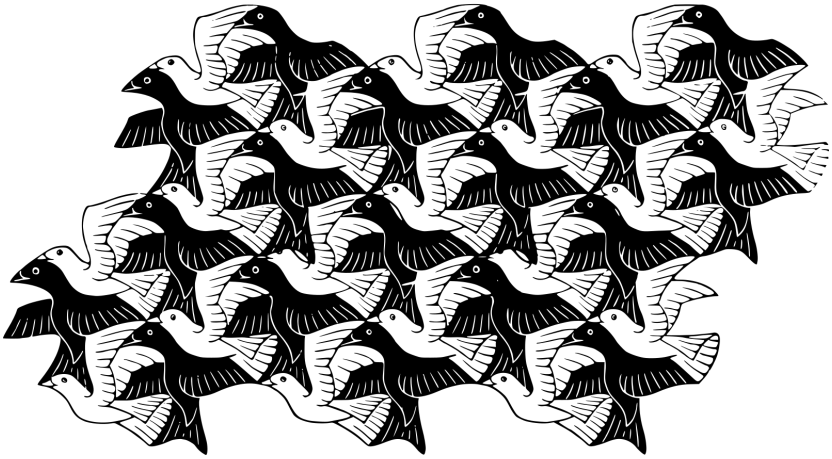
I SISTEMI FORMALI

TESI DI FINE ANNO DI

MICHELE GUERINI ROCCO



ANNO 2014/2015

LICEO SCIENTIFICO GALILEO GALILEI, CREMA

In copertina: Regular Division of the Plane I di M.C. Escher (1936)

Dedicata a Emil Post (1897-1954) che ha di fatto creato la teoria della computazione e reso possibile tutto questo ricevendo poco o nulla come riconoscimento.

I sistemi formali

Michele Guerini Rocco

25 Giugno 2015

Sommario

Un guida attraverso il percorso della ricerca dei fondamenti della matematica, la nascita del formalismo e la dimostrazione del teorema di incompletezza di Gödel. Vedremo poi come i sistemi formali sono utilizzati nell'informatica e sono ancora oggi importanti. In conclusione un'interessante analogia tra il lavoro di Gödel e le opere del metateatro di Pirandello.

1 I sistemi Formali

1.1 Introduzione: l'indagine sui fondamenti

Nei cinquant'anni a cavallo tra XIX e XX secolo si è sviluppata un'importante **riflessione** e ricerca sui **fondamenti della matematica**. Partì dalla necessità di definire in modo rigoroso il calcolo infinitesimale nel quale rimanevano ancora poco chiari concetti come quello di *infinitesimo*, *limite* e *integrale*. Le spiegazioni intuitive fornite dai tempi di Leibniz che sfruttavano la geometria non potevano essere infatti le vere basi epistemologiche del calcolo ma queste andavano ricercate altrove.

1.2 Aritmetizzazione

In una prima fase l'obbiettivo fu quello di **ridurre** tutte le nozioni della matematica ai concetti più semplici possibili dell'**aritmetica** a cui contribuirono matematici come Dedekind, Weierstrass e Cantor.

I **numeri** interi, razionali, reali e complessi furono ricondotti a numeri naturali. I complessi come coppia di numeri reali; i reali per esempio come *sezioni* nell'insieme dei razionali; i razionali come coppie di naturali.

Figura 1: Georg Cantor

1.2.0.1 Teoria degli insiemi

La teoria degli **insiemi** che era stata fino a quel momento molto generica e intuitiva venne studiata a fondo. Georg Cantor negli anni '80 del XIX secolo fece chiarezza sugli insiemi di **infiniti elementi** (come i numeri naturali) dove la concezione intuitiva di insieme inizia ad avere problemi. Definì i concetti di cardinalità e numeri transfiniti arrivando alla rivoluzionaria scoperta dell'esistenza di **infiniti di diverso ordine**. I matematici Zermelo e Frankel a partire dal 1908 produssero la prima teoria *assiomatica* degli insiemi su cui si basa ancora oggi tutta la matematica ordinaria.

