Informatica: salto nel vuoto? No, salviamo la vita sulla terra unificando il linguaggio umano con quello degli algoritmi

Chiarimenti riguardanti il titolo

Il titolo è solo apparentemente favoloso. Il "salto nel vuoto" si riferisce al senso di spaesamento generato dai continui balzi in avanti, verso la prossima futura applicazione, che l'informatica ha compiuto dagli inizi. La sua popolarità è in continuo aumento.

Infatti l'informatica oggi facilita la pressoché istantanea comunicazione di ogni tipo di evento interessante l'umanità, sia globalmente che individualmente. Scienziati, industriali e commercianti annunciano continuamente ed offrono delle novità, la cui importanza è molto difficile da valutare, in quanto riguardanti sia nuovi rimedi, sia pericoli da evitare (se possibile) o svaghi con immediato godimento.

Riassumendo, la domanda riflette la naturale sorpresa che ciascuno di noi prova nel cercare di definire il preciso significato della parola *informatica*, oggi; ne proporrò uno nel prossimo paragrafo.

L'espressione "*salviamo la vita sulla terra*" si riferisce alla possibile soluzione di problemi attualissimi, quale il riscaldamento globale o al raggiungimento di obiettivi auspicabili ma al momento apparentemente fuori dalla nostra portata quali *sviluppo sostenibile, economia verde, fonti di energia rinnovabili.*

L'espressione "*unificando il linguaggio umano con quello degli algoritm*i" esprime in modo molto conciso la comune essenza del metodo risolutivo di ciascuno dei 4 problemi appena enunciati.

Significato dei vocaboli "Informatica", "computer" e "algoritmo"

Vogliamo utilizzare subito un prodotto del nuovo millennio destinato alla

definizione delle parole e quindi utile strumento nel processo della comunicazione: lo Zingarelli 2008 VOCABOLARIO DELLA LINGUA ITALIANA, che definisce

Informatica: traduzione dal francese (1968): scienza e tecnica dell'elaborazione dei dati

*comput*er: vocabolo introdotto nel 1966 traduzione inglese di calcolatore

algoritmo: vocabolo derivante dal nome del matematico arabo al-Huwarizmi (secolo IX) e significante: Procedimento per la soluzione d'un problema.

Dubbio: ma il titolo…scotta?

Se dunque l'algoritmo è definito come "procedimento per la soluzione d'un problema" e la proposta "Unifichiamo il linguaggio umano con quello degli algoritmi" pone chiaramente un problema da risolvere, ci chiediamo immediatamente: esisterà davvero un algoritmo per la soluzione di tale problema? La minaccia è che esso non abbia mai termine ovvero che tale problema non abbia soluzione.

Ciò succede spesso quando un'entità ha a che fare con sé stessa, come vedremo. Tuttavia non dobbiamo disperare. In certi casi, neppure se abbiamo la dimostrazione dell'inesistenza della soluzione, un teorema di Gödel ci viene in soccorso, come ben presto sarà chiarito. Si deve, in qualche modo, riuscire ad interrogare il computer.

L'importanza della domanda rispetto alla risposta

Può succedere che per alcune domande, poste da un ricercatore, non esista una risposta (e che ciò possa addirittura essere dimostrabile!). A questo punto l'alternativa che si pone al ricercatore è allargare o modificare il contesto in cui la domanda viene formulata aggiungendo almeno un grado di libertà.

Questo è ciò che è accaduto quando, nel 1931, il logico matematico Kurt Gödel sconvolse il mondo dei suoi colleghi col "teorema di incompletezza" che ammetteva l'esistenza di teoremi "veri" ma non dimostrabili. Questo teorema non è da considerare una sconfitta, come invece venne intesa da una parte della comunità dei logici, primo fra tutti Russell, che ritenne opportuno creare la teoria dei tipi per considerare applicabili solo i teoremi dimostrabili. Gödel invece, formulando questo teorema, ha sfruttato i paradossi aprendo vie illimitate alla creatività di nuovi risultati formali in logica matematica e quindi applicabili in informatica.

Nel caso del dialogo "uomo-macchina" è interessante ricordare come questo si sia evoluto negli anni e richieste sempre più esigenti abbiano ricevuto "risposte" tecnologiche sempre più complesse ed elaborate. Si è infatti passati, dalla possibilità di generare le tavole di tiro balistico, nella prima decina d'anni dell'uso del computer, alla capacità di manipolare immagini prima bidimensionali e poi tridimensionali fino a giungere, nell'attualità, all'invenzione della cosiddetta "stampante tridimensionale", in grado perfino di "modellare" oggetti reali.

Il logico A. M. Turing inventore dell'omonima macchina, sfruttandone l'universalità ha cercato di aiutare la fantasia dei logici; ma, a mia conoscenza, il suo tentativo ha riscosso risultati concreti non estrapolabili, purtroppo, dal punto di vista metodologico.

Abuso dei temini scientifici

La diffusione della terminologia scientifica per descrivere fenomeni universali ha purtroppo sollecitato anche il suo abuso. Esistono discipline, quali ad esempio l'antroposofia di Rudolf Steiner, che importano termini e concetti scientifici in

contesti morali e religiosi. Già nel 1957 uscì il famoso libro di Martin Gardner "Fads and Fallacies in the Name of Science" (Manie e credenze errate in nome della Scienza) con il pungente sottotitolo: "The curious theories of modern pseudoscientists and the strange amusing and alarming cults that surround them. A study in human gullibility" (Le curiose teorie dei pseudoscienziati moderni e gli strani, divertenti ma allarmanti, culti che le circondano. Uno studio della credulità umana).

Una elegante difesa della scienza (in questo caso astrofisica) a sostegno del libro di Gardner è la teoria dei molti universi, dovuta all'astrofisico Dennis Sciama (18 Nov 1926-18 Nov 1999); riporto una sua frase tratta da un discorso pronunciato negli anni Novanta (teatro Eliseo di Roma):

"L'universo che conosciamo è in armonia con la nascita della vita, con l'evoluzione dell'uomo e della sua intelligenza. Tutti i parametri cosmologici, astronomici, fisici, e chimici ci appaiono modulati in funzione della nostra specie. Il caso? La mano di Dio? Io preferisco credere che il nostro sia soltanto uno degli infiniti universi esistenti, ciascuno con caratteristiche sue proprie, e inaccessibili tra loro. In questo nostro universo si è formato l'uomo. In altri universi, forse, esistono creature diversissime da noi".

La precedente frase appare come la giusta reazione d'un eccellente cosmologo che elimina d'un sol colpo le trascendenze prive di carattere matematico, fisico, chimico o di scienze naturali.

