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, $S \to T$ è un tipo

Example

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

Grammatica per generare i termini di PCF

$$< nat\_exp > ::= \underline{0}|\underline{1}|\underline{2}| \ldots | < nat\_exp > + < nat\_exp >$$
$$< bool\_exp > ::= true|false|Eq? < nat\_exp >< nat\_exp >$$

$$< \sigma \to \tau\_exp > ::= \lambda(x : \sigma). < \tau\_exp >$$
$$< \sigma \times \tau\_exp > ::=<< \sigma\_exp >, < \tau\_exp >>$$

$$\begin{aligned} < \sigma\_exp > ::=&< \sigma\_var > | \\ & if < bool\_exp > then < \sigma\_exp > else < \sigma\_exp > | \\ & < \sigma\_application > | < \sigma\_projection > | < \sigma\_fixed\_point > \\ < \sigma\_application > ::=&< \tau \to \sigma\_exp >< \tau\_exp > \\ < \sigma\_projection > ::=& Proj_1 < \sigma \times \tau\_exp > |Proj_2 < \tau \times \sigma\_exp > \\ < \sigma\_fixed\_point > ::=& Y_\sigma < \sigma \to \sigma\_exp > \end{aligned}$$

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

Example

- $(\underline{n} + \underline{m}) + \underline{n} : Nat$
- $Eq?(\underline{n})(\underline{m}) : Bool$
- $< true, \underline{n} >: Bool \times Nat$
- $Proj_1 < true, \underline{n} >: Bool$
- $\lambda(x : Nat).x + 1 : Nat \to Nat$ (indichiamolo con $Succ$)
- $Succ(\underline{n}) : Nat$
- $if[Eq?(\underline{n})(\underline{m})] \quad then[\underline{n}] \quad else[Succ]$ non è ben formato
- $if[Eq?(\underline{n})(\underline{m})] \quad then[\underline{n}] \quad else[Succ(\underline{n})] : Nat$
- $\lambda(x : Nat).if[Eq?(\underline{0})(x)] \quad then[true] \quad else[false] : Nat \to Bool$ (Indichiamolo con $IsZero$)
- $Y[Succ] : Nat$
- $Y[IsZero]$ non è ben formato

TODO

Regole di riduzione e Semantica operazionale

- 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

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

Semantica operazionale

Diamo le seguenti regole di riduzione:

$$
\begin{array}{rl}
\text{add} & \underline{n} + \underline{m} \rightarrow \underline{n+m} \\
\text{Eq?} & Eq?(\underline{n})(\underline{n}) \rightarrow true \\
 & Eq?(\underline{n})(\underline{m}) \rightarrow false \text{ (per } n \text{ ed } m \text{ distinti)} \\
\text{cond} & if[true] \quad then[M] \quad else[N] \rightarrow M \\
 & if[false] \quad then[M] \quad else[N] \rightarrow N \\
\text{proj} & Proj_1 < M, N > \rightarrow M \\
 & Proj_2 < M, N > \rightarrow N \\
\alpha & \lambda(x:\sigma).M \rightarrow \lambda(y:\sigma).[y/x]M \text{ (con } y \text{ non libera in } M) \\
\beta & [\lambda(x:\sigma).M](N) \rightarrow [N/x]M \\
Y & Y_\sigma \rightarrow \lambda(f:\sigma\rightarrow\sigma).f(Y_\sigma f)
\end{array}
$$

- 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 operazionale 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

$C[\,] \equiv \lambda(x : Nat).x + [\,]$
Porre il termine $\underline{n}$ nel contesto $C[\,]$ significa considerare il termine $C[\underline{n}] \equiv \lambda(x : Nat).x + \underline{n}$

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

Espressività di PCF

Diciamo che una funzione parziale $f : \mathbb{N} \to \mathbb{N}$ 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 $Eval(t(\underline{n})) = undef$

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:

$$M \overset{\text{obs}}{=} N \Leftrightarrow [\![M]\!] = [\![N]\!]$$

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 più? 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

Piano malefico:

- Definire un gioco e una strategia
- Definire i giochi che interessano
  - Gioco prodotto tensore
  - Gioco implicazione lineare
  - Gioco prodotto
  - Gioco "of course"
- Definire la categoria dei giochi (ergo parlare un po' di TdC)

I giochi

Il modello che andremo a considerare si basa sulla *teoria dei giochi*
Un gioco è una 4-upla $A = (M_A, \lambda_A, P_A, \approx_A)$ dove:

- M_A è l'insieme delle mosse
- λ_A è una funzione da M_A all'insieme $\{O, P\} \times \{Q, A\}$; in particolare:
  - O indica il giocatore "opponent" e P il giocatore "player"
  - Q indica una domanda e A una risposta
- Una partita è una stringa di mosse tale che:
  1. La prima mossa è di O
  2. P e O si alternano
  3. In ogni momento della partita, il numero di risposte deve essere al più uguale al numero di domande (*bracketing condition*)
- P_A è un sottoinsieme prefix-closed di partite; chiameremo P_A l'insieme delle partite valide

- $\approx_A$ è una relazione di equivalenza sulle partite valide tale che:
  - $s \approx_A t \Rightarrow \lambda_A^\star(s) = \lambda_A^\star(t)$
  - $s \approx_A t, s' \sqsubseteq s, t' \sqsubseteq t, |s| = |t| \Rightarrow s' \approx_A t'$
  - $s \approx_A t, sa \in P_A \Rightarrow \exists b.tb \in P_A \wedge sa \approx_A tb$

  DA DIRE IN MANIERA PIÙ UMANA

Example

I giochi *Nat* e *Bool*

Strategie

Una strategia σ è un insieme di partite di lunghezza pari (l'ultima mossa è di P) tali che:

- σ è prefix-closed (da aggiustare)
- $sab, tac \in \sigma \Rightarrow b = c$ (history free)
- $sab \in \sigma, ta \in P_A \Rightarrow tab \in \sigma$ (history free 2)

Estendiamo la relazione $\approx_A$ alle strategie Poniamo:

- $\sigma \preccurlyeq \tau$ iif $sab \in \sigma, s' \in \tau, sa \approx_A s'a' \Rightarrow \exists b'.s'a'b' \in \tau \wedge sab \approx_A s'a'b'$
- $\sigma \approx \tau$ iif $\sigma \preccurlyeq \tau \wedge \tau \preccurlyeq \sigma$

Fatto

$\preccurlyeq$ è un preordine sulle strategie; di conseguenza $\approx_A$ è una relazione di equivalenza parziale

Example

le strategie di *Nat* e *Bool*

Come rappresentiamo i giochi (il tavolo insomma)

Il gioco $A \otimes B$

Piano malefico:

- Mettere un po' di basi di TdC
- Sparare i CANNONI di TdC
- Definire $K_!(\mathcal{G})$
- Definire l'order enrichement su $K_!(\mathcal{G})$ e sparare un po' di proprietà (modello razionale)
- Dire perché è un modello (ergo interpretare i termini)
- Costruire una macchina che computi i costrutti sopra, li analizzi, poi li scarti e vada a sfracellarsi contro un muro