1.2.1 Logicizzazione

In questi stessi anni c'è chi decise di andar oltre l'aritmetica nel processo di riduzione della matematica e cercò quindi di ricondurre tutto ai principi della logica, ancora più originari e indiscutibili.

1.2.1.1 Aritmetica di Peano

Il matematico italiano Giuseppe Peano nel 1889, seguendo l'esempio della geometria di Euclide, individuò le caratteristiche fondamentali dei numeri naturali e le espresse in 5 proposizioni. Senza creare un complesso sistema formale si limitò a fornire le proprietà dei numeri dalle quali poi è possibile produrre tramite il ragionamento l'intera aritmetica. I cinque postulati di Peano possono essere espressi nel seguente modo:

1. Zero è un numero

2. Ogni numero ha un successore
3. Zero non è il successore di alcun numero
4. Numeri diversi hanno successori diversi
5. Se zero ha una proprietà X e ogni numero conferisce X al suo successore allora tutti i numeri hanno X

Vedremo in seguito che questi postulati, specialmente l'ultimo, avranno un ruolo fondamentale, oltre che nel definire i sistemi formali e le loro caratteristiche, nella dimostrazione del teorema di Gödel.

1.2.1.2 Il programma di Frege

Il logico matematico **Gottlob Frege** tentò di ricondurre i numeri alla logica tramite l'uso del concetto logico di **classe** (o insieme) ed **equinumerosità**. Due classi sono equinumerose quando è possibile creare una relazione biunivoca tra gli elementi di esse. Definì quindi i **numeri naturali** in termini di **classe di classi**:

il numero della classe C è la classe di tutte le classi equinumerose a C. Per esempio il numero 2 è la classe di tutte le classi che hanno 2 elementi.

1.2.2 La crisi

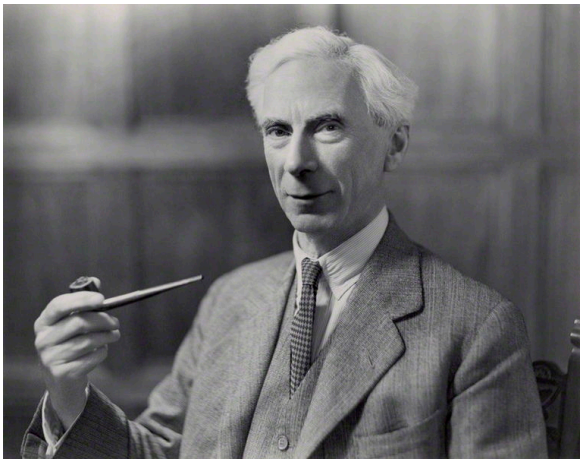


Figura 2: Bertrand Russell

Il sistema logicista entrò ben presto in crisi quando il logico inglese **Bertrand Russell** scoprì un'**antinomia** legata al concetto stesso di insieme che ne evidenziò la contraddittorietà.

È infatti possibile costruire un insieme che porta inevitabilmente ad una **contraddizione**. Alcuni insiemi abbiamo visto possono contenerne altri (come la

definizione di numero per esempio) e alcuni possono contenere sé stessi. Per esempio l'insieme di tutti gli insiemi contiene se stesso mentre l'insieme di tutte le persone non è una persona e quindi non contiene anche se stesso. Ora il **paradosso di Russell** dice questo: "L'insieme di tutti gli insiemi che non contengono se stessi contiene se stesso?"

Se non dovesse avere se stesso come elemento allora per la sua definizione dovrebbe contenere se stesso portando ad una contraddizione. Se invece contenesse se stesso significherebbe che per la sua definizione non dovrebbe contenere se stesso portando ugualmente ad una contraddizione.

