λ-calcoli e Sistema F: il progresso dell'astrazione

Carmine-Emanuele Cella

Università di Bologna

16 aprile 2009

Calcolare, astrarre o provare?

- Concetti interdipendenti

Calcolare, astrarre o provare?

- Concetti interdipendenti
- Ogni teoria adotta un punto di vista diverso

Calcolare, astrarre o provare?

- Concetti interdipendenti
- Ogni teoria adotta un punto di vista diverso
- È possibile costruire isomorfismi tra essi

Calcolare, astrarre o provare?

- Concetti interdipendenti
- Ogni teoria adotta un punto di vista diverso
- È possibile costruire isomorfismi tra essi
- La calcolabilità fornisce spunti costruttivi

Il λ-calcolo

- Descritto da Church nel 1936 per definire la nozione di *funzione effettivamente calcolabile*

Il λ-calcolo

- Descritto da Church nel 1936 per definire la nozione di *funzione effettivamente calcolabile*
- È il più semplice linguaggio di programmazione *interessante* (Turing-completo, di ordine superiore, con legami di variabili e scoping statico)

Il λ-calcolo

- Descritto da Church nel 1936 per definire la nozione di *funzione effettivamente calcolabile*
- È il più semplice linguaggio di programmazione *interessante* (Turing-completo, di ordine superiore, con legami di variabili e scoping statico)
- In tale linguaggio **tutto è una funzione**: le variabili denotano sempre funzioni, le funzioni hanno sempre altre funzioni per parametro, il risultato di una funzione è una funzione

Il λ-calcolo: formalismo

- Un'espressione del linguaggio è definitita dalla seguente grammatica:
  1. espressione := id | funzione | applicazione
  2. funzione := λ id . espressione (*astrazione*)
  3. applicazione := espressione espressione

Il λ-calcolo: formalismo

- Un'espressione del linguaggio è definitita dalla seguente grammatica:
  1. espressione := id | funzione | applicazione
  2. funzione := λ id . espressione (*astrazione*)
  3. applicazione := espressione espressione
- Ogni espressione può essere circondata dalle parentesi

Il λ-calcolo: formalismo

- Un'espressione del linguaggio è definitita dalla seguente grammatica:
  1. espressione := id | funzione | applicazione
  2. funzione := λ id . espressione (*astrazione*)
  3. applicazione := espressione espressione
- Ogni espressione può essere circondata dalle parentesi
- Le sole parole chiave del linguaggio sono λ e '.'

Il λ-calcolo: formalismo

- Un'espressione del linguaggio è definitita dalla seguente grammatica:
  1. espressione := id | funzione | applicazione
  2. funzione := λ id . espressione (*astrazione*)
  3. applicazione := espressione espressione
- Ogni espressione può essere circondata dalle parentesi
- Le sole parole chiave del linguaggio sono λ e '.'
- id è un nome qualsiasi di variabile

Il λ-calcolo: scope

- Un'espressione del tipo $\lambda x.y$ viene chiamata λ-astrazione

Il λ-calcolo: scope

- Un'espressione del tipo $\lambda x.y$ viene chiamata λ-astrazione
- La λ-astrazione $\lambda x.y$ *lega* la variabile x

Il λ-calcolo: scope

- Un'espressione del tipo $\lambda x.y$ viene chiamata λ-astrazione
- La λ-astrazione $\lambda x.y$ *lega* la variabile x
- Lo *scope* di questo legame è il corpo di y

Il λ-calcolo: scope

- Un'espressione del tipo $\lambda x.y$ viene chiamata λ-astrazione
- La λ-astrazione $\lambda x.y$ *lega* la variabile x
- Lo *scope* di questo legame è il corpo di y
- Le occorrenze di x dentro t sono dette **legate**

Il λ-calcolo: scope

- Un'espressione del tipo $\lambda x.y$ viene chiamata λ-astrazione
- La λ-astrazione $\lambda x.y$ *lega* la variabile x
- Lo *scope* di questo legame è il corpo di y
- Le occorrenze di x dentro t sono dette **legate**
- Le occorrenze di x che non sono nello scope di una λ-astrazione che lega x sono dette **libere**