Incompatibilità fra computer e infinito

Per quanto riguarda l'informatica e sopratutto la persona dell'informatico, che vuole trasformare nel modo più semplice, ma anche più generale, le informazioni ricavate da esperimenti in procedure previsionali, l'ipotesi più minacciosa di Sciama è l'infinità degli universi. È abbastanza intuitivo che un computer, per condurre a termine una procedura, deve assolutamente evitare

sia l'infinità della durata di un calcolo sia l'infinità degli oggetti con cui venire a contatto. In questa parte dell'articolo non pretendiamo che il lettore conosca i formalismi che saranno descritti nei prossimi paragrafi. Possiamo però sottolineare la forte analogia fra avvenimenti della storia dell'informatica teorica nel trentennio 1950-1980 e situazioni alquanto particolari e preoccupanti che possono accadere durante la nostra stessa vita.

Riprendiamo dunque il problema del computer che deve evitare sia l'infinità della durata del calcolo sia l'infinità degli oggetti contattati sui quali deve operare.

Una soluzione brillante, ma pericolosa, è che il computer dia la precedenza a degli oggetti ad esso più vicini. Nulla è più vicino ad esso che una sua parte. Ciò appare molto banale. La cosa lo è molto meno se la confrontiamo appunto con certe azioni della nostra vita. Operare su una parte di noi stessi può essere molto pericoloso, per noi. Suicidio?

Non necessariamente: se applichiamo tale tattica ai nostri simili, i più vicini a noi o di cui ci sentiamo responsabili, possiamo causare uno dei seguenti fenomeni:

Fuoco amico

Conflitto di interessi

Scegliere se stesso come avvocato difensore in una causa in cui si è uno fra gli imputati.

Tali fenomeni sembrano possedere un inevitabile carattere paradossale.

Ebbene in un paragrafo successivo, necessariamente più pragmatico dell'attuale, condivideremo con il lettore il piacere di sormontare i citati pericoli e difficoltà senza dovere imporre alcuna disciplina di tipi o simili restrizioni!

Abbiamo appena accennato a dei particolari problemi che il computer deve poter risolvere per potere soddisfare le pretese descritte dal titolo di questo articolo. Non possiamo dimenticare di citare la più importante di esse. Il

computer ci deve aiutare a risolvere i quattro problemi citati che contribuiscono alla soluzione di salvare la vita sulla terra. Il computer deve poter assimilare ed utilizzare le conoscenze sul binomio vita-morte in biologia. E' quindi saggio imitare la natura. Essa si avvale della procedura chiamata apoptosi, equivalente alla condanna a morte di una cellula (per motivi diversi) pianificata dal genoma dell'organismo. Siamo arrivati dunque alla seguente domanda...

Qual è l'apoptosi informatica?

Questa domanda, che sembra rasentare il non-senso perché unisce due parole fra loro estranee, ha invece un'importante motivazione.

E' ipotizzabile che in un prossimo futuro si potranno utilizzare simulazioni informatico-teoriche (in aggiunta alle simulazioni basate sulla struttura molecolare che già usano quotidianamente la tecnologia informatica) per prevedere e descrivere l'azione di nuovi farmaci su sistemi viventi limitando l'uso di cavie animali.

Lo studio delle cellule umane ed animali in vitro permette di seguire e descrivere i meccanismi che regolano la vita, il funzionamento e la morte di tali cellule ampliando le nostre conoscenze sulle azioni che i geni umani compiono da centinaia di milioni di anni aggiornate dall'effetto della selezione naturale. Fra questi meccanismi l'apoptosi, che sicuramente riveste un ruolo centrale nell'equilibrio dell'organismo vivente, potrebbe essere definita da una formula che descrivesse il trattamento e l'eliminazione degli elementi dannosi con strumenti informatici anziché biologici.

L'eliminazione/soppressione è infatti un'azione fondamentale anche nella logica combinatoria; anticipiamo qui che il combinatore K (strumento logico che verrà definito nei prossimi capitoli) ha appunto una funzione di killer!

Possiamo stabilire un'altra analogia fra strumenti biologici e strumenti logici, è il caso del NGF (fattore di crescita delle cellule nervose la cui scoperta valse il

Nobel a Rita Levi-Montalcini) e il funzionale. Questo concetto, che verrà approfondito in seguito, ha la caratteristica di potersi applicare, quindi agire, su diversi oggetti, funzioni e perfino su se stesso. Nell'ultimo decennio gli studi sull'NGF indicano che la sua azione non è limitata a quella di fattore di crescita ma avrebbe un ruolo importante in diversi campi della patologia umana. In un recente studio (1) i ricercatori hanno trovato che l'NGF tiene normalmente bloccata nelle cellule nervose la produzione di un peptide che è il principale responsabile della malattia di Alzheimer.

Scopo di questo articolo è dare il giusto peso al principale progresso scientifico che si è prodotto in questa prima decade del 21° secolo, proprio grazie all'enorme sviluppo dei computer e dell'informatica (hardware e software), sviluppo tutto diretto alla soluzione di problemi riguardanti l'umanità fra i quali spiccano quelli biologici. Alla fine di questo paragrafo si cerca di comprovare quanto affermato finora.
Per avere un'idea quantitativa di ciò che è stato pubblicato ad oggi, nel mondo, nelle varie discipline, useremo una potente innovazione di questo secolo: Google, il più noto motore di ricerca di informazioni. Si sono cercati i termini inglesi che descrivono alcune discipline scientifiche (prima lista) e quelli che descrivono aree di studio di carattere informatico (seconda lista). Google, per ogni termine (o coppia di termini o locuzioni) cercato, riporta il numero di pagine in inglese che, in rete, contengono quel termine (o coppia di termini o locuzioni).
Di seguito sono riportati i risultati relativi ad ogni termine cercato.

Prima lista

Engineering	274 000 000
Psychology	105 000 000

Mathematics	97 700 000
Chemistry	93 700 000
Physics	90 300 000
Biology	75 500 000
Logics	2 540 000

Seconda lista

Nanotechnology	8 780 000
Bioinformatics	7 640 000
"Quantum computing"	623 000
"Theoretical Biology"	316 000
"DNA computing"	72 500

Prima lista. Engineering è un vocabolo con significato generico e non necessariamente scientifico quindi la sua predominanza non deve stupire. E' interessante che Psychology predomini su Mathematics, Chemistry, Physics, Biology, e Logics: viva la salute mentale!

Seconda lista. La graduatoria sembra confermare la validità del titolo scelto per questo articolo. Nanotechnologies, Bioinformatics, Theoretical Biology, e DNA computing sono vocaboli che, combinando termini biologici e informatici, intendono descrivere nuove discipline che intrecciano le competenze di entrambe le scienze e indicano la direzione della ricerca a mio parere più promettente.