Di fronte a questi risultati paradossali Frege stesso decise di abbandonare il programma logicista. Iniziò così la crisi dei fondamenti della matematica.

1.2.3 Principia Mathematica

Russell tuttavia non perse le speranze di dare un fondamento logico all'aritmetica e cercò di risolvere l'antinomia da lui stesso prodotta. Nell'immensa opera *Principia Mathematica* scritta da Russell in collaborazione con il matematico Alfred North Whitehead fu proposta la **teoria dei tipi** nel tentativo di eliminare l'autoreferenza delle classi.

Agli oggetti viene conferito un *tipo*: agli **individui** cioè numeri, oggetti fisici, elementi singoli viene dato il **tipo 0** mentre insiemi che contengono individui hanno **tipo 1**, insiemi di insiemi di tipo 1 hanno **tipo 2** e così via. Un insieme di tipo n può contenere solo oggetti di tipo minore di n.

Questa teoria non si limita agli insiemi: proibisce, per esempio, che io parli di questo documento in questo stesso documento come sto facendo ora: potrei parlarne solo in un metadocumento. Inoltre menzionare me stesso tramite il pronome "io" sarebbe del tutto impossibile. Persino la stessa discussione di questa teoria è una sua sfacciata violazione: questo fa capire, per quanto utile che sia, quanto la teoria dei tipi è limitante.

Con la teoria dei tipi viene eliminata la possibilità per un insieme di contenere se stesso e così anche il paradosso di Russell: tuttavia rimangono sempre possibili delle antinomie che sfruttano lo stesso principio ma legate al linguaggio come il **paradosso del mentitore** o il **paradosso di Grelling-Nelson**. Vedremo in seguito proprio queste saranno la causa del crollo dei *Principia* e in generale della ricerca di fondamenti formali della matematica.

1.2.4 Il programma di Hilbert

Se un concetto così semplice come quello di insieme aveva potuto creare dei problemi così grandi non è allora possibile che ci siano altre contraddizione interne alla matematica? La preoccupazione era che i paradossi della logica potessero essere presenti anche altrove. I matematici del tempo iniziarono a chiedersi se il

Figura 3: David Hilbert

ragionamento e la matematica siano separati dalla logica. Il grande matematico del '900 David Hilbert propose allora di dimostrare che il **sistema formale** dell'aritmetica fornito da Russell e Whitehead in Principia Mathematica fosse **coerente** (privo di contraddizioni) e **completo** (potesse produrre ogni possibile verità matematica) usando i metodi forniti dal sistema stesso. Quello che Hilbert sembra chiedere è che la matematica si tiri fuori da un pantano tirandosi per i capelli.

Figura 4: Il barone di Münchausen

In realtà Hilbert aveva previsto il possibile argomento circolare e aveva pensato perciò di aggirarlo dando una dimostrazione che usasse solo una parte ristretta dell'aritmetica, quella *finitista*: l'aritmetica elementare in cui solo ciò che è

possibile esprimere tramite un numero finito di passaggi è valido. Sarebbe stato allora possibile fare questo in modo assolutamente intuitivo e certo: senza possibili contraddizioni o paradossi.

1.2.5 I teoremi di incompletezza

Figura 5: Kurt Gödel a vent'anni

Nel 1931 tuttavia Kurt Gödel, divenuto in seguito uno dei più grandi logici di tutti i tempi, riuscì a dimostrare che un qualunque sistema formale che possa provare la coerenza di questa teoria è a sua volta potente come quest'ultima e quindi non è possibile aggirare l'argomento circolare rendendo vana ogni possibilità per una teoria di autogiustificarsi. Nonostante il risultato ottenuto da Gödel con i suoi due teoremi di incompletezza non vi è mai stata una fine dell'uso dei sistemi formali né tanto meno della ricerca di rigore che aveva animato all'inizio i matematici ma si è arrivati alla conclusione che **le pretese di giustificare la matematica non sono razionali**.