Il λ-calcolo: semantica operazionale

- Regola di computazione:
  1. $(\lambda x.t)v \longrightarrow [x \mapsto v]t$
     (dove $[x \mapsto v]t$ è l'espressione che risulta dalla sostituzione delle occorrenze di x in t con v)

Il λ-calcolo: semantica operazionale

- Regola di computazione:
  1. $(\lambda x.t)v \longrightarrow [x \mapsto v]t$
     (dove $[x \mapsto v]t$ è l'espressione che risulta dalla sostituzione delle occorrenze di x in t con v)
- Regole di congruenza:
  1. $$\frac{t_1 \longrightarrow t_1'}{t_1\, t_2 \longrightarrow t_1' t_2}$$
  2. $$\frac{t_2 \longrightarrow t_2'}{v_1\, t_2 \longrightarrow v_1\, t_2'}$$

Il λ-calcolo: esempi (1)

- Sia **succ x** una funzione che aggiunge 1 al suo input **x**

Il λ-calcolo: esempi (1)

- Sia **succ x** una funzione che aggiunge 1 al suo input **x**
- Sia **piu3 x** una funzione che aggiunge 3 al suo input **x**; essa è definibile come segue: $piu3 = succ(succ(succx))$

Il λ-calcolo: esempi (1)

- Sia **succ x** una funzione che aggiunge 1 al suo input **x**
- Sia **piu3 x** una funzione che aggiunge 3 al suo input **x**; essa è definibile come segue: $piu3 = succ(succ(succx))$
- Dunque piu3 è una funzione che *produce* la funzione $succ(succ(succx))$; ovvero $piu3 = \lambda x.succ(succ(succx))$

Il λ-calcolo: esempi (1)

- Sia **succ x** una funzione che aggiunge 1 al suo input **x**
- Sia **piu3 x** una funzione che aggiunge 3 al suo input **x**; essa è definibile come segue: $piu3 = succ(succ(succx))$
- Dunque piu3 è una funzione che *produce* la funzione $succ(succ(succx))$; ovvero $piu3 = \lambda x.succ(succ(succx))$
- Essa è definibile **indipendentemente** dal nome che le si assegna: $piu3(succ0) = \lambda x.succ(succ(succx))(succ0)$

Il λ-calcolo: esempi (2)

- Sia $g = \lambda f.f(f(succ0))$ una λ-astrazione

Il λ-calcolo: esempi (2)

- Sia $g = \lambda f.f(f(succ0))$ una λ-astrazione
- Il parametro f è usato nel corpo di g in posizione di fuzione; le espressioni come g vengono dette di *ordine superiore*

Il λ-calcolo: esempi (2)

- Sia $g = \lambda f.f(f(succ0))$ una λ-astrazione
- Il parametro f è usato nel corpo di g in posizione di fuzione; le espressioni come g vengono dette di *ordine superiore*
- Applicando g a piu3, otteniamo la seguente produzione:

$$\begin{aligned} gpiu3 &= (\lambda f.f)f(succ0)))(\lambda x.succ(succ(succx))) \\ &= (\lambda x.succ(succ(succx)))((\lambda x.succ(succ(succx)))(succ0) \\ &= (\lambda x.succ(succ(succ(x)))(succ(succ(succ(succ0)))) \\ &= succ(succ(succ(succ(succ(succ(succ0)))))) \end{aligned}$$

Il λ-calcolo: proprietà

- **TEOREMA DI CHURCH-ROSSER**:
  la regola di riduzione del λ-calcolo è *confluente*
  (es. se $t \longrightarrow s_1$ e $t \longrightarrow s_2$ allora esiste un'espressione u tale che $s_1 \longrightarrow u$ e $s_2 \longrightarrow u$)
- **COROLLARIO**:
  il λ-calcolo è algoritmicamente coerente

Il λ-calcolo tipato ($\lambda^{\rightarrow}$)

- Descritto da Church nel 1940 per cercare di risolvere alcuni usi paradossali del λ-calcolo non tipato

Il λ-calcolo tipato ($\lambda^{\rightarrow}$)

- Descritto da Church nel 1940 per cercare di risolvere alcuni usi paradossali del λ-calcolo non tipato
- La tipizzazione non serve a garantire la coerenza del calcolo (già dimostrata sopra) bensì la *terminazione*