---

(1) www.lswn.it/comunicati/stampa/2008/alzheimer_la_cura_puo_arrivare_dall_ngf_di_rita_levi_montalcini

Parte seconda

Tra il dire e il fare… Gli elementi stabili e la riscrittura Alla ricerca dei combinatori stabili Osservazione Il λ-calcolo come macchina astratta Rappresentazione grafica dei λ-termini chiusi stabili: esempi Sosta riflessiva sui risultati raggiunti e quelli da raggiungere: il funzionale Prima riflessione Seconda riflessione Equazioni, dati e risultati: uno, due, molti Uno alla volta … per carità da: Il barbiere di Siviglia di G. Rossini Conclusione

E' arrivato il momento di mostrare al lettore, per esempio, come ci si possa difendere contemporaneamente dai virus telematici e biologici.

Durante il secolo scorso alcune persone (di professioni diverse fra loro) si sono accorte che, sia per la descrizione dei processi biologici, sia di quelli informatici, ci si avvale essenzialmente delle stesse tre componenti: la comunicazione a distanza, l'aritmetica e la logica (queste due ultime in modo assai elementare). Quanto appena descritto corrisponde esattamente alla descrizione d'un algoritmo, mette in atto cioè la proposta "Unifichiamo il linguaggio umano con quello degli algoritmi" che era una delle conclusioni cui si era giunti nel paragrafo precedente.

Nella prima parte di questo articolo si è cercato di mettere in evidenza:

a) Un tema informatico attuale mirante a un'evoluzione del software un po' meno caotica, dirigendola verso un importante scopo scientifico:

Chiarire e difendere l'evoluzione dell'umanità e del pianeta che ci ospita.

b) Descrivere l'essenza dei computer (basata ancor oggi sui trasferimenti del contenuto da una cella di memoria a un'altra e sulle operazioni aritmetiche necessarie per l'esecuzione dei suoi microprogrammi), come lo studio di metodi di risoluzione di opportune equazioni aventi un particolare carattere algebrico.

c) Le realizzazioni di Kurt Gödel che, malgrado le raccomandazioni che i logici matematici hanno pensato bene di diffondere, non ha posto dei limiti alla risolubilità dei problemi che coinvolgono la logica matematica e l'aritmetica: al contrario, ha garantito la loro risolubilità in un tempo illimitato,

grazie alla libera creatività concessa ai ricercatori.

Non ho perciò nessun dubbio nell'affermare che Kurt Gödel fu il più importante informatico teorico del 20°secolo.

Gli elementi stabili e la riscrittura

Per poter realizzare pienamente la finalità espressa nella risposta alla domanda 2 del paragrafo introduttivo (appunto quella di sfruttare la possibilità aperta da Gödel accennata nelle righe precedenti) dobbiamo superare un ostacolo che si è concretamente manifestato all'inizio del XXI secolo.

Le persone che si dedicano alla produzione di software non hanno un linguaggio sufficientemente aggiornato. Evidentemente manca una distinzione

fondamentale fra i concetti relativi agli *strumenti* usati da queste persone: linguaggi di partenza e di arrivo, traduttori, sistemi operativi, natura e forma dei dati e dei risultati. Ognuno di essi diventa invece una sorgente di complicazioni.

La distinzione che qui si propone è quella fra *caducità* e *perennità*. Gli oggetti relativi a tale distinzione possiamo chiamarli: **trasformabili** e **stabili**. Elementi **stabili** sono: *dati, risultati, o indirizzi di entrambi (tutti rappresentati da numeri interi), o (nomi di) programmi, (di) operatori, (di) operandi.* L'idea principale che vogliamo usare (si tratta pur sempre del significato che vogliamo attribuire a dei frammenti informativi, cioè a un certo numero di bit) è che *tale significato dipende esclusivamente dalla loro posizione relativa.* Siamo arrivati dunque a descrivere ogni processo di calcolo mediante un numero finito di regole di trasformazione del tipo:

... → ...

dove a sinistra della freccia troverà posto soltanto un oggetto *trasformabile* mentre alla sua destra vi dovrà essere un oggetto o *trasformabile* o *stabile*.

Nel 1909 il matematico Axel Thue inventò un sistema di *riscrittura* con la precedente simbologia dove a sinistra e destra della freccia stavano delle *stringhe* alfabetiche (costituenti un linguaggio la cui sintassi ne decideva l'appartenenza) come strumento per decidere problemi sulla risolubilità di semigruppi associati a tali stringhe, chiamato poi, in suo onore: "semi-Thue system". Si trattava di un numero finito di regole di riscrittura *siffatte*:

... → ...

Le strutture delle stringhe a sinistra della freccia, appartenenti a regole diverse, erano diverse fra loro, in modo tale che ogni stringa del linguaggio poteva soddisfare al più un solo primo membro di tali regole. Se ciò succedeva la stringa veniva trasformata in quella del secondo membro di tale regola. Altrimenti (cioè se la stringa non soddisfaceva alcun membro sinistro di tali regole) tale stringa costituiva un *elemento stabile.*

*Alla ricerca dei **combinatori** stabili*

Una conferma di quanto appena riferito ebbe luogo in una conferenza che il logico Moses Schönfinkel fece nel 1920, seguita nel 1924 da un articolo intitolato "Sui mattoni della logica" in cui fu definita la *logica combinatoria*. Da non dimenticare è il contributo importante che il logico Haskell B. Curry diede a tale disciplina nel periodo 1928-82 (anno della sua scomparsa). Noi vorremmo ora illustrare i combinatori di Schönfinkel, insieme al relativo sistema di riscrittura, e caratterizzarne gli elementi stabili. Se vogliamo però usare le nozioni introdotte nel primo capitolo dovremmo tener conto delle nuove visioni sull'informatica, che ci permettono di semplificare la cosiddetta *sintassi* guadagnando in *semantica*. Nessuno ci obbliga a usare solo *una* dimensione cioè le stringhe che, per rendere sufficientemente espressive le formule, abbisognano di un ausilio abbastanza *noioso*: le *parentesi*! Invece guadagniamo nel rappresentare visivamente un'operazione: la cosiddetta *applicazione* (nel senso di applicare una funzione, o operatore, a uno o più

argomenti) usando 2 dimensioni! Adoperiamo qui un nome più umano, più personale e più limitato: *doppia delega*. Cioè deleghiamo un oggetto stabile, per esempio di nome "a" a essere un *operatore*, **oppure** a essere un *operando* a seconda della posizione indicata

```
 Λ        oppure      Λ
a                    a
```

(ovviamente qui sopra non sono illustrate deleghe, ma solo semi-deleghe)

La logica combinatoria può essere generata dai due soli combinatori **S** e **K**, *oggetti stabili*, ripetutamente usati come operandi e operatori. Prima di continuare vogliamo però illustrare come disponendo soltanto di due tipi di deleghe possiamo perfino ottenere che un operatore possa avere anche più di un operando, ma addirittura un numero finito qualsiasi di essi! Intanto notiamo che se a e b sono due nomi di combinatori la notazione:

```
 Λ
a b
```

può star solo a significare la simultanea delega di a come operatore e di b come operando. Quindi, la scrittura *bidimensionale*

```
   Λ
 Λ   c
a   b
```

significherà che l'operatore a è possiede, ordinatamente, i due operandi b e c.