Vedremo quindi approfonditamente in cosa consistono i sistemi formali, come Gödel è arrivato a quanto abbiamo appena visto ed infine come e perché il formalismo è utilizzato ancora oggi.

1.3 I Sistemi formali

Esistono diverse definizioni di sistema formale: ne vedremo una in particolare ed utilizzeremo solamente quella da qui in avanti. Intuitivamente possiamo pensiamo ad un sistema formale in generale come ad un metodo rigoroso che permette di fornire una dimostrazioni ai teoremi.

1.3.1 Sistemi di produzione di Post

I sistemi di produzione di Post o *sistemi canonici* sono un tipo di sistema formale ideato dal logico americano Emil Post negli anni '20. Un sistema formale secondo questa definizione è costituito da un certo insieme di **simboli**, delle sequenze di simboli (**stringhe**) e delle **regole** per produrre nuove stringhe trasformando una stringa in un'altra tramite delle semplici manipolazioni di simboli, da questo il termine "di produzione".

1.3.2 MIU

Un esempio di semplice sistema formale di questo tipo è il sistema **MIU**, proposto da Hofstadter in "Gödel, Escher, Bach: An Ethernal Golden Braid".

Il sistema ci sarà utile più volte, data la sua semplicità, per spiegare le caratteristiche dei sistemi formali, il loro utilizzo e la dimostrazione stessa del primo teorema di incompletezza.

1.3.2.1 Simboli

I simboli a disposizione sono le tre lettere: M, I, U.

1.3.2.2 Stringhe

Sono stringhe del sistema MIU tutte le stringhe composte dei tre precedenti simboli. Esempi di stringhe MIU possono essere: MIU, MU, M, IMUUIUIUM, UMI, MIIIIII

1.3.2.3 Regole

Le regole che possiamo utilizzare sono le seguenti:

1. Ad una stringa che termina con il simbolo *I* si può posporre il simbolo *U*. Per esempio la stringa *MII* può essere trasformata in *MIIU*.
2. Una stringa del tipo *Mx*, dove x indica una qualunque sequenza, può essere trasformata in *Mxx*. Cioè è possibile raddoppiarla. Per esempio *MIIU* può diventare *MIIUIIU*.
3. La sequenza *III* in una stringa, in una posizione qualunque, può essere sostituita con *U*. Per esempio *MIUIIIUI* diventa *MIUUI*.
4. La sequenza *UU* in una posizione qualunque in una stringa può essere cancellata. Per esempio *MIUUIU* diventa *MIIU*.

Usando la notazione di Post possiamo enunciare così le regole:

1. $xI \rightarrow xIU$

2. $Mx \rightarrow Mxx$
3. $xIIIy \rightarrow xUy$
4. $xUUy \rightarrow xy$

Da notare che la freccia va in una direzione sola. Le stringhe **non** sono intercambiabili.

1.3.2.4 Derivazione

In questo sistema abbiamo a disposizione una sola stringa di partenza (**assioma**): *MI*. Applicando le regole di produzione a *MI* possiamo produrre tante nuove stringhe, infinite per la precisione.

Per esempio si può ottenere la stringa *MUIUI* in questo modo

stringa	regola applicata
1. MI	assioma
2. MII	regola 2 con x=I
3. MIIII	regola 2 con x=II
4. MUI	regola 3 con x=M e y=I
5. MUIUI	regola 2 con x=UI

La sequenza di passaggi compiuti per ottenere una stringa si dice derivazione o **dimostrazione** di tale stringa. Poiché abbiamo dimostrato che possibile produrre questa stringa possiamo dire che *MUIUI* è un **teorema** del sistema MIU (lo sono anche gli intermedi *MII*, *MUI*...).