Il λ-calcolo tipato ($\lambda^{\rightarrow}$)

- Descritto da Church nel 1940 per cercare di risolvere alcuni usi paradossali del λ-calcolo non tipato
- La tipizzazione non serve a garantire la coerenza del calcolo (già dimostrata sopra) bensì la *terminazione*
- I tipi utilizzati dal calcolo (e le sue estensioni come prodotti e numeri naturali) sono detti *semplici* in contrasto con i tipi *polimorfici* ed i tipi *dipendenti* utilizzati da calcoli più espressivi

Il λ-calcolo tipato: formalismo (1)

- **Variabile**: una variabile x^A, y^A, z^A è un termine di tipo A

Il λ-calcolo tipato: formalismo (1)

- **Variabile**: una variabile x^A, y^A, z^A è un termine di tipo A
- λ-**astrazione**: se t è un termine di tipo B allora $\lambda x^A.t$ è un termine di tipo (A $\Rightarrow$ B)

Il λ-calcolo tipato: formalismo (1)

- **Variabile**: una variabile x^A, y^A, z^A è un termine di tipo A
- λ-**astrazione**: se t è un termine di tipo B allora $\lambda x^A.t$ è un termine di tipo (A $\Rightarrow$ B)
- **Applicazione**: se t e u sono termini di tipo rispettivamente (A $\Rightarrow$ B) e A allora $t(u)$ è un termine di tipo B

Il λ-calcolo tipato: formalismo (1)

- **Variabile**: una variabile x^A, y^A, z^A è un termine di tipo A
- λ-**astrazione**: se t è un termine di tipo B allora $\lambda x^A.t$ è un termine di tipo (A $\Rightarrow$ B)
- **Applicazione**: se t e u sono termini di tipo rispettivamente (A $\Rightarrow$ B) e A allora $t(u)$ è un termine di tipo B
- La regola di computazione diventa: $(\lambda x^A.t)v \longrightarrow [x \mapsto v]t$ di cui si può confermare la proprietà di confluenza

Il λ-calcolo tipato: formalismo (1)

- **Variabile**: una variabile x^A, y^A, z^A è un termine di tipo A
- λ-**astrazione**: se t è un termine di tipo B allora $\lambda x^A.t$ è un termine di tipo (A $\Rightarrow$ B)
- **Applicazione**: se t e u sono termini di tipo rispettivamente (A $\Rightarrow$ B) e A allora $t(u)$ è un termine di tipo B
- La regola di computazione diventa: $(\lambda x^A.t)v \longrightarrow [x \mapsto v]t$ di cui si può confermare la proprietà di confluenza
- Si possono definire anche il tipo *somma* $X + Y = \lambda x^{n+1}.\lambda y^n.((X)x)((Y)x)y$ ed il tipo *prodotto* $X \times Y = \lambda x^{n+1}.(X)(Y)x$

Il λ-calcolo tipato: formalismo (2)

- **TEOREMA DI NORMALIZZAZIONE FORTE**:
  un termine di t è fortemente normalizzabile (fN) se il $sup|t|$ delle lunghezze di tutte le sequenze di riduzione immediate che partono da t è finito
  (ovvero: tutte le sequenze di riduzione sono finite)

Il λ-calcolo tipato: formalismo (2)

- **TEOREMA DI NORMALIZZAZIONE FORTE**:
  un termine di t è fortemente normalizzabile (fN) se il $sup|t|$ delle lunghezze di tutte le sequenze di riduzione immediate che partono da t è finito
  (ovvero: tutte le sequenze di riduzione sono finite)
- Dato che il linguaggio $\lambda^{\rightarrow}$ è fortemente normalizzabile, diventa decidibile se un *programma* di tale linguaggio arrivi o no al termine

Il λ-calcolo tipato: formalismo (2)

- **TEOREMA DI NORMALIZZAZIONE FORTE**:
  un termine di t è fortemente normalizzabile (fN) se il $sup|t|$ delle lunghezze di tutte le sequenze di riduzione immediate che partono da t è finito
  (ovvero: tutte le sequenze di riduzione sono finite)
- Dato che il linguaggio $\lambda^{\rightarrow}$ è fortemente normalizzabile, diventa decidibile se un *programma* di tale linguaggio arrivi o no al termine
- Ciò porta a concludere che il linguaggio **non è Turing-completo**