Per descrivere la logica combinatoria bastano due regole di riscrittura, descriventi rispettivamente **S** e **K** come operatori usando, come preannunciato, alberi *binari* al posto di stringhe. Tutti i nodi saranno etichettati da **S** o da **K** quali unici generatori della logica combinatoria. Le radici saranno rappresentate dall'operatore *delega* Λ talvolta con @ sovrastante, ma solo come etichetta (superflua) mentre tutti gli altri vertici sono etichettati da **S**, **K** o, se si tratta da oggetti incogniti, da lettere per es. a,b,c,.. che rappresentano combinazioni stabili indeterminate, i cui nodi apparteranno pur sempre all'insieme {**S**,**K**}.

```
      Λ
(S)  Λ  c                Λ                    (K)          Λ
    Λ  b      →         Λ   Λ                              Λ  b  →  a
.  S a                  a   c  b  c                        K a
```

Ora trasformeremo questa riscrittura di alberi binari in quella su stringhe. Attenzione, però! A differenza delle abitudini (cominciare a leggere un albero dalla radice, in alto, verso le foglie in basso), qui si comincia a leggere dalla foglia, la più in basso a sinistra. Per esempio, se post-scriviamo @ al posto dell'operazione "delega", le *riscritture* delle regole (**S**) e (**K**) (da alberiformi a stringhe) diventeranno:

S a @ b @ c @ → **a** c @ b c @ @ **K** a @ b @ → a

Osservazione
Il combinatore **K** è molto particolare. Si può dire che opera su a e b, ma il risultato finale è a. Esso appartiene dunque alla categoria dei *selettori*.

Nota per gli esperti in informatica (cui anche questo articolo è rivolto).

Le stringhe ottenute dalla lettura degli alberi, altro non sono che delle espressioni algebriche scritte in notazione polacca inversa, che, abolendo le parentesi, si prestano ottimamente a essere eseguite da un elaboratore dotato di stack (pila). Si noti che questo risultato è stato raggiunto privilegiando la semantica rispetto alla sintassi.

Segue ora la determinazione di tutte le forme stabili derivanti dalle regole definitrici (**S**) e (**K**) nelle versioni *alberi binari* in cui, come al solito, a,b e c siano elementi dell'insieme {**S,K**}. Esaminiamo perciò le varie ipotesi:

1) a = **S**. Dopo aver usato una volta la regola definitoria di **S** consegue che per poterla applicare una seconda volta 'a' deve essere delegato almeno 3 volte mentre a disposizione ve ne sono soltanto 2, quindi il risultato è stabile. Usando a = **S** nella regola definitoria di **K** consegue che qualunque sia b il risultato è **S** quindi il risultato è stabile.

2) a = **K** Dopo aver usato una volta la regola definitoria di **S** consegue da quanto detto prima cioè che si può applicare una sola volta la regola definitrici di **K** ottenendo l'elemento stabile c.

Il λ-calcolo come macchina astratta

I combinatori che ci hanno interessato finora sono delle entità stabili e, sebbene non dotati di spazio interno disponibile, sono però collocabili su un piano e passibili di diversi comportamenti secondo il ruolo (attivo o passivo) cui sono stati delegati. Se un combinatore ha un ruolo passivo può essere: soppresso, spostato o raddoppiato dal combinatore investito del ruolo attivo. Se invece il ruolo del combinatore, appena nominato, è attivo, esso si comporta come un programma fisso e inalterabile.

I λ-termini stabili (cioè, tradizionalmente, chiusi e in forma normale) che stiamo per introdurre sono delle entità dotate di spazio interno sufficientemente ampliabile che, se riempito, ne può alterare le finalità, rendendo i loro comportamenti più simili a quelli dei sistemi operativi, comprendendo i loro microprogrammi, assumendo perfino un comportamento robotico. Infatti, mentre i combinatori (ce ne sono molti altri oltre ad **S** e **K**) hanno un nome costituito da una lettera maiuscola più o meno segnata, p.es. **I**,**B**, **B**', **W**, **W***, **C**, **C*** ecc., i λ-termini non hanno bisogno di nomi, tuttavia, come vedremo fra breve, la β-regola di riscrittura ne definisce il comportamento. Siamo quindi in grado di confutare la leggenda metropolitana: "Il lambda-calcolo è troppo astratto, troppo lontano dalla realtà e dalle esigenze pragmatiche dei computer e dell'informatica applicata odierna". Basta rammentare l'articolo di Jacopini (1965 Calcolo, Giuseppe Jacopini, Macchina di von Neumann con una sola istruzione) ovvero: Il trasferimento del contenuto da una cella ad un'altra ->LA COMUNICAZIONE! (le operazioni aritmetiche stavano nella gestione del contatore delle istruzioni) in cui si dimostra che gli attuali computer hanno una sola istruzione: un trasferimento d'informazione. Ebbene

anche il lambda-calcolo ha una sola operazione: quella che noi abbiamo chiamato ***delega***. Come avviene il trasferimento dell'informazione? Semplicemente precisando:

Cosa deve essere trasferito e ***dove***. Il nome di questo operatore è ***sostituzione.***

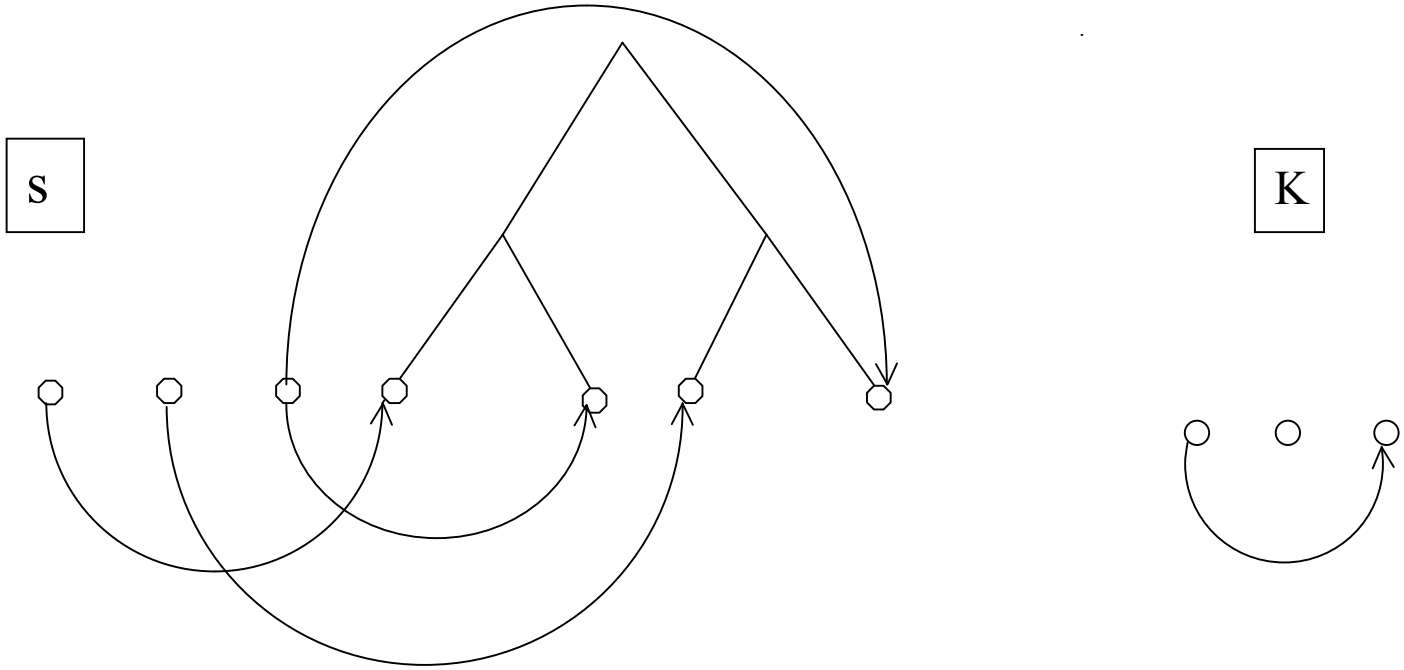
Finita la fase di benevola polemica tendente ad annullare i pregiudizi sul λ-calcolo, si rendono ora necessari degli esempi che confermino le giustificazioni addotte. Ecco i classici λ-termini che definiscono i combinatori **S** e **K** e le loro successive semplificazioni (storiche o nostre, o a grafo) :