1.3.3 Procedure di decisione

In un sistema formale è possibile fare una prima distinzione fra tutte le possibili stringhe: stringhe **ben formate** e non. Le prime sono valide secondo le definizioni del sistema, le ultime invece non lo sono e vengono perciò definite "non sensate".

Nel sistema MIU sono ben formate tutte le stringhe che contengono M,I e U: per esempio *IUIUM* o *MII* sono ben formate mentre *EII* non lo è.

Tra le stringhe ben formate possiamo individuare quali sono teoremi. Per esempio *MUIUI* è un teorema in quanto siamo stati in grado di derivarlo.

1.3.3.1 È possibile stabilire se una stringa è un teorema?

Un **criterio** può essere questo: partendo dagli assiomi applichiamo tutte le possibile regole ottenendo nuovi teoremi che mettiamo da parte. Ripetiamo il processo

su questi ultimi ottenendo altri nuovi teoremi e così via in modo da essere certi che tutti i teoremi saranno prima o poi ottenuti tra cui la stringa in questione. Se non la otteniamo significa perciò che non è un teorema.

Questo criterio però presuppone che si abbia un tempo potenzialmente infinito a disposizione per la verifica, il che non è molto ragionevole. Se un criterio di questo tipo richiede invece un tempo accettabile, se non immediato, viene detto **procedura di decisione**

In un sistema formale esiste sempre almeno una procedura di decisione che è quella per riconoscere gli assiomi. In alcuni sistemi infatti esistono infiniti assiomi e non è possibile indicarli tutti: si procede quindi definendo una procedura di decisione per gli assiomi. Per i teoremi invece questa non esiste necessariamente.

1.3.3.2 Problema

Questo sistema è stato pensato per essere un rompicapo: il problema posto è **"*MU* è un teorema del sistema MIU?"** Cioè se è possibile ottenere *MU* applicando ripetutamente le quattro regole partendo da *MI*.

Osservando bene le regole deduciamo che nessuna di essa modifica né il numero né la posizione delle *M* nelle stringhe. Quindi tutte le stringhe che possono essere prodotte, per l'assioma *MI*, iniziano inevitabilmente per *M*. Questo ci permette di scartare le stringhe come *IUIM* ma non *MU* che potrebbe essere ancora producibile.

Allo stesso modo osserviamo che il numero di *I* è lasciato invariato dalla regola 1 e 4, nella regola 2 viene sempre raddoppiato e nella regola 3 diminuito di 3. Poiché partiamo da *MI*, che contiene una *I*, possiamo ottenere stringhe con 2^n o $2^n - 3m$ *I*, con n il numero di applicazioni della regola 2 e m della regola 3.

In *MU* il numero di *I* è zero: per eliminare tutte le *I* dovremmo avere $2^n - 3m = 3x$ che non ammette alcuna soluzione: non è possibile ottenere un multiplo di 3 sottraendo 3 da una potenza di 2.

La soluzione è quindi **no**: non è possibile ottenere *MU*.

1.3.4 pq•

Il sistema pq• è un altro semplice sistema formale come MIU a cui tuttavia è possibile dare un'interpretazione che risulta familiare (in realtà anche più di una).

1.3.4.1 Simboli

Come risulta già chiaro dal nome i simboli consentiti dal sistema sono: p, q, •

1.3.4.2 Assiomi

pq• ammette infiniti assiomi. Per caratterizzarli si utilizza la seguente definizione (una procedura di decisione):
sono assiomi del sistema pq• tutte le stringhe del tipo xp•qx• dove x rappresenta una stringa qualunque di "•".

1.3.4.3 Regola

pq• ha una sola regola di produzione:

1. se xpyqz è un teorema allora anche xpy•qz• è un teorema

dove x,y,z sono stringhe di "•" di lunghezza qualsiasi.

Prendiamo per esempio la stringa ••p•q••• (è un assioma infatti ha forma xp•qx•) e applichiamo la regola per ottenere: ••p••q•••••. Di nuovo e si ottiene: ••p•••q••••••. Partendo invece da •••p•q•••• (altro assioma) produciamo •••p••q•••••, •••p•••q•••••• e così via.