Il λ-calcolo tipato: l'isomorfismo CH

- L'isomorfismo di **Curry-Howard** enuncia la completa e totale equivalenza tra i seguenti calcoli:
  1. *deduzione naturale (NJ)*: enunciati A, deduzioni di A, normalizzazione in deduzione naturale
  2. *λ-calcolo tipato*: tipi A, termini di tipo A, normalizzazione forte

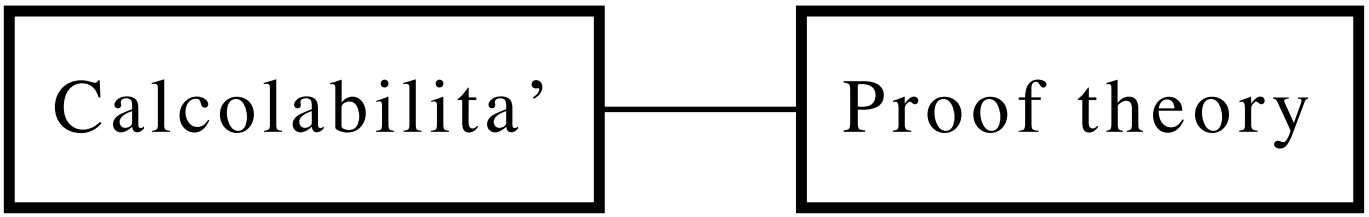
Il λ-calcolo tipato: l'isomorfismo CH

- L'isomorfismo di **Curry-Howard** enuncia la completa e totale equivalenza tra i seguenti calcoli:
  1. *deduzione naturale (NJ)*: enunciati A, deduzioni di A, normalizzazione in deduzione naturale
  2. *λ-calcolo tipato*: tipi A, termini di tipo A, normalizzazione forte
- Per operare l'isomorfismo è necessario adottare il punto di vista *locativo* di De Bruijn: ogni variabile dello stesso tipo viene indicizzata e considerata come istanza diversa; ovvero: **per ogni tipo esiste solo una variabile**

Il λ-calcolo tipato: l'isomorfismo CH

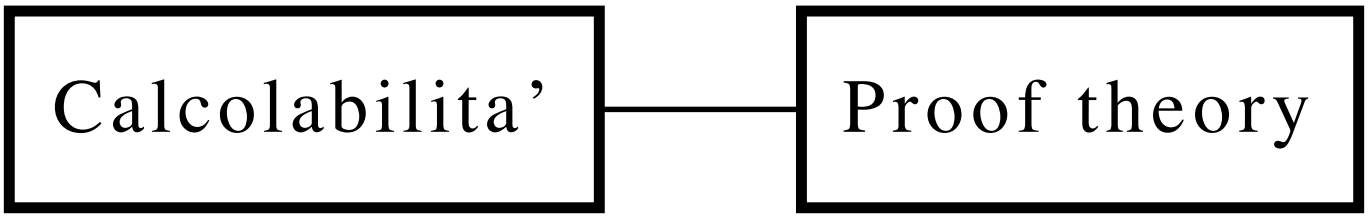
- L'isomorfismo di **Curry-Howard** enuncia la completa e totale equivalenza tra i seguenti calcoli:
  1. *deduzione naturale (NJ)*: enunciati A, deduzioni di A, normalizzazione in deduzione naturale
  2. *λ-calcolo tipato*: tipi A, termini di tipo A, normalizzazione forte
- Per operare l'isomorfismo è necessario adottare il punto di vista *locativo* di De Bruijn: ogni variabile dello stesso tipo viene indicizzata e considerata come istanza diversa; ovvero: **per ogni tipo esiste solo una variabile**
- L'isomorfismo CH rende questo calcolo molto vicino alla logica intuizionista, mediante l'uso del solo connettivo di implicazione $\longrightarrow$

