili da codificare in λProlog. Prendiamo come esempio i due casi indicati precedentemente:

- Per codificare il tipaggio del let-in è necessario l'utilizzo del meccanismo mode di ELPI. Infatti esso offre un maggior controllo sugli elementi del codice poiché permette di accorgersi se essi sono delle variabili non istanziate, così da poterle gestire in modo appropriato.

- Per codificare il tipaggio della type class è necessario l'utilizzo dei vincoli. Infatti questi possono sussistere anche non totalmente istanziati e quindi permettono, ad esempio, di fissare l'obbligo di appartenenza di una variabile di tipo non istanziata ad un'istanza di type class.

Entrambi i requisiti sono caratteristiche presenti in ELPI ma non in λProlog. Risulta dunque evidente la necessità di utilizzare ELPI come linguaggio di programmazione per poter raggiungere lo scopo prefissato.

1.2 ELPI

1.2.1 Introduzione al linguaggio

1.2.2 Sistema di propagazione dei vincoli

1.3 Haskell

1.3.1 BNF utilizzata

1.3.2 Type class

1.3.3 Let-in

1.4 Finalità

Capitolo 2

Implementazione

2.1 STLC

2.2 Funzioni ricorsive, match

2.3 Type class, instanza, schema

2.4 Let-in

Capitolo 3

Conclusioni

3.1 Riassumendo

3.2 La mia esperienza con ELPI

3.3 Il mio lavoro

3.4 Sviluppi futuri

3.4.1 Parser

3.4.2 Testing

3.4.3 Estensioni

Ringraziamenti