Storica	**S** = λ x y z. x z(y z)	**K** = λ x y.x
De Bruijn	**S** = λ 1 2 3.1 3(2 3)	**K** = λ 1 2.1
nostra	**S** = λ 1 2 3.1 3(2 3)	**K** = λ 1 2.1

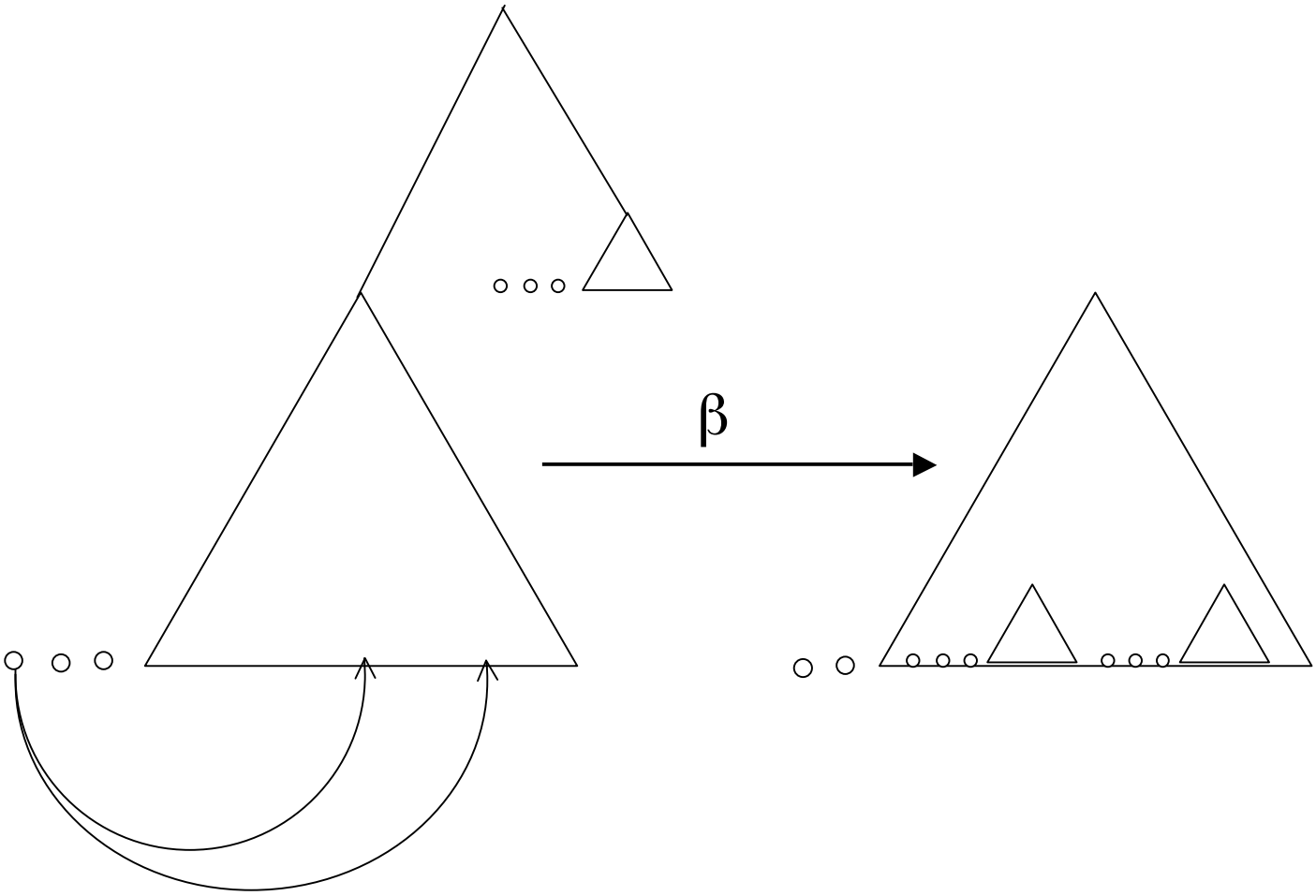
Nota. Per rispetto alla tradizione, in questo articolo, rinunciamo all'abolizione della lettera λ. Riserviamo invece per la prossima sezione la rappresentazione a grafo di **S** e **K** come λ -termini.

Rappresentazione grafica dei λ-termini chiusi stabili: esempi

Si noti che l'albero di **S** come visto in (**S**) si ritrova nella definizione di **S**, più le frecce, che indicano i posti dove deve avvenire la sostituzione. Lo stesso fenomeno succede, anche se in modo molto più semplice per **K** *e così anche per ogni λ-termine stabile.*



L'ultima osservazione *è importante.* Infatti dà la chiave per individuare l'unica regola di riscrittura, per noi, del λ-calcolo, chiamata β-regola, qui chiamata (doppia) delega.
In effetti, nella versione classica del λ-calcolo, esiste un'altra regola di riscrittura, chiamata α-regola, che consiste nel ridenominare tutte le variabili legate. Nel nostro caso i nomi delle variabili sono spariti poiché i luoghi di sostituzione sono imposti dalle frecce del grafo. Quindi a noi basta descrivere la seguente regola (unica) di riscrittura cui assegneremo il nome $\_\beta\_\_>$.