Calcolare è provare



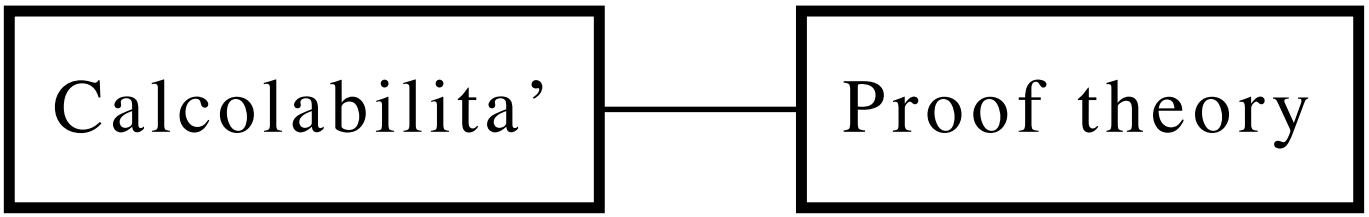
- L'equivalenza tra calcolo della deduzione naturale e λ-calcolo tipato completa una parte del diagramma iniziale

Calcolare è provare



- L'equivalenza tra calcolo della deduzione naturale e λ-calcolo tipato completa una parte del diagramma iniziale
- La relazione tra programmi per computer e prove matematiche mostra l'analogia sintattica tra sistemi logici formali e sistemi computazionali

Calcolare è provare



- L'equivalenza tra calcolo della deduzione naturale e λ-calcolo tipato completa una parte del diagramma iniziale
- La relazione tra programmi per computer e prove matematiche mostra l'analogia sintattica tra sistemi logici formali e sistemi computazionali
- È possibile completare anche la parte rimanente del diagramma?

Il Sistema F

- Descritto indipendentemente da Girard nel 1970 e da Reynolds nel 1974

Il Sistema F

- Descritto indipendentemente da Girard nel 1970 e da Reynolds nel 1974
- Motivazioni molto diverse:
  1. Girard: interpretazione della logica del secondo ordine
  2. Reynolds: programmazione funzionale

Il Sistema F

- Descritto indipendentemente da Girard nel 1970 e da Reynolds nel 1974
- Motivazioni molto diverse:
  1. Girard: interpretazione della logica del secondo ordine
  2. Reynolds: programmazione funzionale
- Ha avuto influenze profonde sugli sviluppo delle teorie dei tipi (Martin-Löf, calcolo delle costruzioni di Coquand, ecc.)

Il Sistema F: formalismo (1)

- Il linguaggio si ottiene come generalizzazione del calcolo $\lambda^{\rightarrow}$ mediante l'aggiunta di una operazione di *astrazione sui tipi*

Il Sistema F: formalismo (1)

- Il linguaggio si ottiene come generalizzazione del calcolo $\lambda^{\rightarrow}$ mediante l'aggiunta di una operazione di *astrazione sui tipi*
- I *tipi* sono definiti partendo da *variabili tipo* X, Y, Z, ... attraverso due operazioni:
  1. se U e V sono tipi, allora $U \rightarrow V$ è un tipo
  2. se V è un tipo e X una variabile tipo, allora $\prod X.V$ è un tipo

Il Sistema F: formalismo (2)

- Esistono 5 schemi per la creazione dei *termini*:
  1. **variabili**: $x_T, y^T, z^T, \ldots$ di tipo T
  2. **applicazione**: $t(u)$ di tipo V, dove t è il tipo di $U \longrightarrow V$ e u è di tipo U
  3. λ-**astrazione**: $\lambda x^U.v$ di tipo $U \longrightarrow V$, dove x^U è una variabile di tipo U e v è di tipo V
  4. **astrazione universale**: se v è un termine di tipo V, allora è possibile formare $\Lambda X.v$ di tipo $\prod X.V$ in modo che la variabile X non sia libera nel tipo di una variabile libera v
  5. **applicazione universale (estrazione)**: se t è un termine di tipo $\prod X.V$ e U è un tipo, allora $t(U)$ è un termine di tipo $V[U \longrightarrow X]$

Il Sistema F: semantica operazionale

- Regola di produzione:

$$(\Lambda X.t)U \longrightarrow [X \mapsto U]t$$

Il Sistema F: semantica operazionale

- Regola di produzione:

$$(\Lambda X.t)U \longrightarrow [X \mapsto U]t$$

- Anche in questo caso vale il teorema di Church-Rosser e dunque il calcolo è coerente