1.3.4.4 Procedura di decisione

A differenza del precedente sistema MIU in cui a seconda della regola si produce una stringa più lunga o più corta della precedente si nota subito che nel sistema pq• si ricavano teoremi via via **più lunghi**. Questa caratteristica determina l'**esistenza** di una **procedura di decisione** per i propri teoremi.

A questo punto la procedura di decisione di pq• dovrebbe essere già stata individuata. È la seguente:
xpyqz è un teorema del sistema pq• se e solo se il numero di "•" in z è uguale alla somma del numeri di "•" in x e in y

Se indichiamo la lunghezza di una stringa x di "•" con $\sharp x$ allora possiamo riscrivere il criterio così:
xpyqz è un teorema del sistema pq• se e solo se $\sharp x + \sharp y = \sharp z$

1.3.5 Isomorfismo

Il sistema pq• ha una proprietà particolare: produce solo teoremi che soddisfano i criteri dell'**addizione**. Possiamo interpretare il teorema "•••p••q•••••" come "3 più 2 uguale 5". In generale xpyqz come $x + y = z$. Non ha caso "p" è stato scelto perchè il suo suono ricorda "plus"e "q" "equals". Questa relazione tra teoremi pq• e addizioni è un *isomorfismo*. Un isomorfismo è una particolare trasformazione tra strutture complesse in cui le relazioni interne tra le parti sono preservate.

1.3.6 Interpretazione e significato

L'isomorfismo quindi causa l'acquisizione di un **significato** da parte dei simboli del sistema:

simbolo	significato
p	più
q	è uguale a
•	uno
• •	due
• • •	tre
...	...

Se potessimo leggere ogni possibile teorema pq in questo modo troveremmo che la sua interpretazione risulta immancabilmente vera. A questo punto viene naturale fare anche il processo contrario: cioè si potrebbe pensare che poichè "cinque più due è uguale a sette" la stringa "• • • • • p • • q • • • • • • • " è un teorema. Certamente lo è però questa non è una giustificazione valida infatti possiamo dire che una stringa è un teorema soltanto dopo averne trovato la dimostrazione seguendo le regole del sistema formale.

È ovviamente possibile uscire dalle costrizioni del sistema per ragionarci su, dare significati ai simboli, e quant'altro ma non ci è poi concesso di utilizzare nuove regole o considerare teoremi alcune stringhe sulla base delle interpretazioni trovate. Il significato conferito al sistema formale deve sempre rimanere **passivo**. Infatti si può trovare più di un isomorfismo e le interpretazioni da questi prodotte potrebbero essere anche contrastanti: per questo motivo bisogna sempre attenersi alle regole del sistema formale (**requisito di formalità**).

Un'altra interpretazione che risulta altrettanto valida del sistema pq è questa:

simbolo	significato
p	è uguale a
q	sottratto da
•	uno
• •	due
• • •	tre
...	...

• • • p • • q • • • • • ↔ "tre è uguale a due sottratto da cinque"

1.3.7 Coerenza e Completezza

Una volta introdotta un'interpretazione per un sistema formale si può parlare di **coerenza** del sistema. Esistono due diverse definizioni di coerenza:

1. Un sistema formale si dice *coerente* se ogni teorema una volta interpretato risulta essere una proposizione vera.
2. Un sistema formale si dice *coerente* se tutti i suoi teoremi una volta interpretati sono tra loro compatibili.

Nel primo caso si parla di **coerenza esterna**. Il sistema formale risulta coerente se le affermazioni prodotte dalla sua interpretazione ottengono un riscontro nel mondo esterno, la nostra realtà. La seconda definizione è invece quella di **coerenza interna**. Questa coerenza non necessita di una conformità con il nostro mondo bensì con un qualunque possibile mondo immaginabile. È sufficiente che non vi siano **contraddizioni** al suo interno: non possono esistere per esempio due teoremi che si escludano a vicenda o uno che affermi l'opposto dell'altro. Sarà sempre intesa quest'ultimo tipo quando si parlerà di coerenza più avanti.