Si noti che:
1) le frecce arrivano alla base del termine a sinistra della freccia β perché tale termine, delegato come operatore, è scritto su una riga.
2) il termine delegato come operando è troppo piccolo per vederne le frecce.

Sosta riflessiva sui risultati raggiunti e quelli da raggiungere: il funzionale
Nella prima parte di questo articolo si è discusso sui vari atteggiamenti che si possono (o meglio si dovrebbero) assumere nei confronti dell'attuale informatica, cercando di capire i suoi vari intendimenti. In questa seconda parte si è cercato di descrivere, nel modo più uniforme possibile, i mezzi tecnici d'approccio alla soluzione di tutti gli interrogativi posti nella prima parte.

Sembra giusto chiamare tale approccio *Programmazione funzionale.*
Attenzione però! Se vogliamo unificare il nostro approccio (e in realtà a questo punto lo dobbiamo) il vocabolo *funzionale* deve poter acquistare nuovi significati e incarichi.
1° Quindi l'aggettivo funzionale può essere anche un nome.
2° Come tale può significare: numero intero, combinatore, e (con tutti i suoi aggettivi) funzione (con argomenti e/o valori appartenenti a ciascuna delle quattro categorie appena nominate).
3° Questa poli-semantica acquisita da uno stesso vocabolo semplifica ed aumenta grandemente il potere risolutivo degli oggetti finora introdotti rispetto ai problemi che ci siamo posti all'inizio di questo articolo.
Quanto appena detto si applica soprattutto a compiti accennati nella prima parte di questo articolo: il lettore si ricorda certo di **K** considerato come killer e di problemi che descrivono fenomeni spiacevoli, quali "il fuoco amico" ed altri pericoli da evitare, derivanti dal fatto che un funzionale si trova nella condizione di dovere operare su sé stesso.

Prima riflessione E' sempre così pericoloso che un funzionale operi su sé stesso? Una risposta viene dall'illustre creatore del lambda-calcolo Alonzo Church intorno alla fine degli anni '30 con i cosiddetti "numerali di Church", che rappresentano i numeri naturali con lambda termini chiusi in forma normale. Se *n* rappresenta il numero naturale n allora *n^n* rappresenterà la potenza n-esima di n (più precisamente *n^m* rappresenterà m^n). Quindi la risposta alla domanda è no. Data la sua importanza riprenderemo fra poco il problema dell'operare su sé stessi, in quanto suggeritore di ulteriori verità nel campo scientifico proposto nel presente articolo.

Seconda riflessione Il fatto d'aver dato il nome di funzionale ad ogni elemento stabile della logica combinatoria (generata da **S** e **K**) può ovviamente essere esteso agli elementi stabili (le forme normali chiuse) del lambda-calcolo e tutto ciò faciliterà il trattamento equazionale dei problemi annunciati nella prima parte di questo articolo.

Equazioni, dati e risultati: uno, due, molti
E' arrivato il momento di avvalersi della seconda riflessione, appena espressa.
Dunque le equazioni verteranno su funzionali, alcuni noti, altri incogniti, comunque stabili. Il funzionale fondamentale è quello denominato "^" operante su 2 argomenti che abbiamo incontrato, prima fra i combinatori, poi fra i lambda-termini. Ciò che ora ci deve interessare (vedi il sottotitolo) è la molteplicità degli oggetti stabili che prenderemo in considerazione; ve ne sono due specie, ambedue ordinate linearmente: una, in cui la molteplicità è nota, è chiamata *n-pla*; l'altra, invece, in cui la molteplicità può essere qualsiasi, purché finita, è chiamata *lista*.
Diamo la preferenza al funzionale n-pla, ed in particolare a quello inventato da Church detto appunto **n-pla di Church** (nCh), in quanto denso di significati nei due possibili ruoli cui può venire delegato: quello di operatore e quello di operando. Usando la notazione di De Bruijn l'n-pla "nCh" dei funzionali $f_1,\ldots,f_n$ sarà descritta da

$$nCh = \lambda\ 1.1\ f_1 \ldots f_n\ .$$

Definiamo ora un funzionale U^n_i (selettore della i-esima componente di una nCh)

$$U^n_i = \lambda\ 1\ 2 \ldots n.i$$

(si tratta d'un operatore che applicato a n funzionali $f_1 \ldots f_n$ li cancella tutti tranne l'i-esimo f_i).

Esempio: Usando le definizioni e gli schemi precedenti vale la relazione:

$$nCh \quad U^n_i \quad \overset{\Lambda\ \beta}{\rightarrow} \quad f_i$$

ovvero, scrivendo per esteso i λ– termini,

$$\lambda\ 1.1\ f_1 \ldots f_n \quad \lambda\ 1\ 2 \ldots n.i \quad \overset{\Lambda\ \beta}{\rightarrow} \quad f_i\ .$$

In un teorema del 1974 (Böhm e Dezani) è stato dimostrato che ogni funzionale con una sola astrazione esterna D , chiamato *deed* [R. Statman(1979?), se applicato ad un opportuno funzionale, può ottenere come risultato un funzionale arbitrario.

Nota Il più semplice deed è ovviamente $\lambda 1.1$ cioè l'operatore che restituisce inalterato il suo operando, ma lo è anche un'n-pla di Church!.

Da un successivo articolo Böhm e Intrigila (19??) risulta che ogni deed D possiede almeno un punto fisso F che a sua volta è un funzionale:

$$D\hat{}F = F$$

Successivamente l'autore di questo articolo (Böhm, 200?) usando quest'ultimo risultato, ha dimostrato che per ogni coppia di deed D_1, D_2 l'equazione

$$D_1x = D_2x$$

ha sempre almeno un funzionale come soluzione.

D'altra parte,usando le n-ple di Church e i suoi proiettori, è sempre possibile:

a) trasformare ogni sistema di h >1 equazioni coinvolgenti n funzionali dati $M_1,\ldots, M_n$ in una sola equazione con due funzionali dati M'_1 ,M'_2

$$M'_1x=M'_2x$$

[E.Engeler, The combinatory programme, Birchhause 1995, p.7]

b) trasformare ogni funzionale M in un deed D

c) trasformare quindi la precedente equazione in quest'ultima

$$D'_1x=D'_2x\ .$$

Vale allora la conclusione :

Si può risolvere ogni sistema di equazioni funzionali mediante i deed.

Vorremmo far notare però che mentre la trasformazione di un sistema di equazioni in una sola, col metodo delle n-ple di Church ed i proiettori è rigorosamente reversibile, la trasformazione dei due M in due D non conserva (ma in senso ottimistico) la risolubilità delle rispettive equazioni. Se la prima delle due equazioni non è risolvibile la seconda lo è

sempre! (vedi terz'ultima equazione). E' opportuno, a questo punto un esempio:

K x = **O** x ovvero (con i λ-termini) (λ12.1)x = (λ12.2)x

Omettiamo a questo punto una discussione filosofica che si connette inevitabilmente con il contributo di Gödel (vedi prima parte di questo articolo) od anche ai risultati dei fisici teorici che, per risolvere i loro problemi, sono ricorsi a più dimensioni, ben oltre alla quarta (spazio-tempo) raggiungendo talvolta l'undicesima od anche infinite dimensioni (teoria dei quanta).

