

Il linguaggio PCF

PCF (*Programming Computable Functions*) è un linguaggio di programmazione basato sul λ calcolo tipato con in aggiunta un operatore Y di punto fisso

I tipi di PCF

I tipi di PCF sono definiti ricorsivamente a partire dalle seguenti clausole:

- Nat e $Bool$ sono tipi (*i tipi base*)
- Se S e T sono tipi, $S \times T$ è un tipo
- Se S e T sono tipi, $S \rightarrow T$ è un tipo

Example

- $Nat \times Nat$
- $(Nat \times Bool) \rightarrow Bool$
- $Nat \rightarrow Nat \rightarrow Nat$ (da intendere $Nat \rightarrow (Nat \rightarrow Nat)$)
- $(Nat \rightarrow Nat) \rightarrow Nat \rightarrow Nat$

Grammatica per generare i termini di PCF

$$\langle \text{nat_exp} \rangle ::= 0 | 1 | 2 | \dots | \langle \text{nat_exp} \rangle + \langle \text{nat_exp} \rangle$$
$$\langle \text{bool_exp} \rangle ::= \text{true} | \text{false} | \text{Eq? } \langle \text{nat_exp} \rangle \langle \text{nat_exp} \rangle$$
$$\langle \sigma \rightarrow \tau_exp \rangle ::= \lambda(x : \sigma). \langle \tau_exp \rangle$$
$$\langle \sigma \times \tau_exp \rangle ::= \langle \langle \sigma_exp \rangle, \langle \tau_exp \rangle \rangle$$
$$\langle \sigma_exp \rangle ::= \langle \sigma_var \rangle |$$
$$\text{if } \langle \text{bool_exp} \rangle \text{ then } \langle \sigma_exp \rangle \text{ else } \langle \sigma_exp \rangle |$$
$$\langle \sigma_application \rangle | \langle \sigma_projection \rangle | \langle \sigma_fixed_point \rangle$$
$$\langle \sigma_application \rangle ::= \langle \tau \rightarrow \sigma_exp \rangle \langle \tau_exp \rangle$$
$$\langle \sigma_projection \rangle ::= \text{Proj}_1 \langle \sigma \times \tau_exp \rangle | \text{Proj}_2 \langle \tau \times \sigma_exp \rangle$$
$$\langle \sigma_fixed_point \rangle ::= Y_\sigma \langle \sigma \rightarrow \sigma_exp \rangle$$

Con $t : T$ indichiamo che il termine t è di tipo T

Example

- $(\underline{n} + \underline{m}) + \underline{n} : \text{Nat}$
- $\text{Eq?}(\underline{n})(\underline{m}) : \text{Bool}$
- $\langle \text{true}, \underline{n} \rangle : \text{Bool} \times \text{Nat}$
- $\text{Proj}_1 \langle \text{true}, \underline{n} \rangle : \text{Bool}$
- $\lambda(x : \text{Nat}).x + 1 : \text{Nat} \rightarrow \text{Nat}$ (indichiamolo con Succ)
- $\text{Succ}(\underline{n}) : \text{Nat}$
- $\text{if}[\text{Eq?}(\underline{n})(\underline{m})] \text{ then}[\underline{n}] \text{ else}[\text{Succ}]$ non è ben formato
- $\text{if}[\text{Eq?}(\underline{n})(\underline{m})] \text{ then}[\underline{n}] \text{ else}[\text{Succ}(\underline{n})] : \text{Nat}$
- $\lambda(x : \text{Nat}).\text{if}[\text{Eq?}(\underline{0})(x)] \text{ then}[\text{true}] \text{ else}[\text{false}] : \text{Nat} \rightarrow \text{Bool}$
(Indichiamolo con IsZero)
- $Y[\text{Succ}] : \text{Nat}$
- $Y[\text{IsZero}]$ non è ben formato

TODO

Regole di riduzione e Semantica operativa

- cos'è un programma di PCF [Done]
- Riduzione non deterministica (e relativa semantica) [Done]
- Proprietà di Church-Rosser [Done]
- Riduzione left-most [Ma potremmo farne a meno]

Programmi

Un programma di PCF è un termine:

- ben formato
- chiuso
- di tipo Nat o $Bool$ (tipi osservabili)

Example

- $Eq?(n)(m) : Bool$ è un programma
- $Y[Succ] : Nat$ è un programma
- $Succ : Nat \rightarrow Nat$ non è un programma (tipo non osservabile)
- $x + \underline{n} : Nat$ non è un programma (non è chiuso)