Complementare al concetto di coerenza vi è quello di **completezza** del sistema formale. È così definito:
Un sistema formale è *completo* se qualunque proposizione vera **esprimibile** in una stringa ben formata risulta essere un teorema.

È importante evidenziare il fatto che debba essere esprimibile nei termini del sistema formale: il sistema si occupa solo di un certo ambito della realtà di sua competenza ed è completo se è in grado in esprimerla nella sua interezza. Altrimenti solo un sistema formale "universale" potrebbe essere completo.

Anche in questa definizione il valore di verità è riferibile alla nostra realtà ma si utilizzerà il termine "completo" sempre rispetto a mondo immaginabile coerente.

È fondamentale capire che entrambi i concetti di coerenza e completezza non sono proprietà del sistema formale ma **relativi** alla sua interpretazione: nonostante il sistema acquisisca inevitabilmente un significato passivo rimane comunque un insieme di regole di manipolazione di simboli di per sé senza senso.

1.3.8 Calcolo proposizionale

Il calcolo proposizionale è una formalizzazione della **logica delle proposizioni**. Ovvero lo studio del valore di verità delle proposizioni complesse.

1.3.8.1 Simboli

Sono consentite le tre lettere P, Q, R, i simboli $\wedge$, $\vee$, $\rangle$, $\langle$, $\sim$, $\supset$ e l'apice $'$.

Le seguenti stringhe sono ben formate:

1. $\sim x$
2. $\langle x \wedge y \rangle$
3. $\langle x \vee y \rangle$
4. $\langle x \supset y \rangle$

dove x e y sono stringhe ben formate.

P, Q e R sono detti **atomi** e corrispondono a proposizioni semplici. Si possono formare ulteriori atomi aggiungendo degli apici a quest'ultimi. Per esempio P', Q'', R'. Tutti gli atomi sono stringhe ben formate.

1.3.8.2 Regole di produzione

1. **regola dell'unione**: se x e y sono teoremi allora $\langle x \wedge y \rangle$ è un teorema
2. **regola della separazione**: se $\langle x \wedge y \rangle$ è un teorema allora entrambi x e y sono teoremi.
3. **regola della doppia tilde**: la stringa "$\sim\sim$" può essere cancellata ovunque occorre in un teorema oppure inserita per ottenere una stringa ben formata.
4. **regola della fantasia**: se è possibile dimostrare y assumendo che x sia un teorema allora $\langle x \supset y \rangle$ è un teorema.
5. **regola del riporto**: all'interno di una *fantasia* si può utilizzare qualunque teorema del livello immediatamente esterno.
6. **regola del distaccamento**: se sia x che $\langle x \supset y \rangle$ sono teoremi allora y è un teorema.
7. **regola dell'antitesi**: $\langle x \supset y \rangle$ e $\langle \sim y \supset \sim x \rangle$ sono intercambiabili.
8. **regola di De Morgan**: $\langle \sim x \wedge \sim y \rangle$ e $\sim \langle x \vee y \rangle$ sono intercambiabili.
9. **regola di Switcheroo** (modus tollendo ponens): $\langle x \vee y \rangle$ e $\langle \sim x \supset y \rangle$ sono intercambiabili.

1.3.8.3 Assiomi

La particolarità di questo sistema è di non avere alcun assioma. A differenza dei sistemi precedenti, come pq il cui significato risiede quasi esclusivamente negli assiomi scelti, nel calcolo proposizionale sono le regole che determinano tutto il sistema. Come è possibile allora produrre teoremi se non abbiamo niente da cui partire? Semplice: usiamo la regola della **fantasia** che permette di produrre teoremi "dal nulla".