Uno alla volta … per carità da: Il barbiere di Siviglia di G. Rossini

Questa penultima sezione tenta di rispondere nel modo più desiderabile alle domande poste nella prima parte del presente articolo, fornendo delle soluzioni ai problemi più critici colà posti e cioè:

1) sfruttamento dell'apoptosi (eliminazione delle cellule malate con strumenti informatici).

2) sfruttamento del fattore NGF per bloccare lo sviluppo della malattia Alzheimer, etc.

3) eliminazione dei virus informatici e biologici sopprimendo le azioni dirette di agenti su loro stessi (tra cui la riproduzione).

Per risolvere i problemi presenteremo un funzionale M, il più semplice possibile, che contiene le minacce citate e che spariranno trasformandolo in un particolare deed (deed ereditariamente finito, in breve *def*) con un algoritmo illustrato proprio in questa sezione.

Riassunto della nomenclatura e risultati principali. Funzionale è un λ–termine chiuso in forma normale. Nelle equazioni dati e incognite sono funzionali. Ogni sistema di n equazioni è riducibile (tramite l'uso di n-ple di Church e di proiettori, come visto nel paragrafo precedente) a una sola equazione del tipo

$$M_1\ x = M_2\ x$$

dove M_1 e M_2 sono funzionali noti mentre x è quello incognito. L'equazione potrebbe non avere soluzione esattamente quando M_2 è (essendone stato scelto, per convenzione iniziale, il numero di astrazioni iniziali minore o uguale a quello di M_1) un sottotermine di M_1.

Per aumentare l'efficienza dell'algoritmo di risoluzione è conveniente riuscire a trasformare M_1 e M_2 in deed ereditariamente finiti *def* M_1 e *def* M_2 .Un deed ereditariamente finito è un funzionale che possiede una sola astrazione esternamente mentre le astrazioni interne possono bensì essere presenti (altrimenti tale funzionale sarebbe semplicemente un' n-pla di Church!) ma apparire soltanto una alla volta (ecco la ragione del sottotitolo rossiniano!).

Vogliamo ora costruire $\mathcal{D}$M, l'albero di Dezani (1974, già citato) partendo dal funzionale M descritto come stringa del λ-calcolo.

Tale stringa in generale può contenere parentesi aperte e chiuse che appunto verranno eliminate in $\mathcal{D}$M; però rappresentando M un funzionale stabile a nessuna sottostringa di esso si dovrà potere applicare la β-regola. Per non annoiare il lettore enuncerò soltanto alcune di queste proibizioni.

1) Nella stringa che rappresenta M non si dovranno mai presentare le seguenti sottostringhe: ((, (un solo intero positivo) .
Preciso maggiormente il perché della prima proibizione.
Una stringa rappresentante un funzionale non può iniziare con una parentesi aperta o perché la corrispondente chiusa dovrebbe terminare il funzionale (e allora sarebbe inutile) oppure perché dovrebbe operare su ciò che segue perdendo la suo stabilità. Perciò ogni funzionale inizia con una stringa λ1 2…n.i (che, se finisse lì, rappresenterebbe il selettore U^n_i).
In tal caso $\mathcal{D}\ U^n_i = <n,i>$ e l'albero è una radice-foglia che coincide con la sua etichetta.
Ora siamo in grado di descrivere come è strutturato l'albero $\mathcal{D}$M per ogni funzionale M descritto come stringa. Abbiamo visto che la sua radice è etichettata dalla coppia < n,i > che corrisponde al selettore U^n_i .
Il bello è che <u>ogni nodo dell'albero $\mathcal{D}$M è etichettato da un selettore</u>.
Scegliamo come esempio un M di cui abbiamo anticipato i pregi nelle prime righe della presente sezione

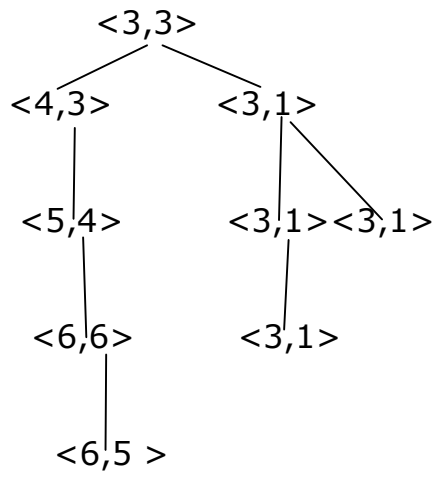
(Es. M) λ123 . 3 (λ4 . 3 (λ5 . 4 (λ6 . 6 5))) (1 1 1)

Cominciamo con delle osservazioni immediate e intuitive come a conferma della frase sottolineata. Scegliamo, cominciando da sinistra, coppie di numeri separati da un punto: per prima otteniamo o <3,3> (l'etichetta della radice) Le due coppie di parentesi(aperta-chiusa) che seguono 3 in M ci fanno capire che il nodo della radice ha due figli:il primo con etichetta <4,3> ed il secondo con etichetta non ancora evidente, mentre è indiscutibile che l'etichetta <4,3> ha il figlio <5,4> e un nipote (figlio del figlio) <6,6>.
Continuiamo la ricerca di ciò che nella stringa può seguire immediatamente a destra dell'etichetta della radice:
a) un numero non superiore ad n che diventi in $\mathcal{D}$ M il primogenito della radice <n,i>
b) Una parentesi aperta che segnali nell'albero $\mathcal{D}$ M la presenza di un nuovo figlio o di un nuovo fratello.
c) Confermiamo che in tale stringa non si presentano due parentesi aperte consecutive perché ciò preannuncerebbe che N non è stabile e quindi non può essere un funzionale.
Ogni funzionale comincia col selettore λ12..n.i cui seguono numeri naturali, parentesi aperte,ecc. Cerchiamo di completare la conferma della frase sottollneata cioè:
1)La trasformazione della stringa rappresentante un funzionale M nell'albero, non necessariamente binario, $\mathcal{D}$M i cui nodi siano etichettati da coppie di naturali positivi fra loro in ordine non crescente.
Ci rimane dunque solo di trovare le coppie in (111)!
Osserviamo che (1 1 1) risulta essere il terzo figlio della radice <3,3>.
(Attenzione! Il discorso deve essere generale): poiché, risalendo verso la radice dell'albero che stiamo costruendo, la prima astrazione che si incontra è 3 (non badiamo che in questo caso è anche l'unica!) possiamo dunque al posto scrivere, al posto di (1 1 1): (< 3,1> 1 1).
Lo stesso discorso si può ripetere per gli ultimi(potenziali) figli di <3,1>.