Il Sistema F: semantica operazionale

- Regola di produzione:

$$(\Lambda X.t)U \rightarrow [X \mapsto U]t$$

- Anche in questo caso vale il teorema di Church-Rosser e dunque il calcolo è coerente
- È possibile formare $\Lambda X.\lambda x^X.x^X$ di tipo $\prod X.X \rightarrow X$, che rappresenta l'identità di ogni tipo

Il Sistema F: commenti

- L'interpretazione näive del tipo $\prod$ è che un oggetto di tipo $\prod X.V$ è una funzione che ad ogni tipo U associa un oggetto di tipo $[X \mapsto U]V$

Il Sistema F: commenti

- L'interpretazione näive del tipo $\prod$ è che un oggetto di tipo $\prod X.V$ è una funzione che ad ogni tipo U associa un oggetto di tipo $[X \mapsto U]V$
- L'operatore Λ rappresenta l'astrazione del linguaggio: permette di calcolare non su funzioni ma su *tipi* di funzioni, portando i tipi ad oggetti *first-class*

Il Sistema F: commenti

- L'interpretazione näive del tipo $\prod$ è che un oggetto di tipo $\prod X.V$ è una funzione che ad ogni tipo U associa un oggetto di tipo $[X \mapsto U]V$
- L'operatore Λ rappresenta l'astrazione del linguaggio: permette di calcolare non su funzioni ma su *tipi* di funzioni, portando i tipi ad oggetti *first-class*
- Esso agisce in effetti come **quantificatore universale** di secondo ordine

Il Sistema F: commenti

- L'interpretazione näive del tipo $\prod$ è che un oggetto di tipo $\prod X.V$ è una funzione che ad ogni tipo U associa un oggetto di tipo $[X \mapsto U]V$
- L'operatore Λ rappresenta l'astrazione del linguaggio: permette di calcolare non su funzioni ma su *tipi* di funzioni, portando i tipi ad oggetti *first-class*
- Esso agisce in effetti come **quantificatore universale** di secondo ordine
- Il Sistema F è dunque un calcolo tipato del secondo ordine e viene spesso chiamato $\lambda 2$

Il Sistema F: esempi (1)

- Tipo **Booleano**: $Bool \triangleq \prod X.X \to X \to X$, dove:
  1. $True \triangleq \Lambda X.\lambda x^X.\lambda y^x.x$
  2. $False \triangleq \Lambda X.\lambda x^X.\lambda y^x.y$
  3. $D(u(v(t))) \triangleq t(U(u(v)))$ con u, v, t rispettivamente di tipo $U, U, Bool$

Il Sistema F: esempi (1)

- Tipo **Booleano**: $Bool \triangleq \prod X.X \to X \to X$, dove:
  1. $True \triangleq \Lambda X.\lambda x^X.\lambda y^x.x$
  2. $False \triangleq \Lambda X.\lambda x^X.\lambda y^x.y$
  3. $D(u(v(t))) \triangleq t(U(u(v)))$ con u, v, t rispettivamente di tipo $U, U, Bool$
- Calcolo di un caso *True*:

$$\begin{aligned} D(u(v(T))) &= (\Lambda X.\lambda x^X.\lambda y^X.x)Uuv \\ &= (\lambda x^U.\lambda y^U.x)uv \\ &= (\lambda y^U.u)v \\ &= u \end{aligned}$$

Il Sistema F: esempi (2)

- Calcolo di un caso *False*:

$$\begin{aligned} D(u(v(F))) &= (\Lambda X.\lambda x^X.\lambda y^X.y)Uuv \\ &= (\lambda x^U.\lambda y^U.y)uv \\ &= (\lambda y^U.y^U)v \\ &= v \end{aligned}$$

Il Sistema F: esempi (2)

- Calcolo di un caso *False*:

$$\begin{aligned} D(u(v(F))) &= (\Lambda X.\lambda x^X.\lambda y^X.y)Uuv \\ &= (\lambda x^U.\lambda y^U.y)uv \\ &= (\lambda y^U.y^U)v \\ &= v \end{aligned}$$