Semantica operativa

Diamo le seguenti regole di riduzione:

$$\text{add } \underline{n} + \underline{m} \rightarrow \underline{n + m}$$

$$\text{Eq? } \text{Eq?}(\underline{n})(\underline{n}) \rightarrow \text{true}$$

$$\text{Eq?}(\underline{n})(\underline{m}) \rightarrow \text{false} \text{ (per } n \text{ ed } m \text{ distinti)}$$

$$\text{cond } \text{if}[\text{true}] \text{ then}[M] \text{ else}[N] \rightarrow M$$

$$\text{if}[\text{false}] \text{ then}[M] \text{ else}[N] \rightarrow N$$

$$\text{proj } \text{Proj}_1 < M, N > \rightarrow M$$

$$\text{Proj}_2 < M, N > \rightarrow N$$

$$\alpha \quad \lambda(x : \sigma).M \rightarrow \lambda(y : \sigma).[y/x]M \text{ (con } y \text{ non libera in } M)$$

$$\beta \quad [\lambda(x : \sigma).M](N) \rightarrow [N/x]M$$

$$Y \quad Y_\sigma \rightarrow \lambda(f : \sigma \rightarrow \sigma).f(Y_\sigma f)$$

- Indichiamo con \twoheadrightarrow la chiusura transitiva della relazione \rightarrow
- Diciamo che un termine N è in forma normale se non può essere ridotto tramite le regole sopra introdotte
- Dato un termine M , diciamo che la sua valutazione rispetto alla semantica operativa è N se
 - N è in forma normale
 - $M \twoheadrightarrow N$

E lo indichiamo con $Eval(M) = N$

Theorem (Proprietà di Church-Rosser)

Se $M \twoheadrightarrow N_1$ e $M \twoheadrightarrow N_2$, allora esiste P tale che $N_1 \twoheadrightarrow P$ e $N_2 \twoheadrightarrow P$

Questo risultato assicura l'unicità della valutazione. Non sempre però un termine ha una forma normale, in questo caso scriviamo $Eval(M) = \perp$

Equivalenza osservazionale

Definiamo un *contesto* come un termine in cui compare un "buco" indicato con $[]$

Example

$$C[] \equiv \lambda(x : \text{Nat}).x + []$$

Porre il termine \underline{n} nel contesto $C[]$ significa considerare il termine

$$C[\underline{n}] \equiv \lambda(x : \text{Nat}).x + \underline{n}$$

Diciamo che due termini M ed N sono osservazionalmente equivalenti se per ogni contesto $C[]$ si ha $\text{Eval}(C[M]) = \text{Eval}(C[N])$ e lo indichiamo con $M \stackrel{\text{obs}}{=} N$

Espressività di PCF

Diciamo che una funzione parziale $f : \mathbb{N} \rightarrow \mathbb{N}$ è *calcolabile* se esiste un programma per computer* P tale che:

- Se $f(n) = m$, allora il programma P con input n termina con output m
- Se $f(n)$ non è definita, allora il programma P con input n non termina

* Con computer si intende una macchina a registri (URM); idealmente, un computer con infinita memoria

fatto

Non esiste un algoritmo per capire se un generico programma termini o meno

Fatto

Data una funzione parziale calcolabile f , esiste un termine di PCF t tale che

- Se $f(n) = m$, allora $Eval(t(\underline{n})) = \underline{m}$
- Se $f(n)$ non è definito, allora $Eval(t(\underline{n})) = \text{undef}$

Fatto

Non esiste un algoritmo per capire se, dato un generico termine di PCF questo ammetta una forma normale

Fatto

Non esiste un algoritmo per capire se due termini di PCF siano osservazionalmente equivalenti

Full Abstraction

Diciamo che un modello per PCF è Fully Abstract se e solo se per ogni coppia di termini M e N :

$$M \stackrel{\text{obs}}{=} N \Leftrightarrow \llbracket M \rrbracket = \llbracket N \rrbracket$$

Diciamo che un modello per PCF è intensionally fully abstract se:

- È algebrico
- Gli elementi compatti sono definibili in PCF

Teorema

Dato un modello \mathcal{I} intensionally fully abstract, esiste una relazione di equivalenza \approx tale che $\mathcal{E} = \mathcal{I} / \approx$ sia un modello fully abstract

A questo punto vorremmo un modello per PCF tale che:

- 1 Sia fully abstract
- 2 Il modello sia *definibile* (cioè ogni elemento del modello sia interpretazione di un termine di PCF)
- 3 Il modello sia *minimale* (cioè esista una "immersione" in ogni altro modello fully abstract)

Si può chiedere di piu? da sistemare

Si potrebbe richiedere che la denotazione fornisca algoritmi per decidere se due termini siano osservazionalmente equivalenti, almeno per i termini di FinitaryPCF (PCF costruito partendo dal solo tipo base *Bool*); un risultato di Loader ci dice che questo NON è possibile