Riassumendo:

M λ123 . 3 (λ4 . 3 (λ5 . 4 (λ6 . 6 5))) (1 1 1)

$\mathcal{D}$M



A questo punto descriviamo (ciò che più ci interessa) la trasformazione da $\mathcal{D}$M in $\mathcal{D}$ (*def* M).

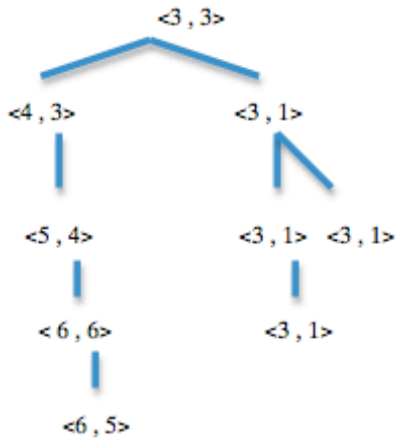
$\mathcal{D}$ (*def* M) avrà il numero doppio dei nodi di $\mathcal{D}$M poiché, per generarlo, dovremo partire dall'albero $\mathcal{D}$M, *e ad ogni nodo*, come prima operazione, aggiungergli un figlio (in posizione primogenita, con etichetta uguale a quella del padre) che diventa perciò una *foglia* di $\mathcal{D}$ (*def* M). Dopo il raddoppiamento dei nodi con le rispettive etichette stabiliremo l'ordine di visita dei nodi di $\mathcal{D}$M che devono subire ovviamente un cambio di etichetta per diventare nodi di $\mathcal{D}$(*def* M). Cominciamo dalla radice di $\mathcal{D}$M. Poiché *def* M è (per definizione) un deed, <1,1> deve essere l'etichetta della radice di $\mathcal{D}$(*def* M). Decidiamo ora l'ordine in cui dobbiamo visitare i nodi di $\mathcal{D}$M, dopo la radice, per trasformarli in nodi di $\mathcal{D}$(*def* M).
Pensiamo a $\mathcal{D}$M come se il suo albero dovesse costituire la parte solida del delta di un fiume allo sbocco nel mare. Si tratta di visitare uno dopo l'altro i "porti" del delta,cioè i nodi dell'albero, e trasformarne l'etichetta, circumnavigando il delta in senso antiorario, cioè partendo dalla "("e arrivando alla")" sua radice etichettata <1,1>.
Avendo preannunciato l'ordine di visita dei vari nodi di $\mathcal{D}$ M esaminiamone ora la trasformazione delle etichette (eccetto la radice ed i nuovi primogeniti, già sistemati) per ottenere quelle di $\mathcal{D}$ (*def* M).
Cominciamo dunque col calcolare la nuova prima componente dell'etichetta del nodo in esame. Confrontiamone la corrispondente vecchia (ora già nella foglia del nuovo primogenito di suo padre) con l'attuale prima componente del padre. Soltanto se il primo dei due numeri supera il secondo allora la nuova prima componente è quella del nodo padre incrementata di 1, altrimenti essa coincide con quella del padre. Confrontiamo ora la seconda (vecchia) componente (sempre nella foglia del nuovo primogenito di suo padre) con l'attuale seconda componente del padre. La nuova seconda componente viene ora calcolata in modo

simmetrico alla prima. Tutto ciò viene ripetuto fino alla fine della circumnavigazione.

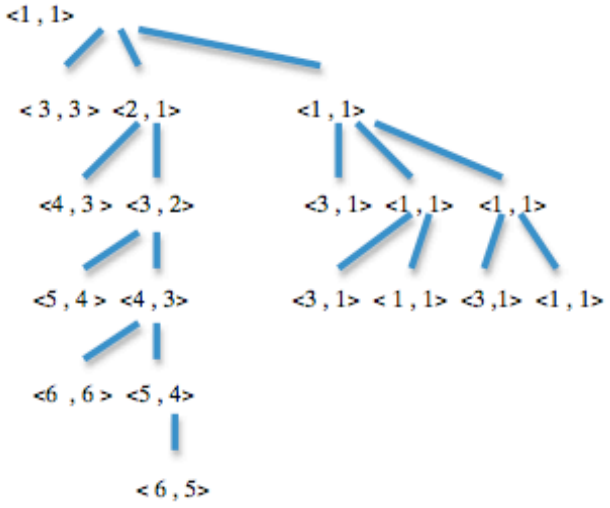
.Attenzione!
I GRAFI FINALI SONO nel documento GrafoCorrado3.pdf
M i titoli a sinistra vanno corretti (qui sotto impiccoliti) D-> $\mathcal{D}$

M λ123 . 3 (λ4 . 3 (λ5 . 4 (λ6 . 6 5))) (1 1 1)

DM



D(*def* **M**)



def **M**

λ1 . 1(λ3 . 3) (λ2 . 1(λ4 . 3) (λ3 . 2(λ5 . 4) (λ4. 3(λ6 . 6) (λ5. 4(λ6 . 5))))(λ1 . 1(λ3 . 1)

(λ1 . 1(λ3 . 1) (λ1 . 1))(λ1 . 1(λ3 . 1)(λ1 . 1)))

Nota per gli specialisti Vantaggi del metodo scelto:Quando le equazioni non hanno soluzione non è più necessario η- espandere di una variabile legata (posta all'ultimo posto delle astrazioni di entrambi M_1 e M_2)

Conclusione

L'uso dei deed ereditariamente finiti (*def* M) dove M è un funzionale, p.es. la soluzione d'una equazione funzionale, garantisce alcuni ulteriori importanti risultati:

1) Le auto-applicazioni che si possono trovare in M spariscono in *def* M. Ciò elimina le azioni distruttive che si possono presentare sia in un virus informatico oppure in un funzionale che descrive il funzionamento d'un virus biologico, permettendo inoltre di impiegare le tecniche, usate come difesa dai cultori della teoria dei tipi elementari.

2) Un *def* M grazie all'intervento dei funzionali che fanno delle sostituzioni una per volta, può far risparmiare spazio e tempo nell'esecuzione di programmi, la cui complessità ha carattere esponenziale, rendendola più simile a quelli la cui complessità ha carattere polinomiale.

3) Infine il software che usa i *def* M anziché gli M incoraggia la produzione di robot nanotecnologici che potranno entrare più facilmente nell'organismo umano e liberarci dalle malattie operando direttamente sulle cellule malate.

Per terminare, non oso pensare a quello che potrà succedere ai cellulari che potranno abitare nel nostro corpo e trasmetterci p. es. film a carattere olografico, quindi a tre dimensioni, senza pellicola e senza schermo, obbedendo a istruzioni non trasmesse dalle nostre dita o dalla nostra voce, ma generate dalla nostra mente, dai nostri pensieri!