- Tipo **Intero**: $Int \triangleq \prod X.X \to (X \to X) \to X$

Il Sistema F: esempi (2)

- Calcolo di un caso *False*:

$$\begin{aligned} D(u(v(F))) &= (\Lambda X.\lambda x^X.\lambda y^X.y)Uuv \\ &= (\lambda x^U.\lambda y^U.y)uv \\ &= (\lambda y^U.y^U)v \\ &= v \end{aligned}$$

- Tipo **Intero**: $Int \triangleq \prod X.X \to (X \to X) \to X$
- L'intero n viene rappresentato dalle n ricorrenze di: $\Lambda X.\lambda x^X.\lambda t^{X\to X}.y(y(y\ldots(yx)\ldots))$

Il Sistema F: esempi (2)

- Calcolo di un caso *False*:

$$\begin{aligned} D(u(v(F))) &= (\Lambda X.\lambda x^X.\lambda y^X.y)Uuv \\ &= (\lambda x^U.\lambda y^U.y)uv \\ &= (\lambda y^U.y^U)v \\ &= v \end{aligned}$$

- Tipo **Intero**: $Int \triangleq \prod X.X \to (X \to X) \to X$
- L'intero n viene rappresentato dalle n ricorrenze di: $\Lambda X.\lambda x^X.\lambda t^{X\to X}.y(y(y\ldots(yx)\ldots))$
- È possibile rappresentare anche *List*, *Tree*, ...

Il Sistema F: l'isomorfismo CH

- I tipi nel Sistema F non sono altro che proposizioni quantificate al secondo ordine

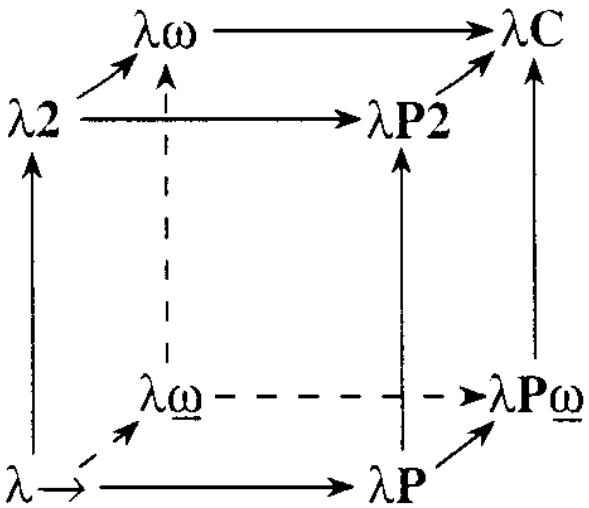
Il Sistema F: l'isomorfismo CH

- I tipi nel Sistema F non sono altro che proposizioni quantificate al secondo ordine
- L'isomorfismo CH già mostrato per il caso $\lambda^{\rightarrow}$ si estende dunque al caso $\lambda 2$; le regole:

$$\frac{A}{\forall X.A}\forall^2 I \qquad \frac{\forall X.A}{[X \mapsto B]A}\forall^2 E$$

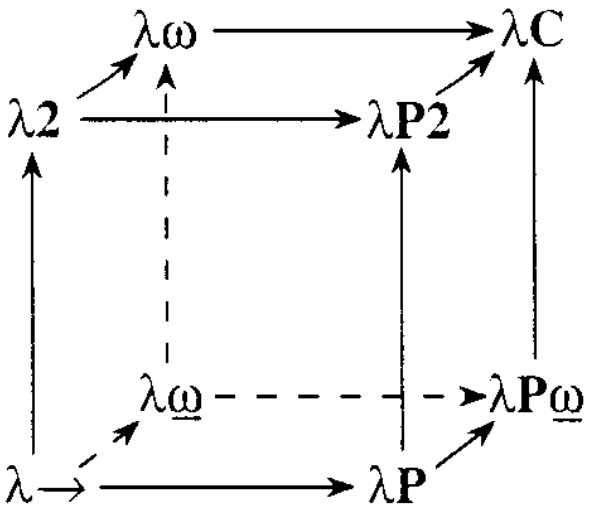
corrispondono esattamente alla astrazione ed alla applicazione universali

Il λ-cubo



- Framework proposto da H. Barendregt (1991) per inquadrare il **progresso dell'astrazione** nel calcolo

Il λ-cubo



- Framework proposto da H. Barendregt (1991) per inquadrare il **progresso dell'astrazione** nel calcolo
- Ogni spigolo del cubo rappresenta la relazione d'inclusione

Il λ-cubo: i calcoli

- La maggior parte degli 8 λ-calcoli è nota:
  1. $\lambda^{\rightarrow}$ è il calcolo tipato semplice
  2. $\lambda 2$ è il calcolo polimorfico del secondo ordine (Sistema F)
  3. $\lambda\omega$ è il Sistema Fω di Girard
  4. λP è il linguaggio per la dimostrazione AUTOMATH
  5. $\lambda P2$ è un sistema studiato da Longo e Moggi (1988)
  6. λC è il calcolo delle costruzioni di Coquand e Houet (1986)
  7. $\lambda\underline{\omega}$ è legato al sistema POLYRET di R. de Lavalette (1985)

Il λ-cubo: i calcoli

- La maggior parte degli 8 λ-calcoli è nota:
  1. $\lambda^{\rightarrow}$ è il calcolo tipato semplice
  2. $\lambda 2$ è il calcolo polimorfico del secondo ordine (Sistema F)
  3. $\lambda\omega$ è il Sistema Fω di Girard
  4. λP è il linguaggio per la dimostrazione AUTOMATH
  5. $\lambda P2$ è un sistema studiato da Longo e Moggi (1988)
  6. λC è il calcolo delle costruzioni di Coquand e Houet (1986)
  7. $\lambda\underline{\omega}$ è legato al sistema POLYRET di R. de Lavalette (1985)
- $\lambda P\underline{\omega}$ non è ancora stato studiato compiutamente

Il λ-cubo: gli assi e l'astrazione

- Ogni asse rappresenta una nuova forma di astrazione, in base alla dipendenza:
  1. **X**: tipi che dipendono su termini, o *tipi dependenti* (λP)
  2. **Y**: termini che dipendono su tipi, o *polimorfismo* (Sistema F)
  3. **Z**: tipi che dipendono su tipi, o *type operators* (Sistema Fω)

Il λ-cubo: gli assi e l'astrazione

- Ogni asse rappresenta una nuova forma di astrazione, in base alla dipendenza:
  1. **X**: tipi che dipendono su termini, o *tipi dependenti* (λP)
  2. **Y**: termini che dipendono su tipi, o *polimorfismo* (Sistema F)
  3. **Z**: tipi che dipendono su tipi, o *type operators* (Sistema Fω)
- Per tutti i calcoli del λ-cubo vale il teorema di Church-Rosser e dunque sono **fortemente normalizzabili**

Il λ-cubo: gli assi e l'astrazione

- Ogni asse rappresenta una nuova forma di astrazione, in base alla dipendenza:
  1. **X**: tipi che dipendono su termini, o *tipi dependenti* (λP)
  2. **Y**: termini che dipendono su tipi, o *polimorfismo* (Sistema F)
  3. **Z**: tipi che dipendono su tipi, o *type operators* (Sistema Fω)
- Per tutti i calcoli del λ-cubo vale il teorema di Church-Rosser e dunque sono **fortemente normalizzabili**
- Il λ-cubo è stato generalizzato da Berardi (1988) e Terlouw (1988) indipendentemente; ciò ha introdotto la nozione di sistema generalizzato dei tipi (GTS)

Conclusioni

- L'isomorfismo CH permette di correlare la calcolabilità alla proof-theory

Conclusioni

- L'isomorfismo CH permette di correlare la calcolabilità alla proof-theory
- La normalizzazione forte CR permette di creare *relazioni di inclusione* tra i calcoli (λ-cubo) e dunque di correlare il concetto di astrazione a quello di calcolabilità

Bibliografia

- **J. Backus**, Can programming be liberated from the von Neumann style? A functional style and its algebra of programs
- **H. Barendregt**, Lambda calculi with types
- **H. Barendregt**, Introduction to generalized type systems
- **T. Coquand - G. Houet**, Introduction to calculus of costructions
- **J.I. Girard**, Proofs and types
- **J.I. Girard**, Le point aveugle, vol. 1