1.3.8.4 Dimostrazioni

Ecco un esempio di dimostrazione:

1.	[	
2.	$\langle P \vee Q \rangle$	premessa
3.	$\langle \sim P \supset Q \rangle$	Switcheroo
4.	$\langle \sim Q \supset \sim\sim P \rangle$	antitesi
5.	$\langle \sim Q \supset P \rangle$	doppia tilde
6.	$\langle Q \vee P \rangle$	Switcheroo
7.	]	
8.	$\langle \langle P \vee Q \rangle \supset \langle Q \vee P \rangle \rangle$	fantasia

Le parentesi quadrate segnano l'inizio e il termine della fantasia. In questo caso partendo dalla premessa che $\langle P \vee Q \rangle$ sia un teorema abbiamo ottenuto che anche $\langle Q \vee P \rangle$ è un teorema. Quindi una volta fuori dalla fantasia possiamo riassumere il processo compiuto nel nuovo **teorema** $\langle \langle P \vee Q \rangle \supset \langle Q \vee P \rangle \rangle$.

È possibile aprire più fantasie una nell'altra ed usare la regola del riporto così:

1.	[	
2.	$\langle P \wedge \sim P \rangle$	premessa
3.	$\sim P$	separazione
4.	P	separazione
5.	[	
6.	$\sim Q$	premessa
7.	P	riporto da riga 3
8.	$\sim\sim P$	doppia tilde
9.	]	
10.	$\langle \sim Q \supset \sim\sim P \rangle$	fantasia
11.	$\langle \sim P \supset \sim\sim Q \rangle$	antitesi
12.	$\langle \sim P \supset Q \rangle$	doppia tilde
13.	Q	distaccamento riga 3 e 12
14.	]	
15.	$\langle \langle P \wedge \sim P \rangle \supset Q \rangle$	fantasia

1.3.8.5 Interpretazione del sistema

Come già detto questo sistema formalizza la logica proposizionale. Di seguito vi è un elenco di simboli e alcuni teoremi con le loro interpretazioni anche se a questo punto saranno già chiare:

sistema	significato
P, Q, R	proposizioni semplici

sistema	significato
$\sim$	non, negazione
$\wedge$	e, congiunzione
$\vee$	o, disgiunzione
$\supset$	implica che
$\sim \langle P \wedge Q \rangle$	non P e Q
$\sim \langle \langle P \vee \sim Q \rangle \supset \langle \sim P \vee Q \rangle \rangle$	non P o non Q implica non P o Q

Il teorema prima dimostrato $\langle \langle P \wedge \sim P \rangle \supset Q \rangle$ è il noto **principio di esplosione** o *ex falso quodlibet*; che è interpretabile come "P o non P implica Q". Se ammettiamo che sia P sia sua negazione siano vere dobbiamo ammetere anche che qualunque altra proposizione Q sia vera! Aver dimostrato questo teorema significa che nel calcolo proposizionale non esistono contraddizioni. Se non fosse così sarebbe possibile dimostrare qualunque cosa e l'intero sistema (la sua interpretazione in realtà) cadrebbe. Dobbiamo sempre ricordare che per il sistema formale $\langle \langle P \wedge \sim P \rangle \supset Q \rangle$ è una semplice sequenza di simboli trasformabile secondo delle regole ed di per sè priva d'ogni significato.

1.3.8.6 Coerenza e metateorie

È sufficiente il fatto appena scoperto ad assicurarci che il calcolo proposizionale sia coerente? Assolutamente no, sarebbe necessaria una prova dell'assenza di contraddizioni. Tuttavia questa prova non è possibile tramite una derivazione di qualche teorema all'interno del calcolo proposizionale. C'è una grossa differenza tra una *derivazione* di un teorema e una *prova* (dimostrazione). Una dimostrazione è qualcosa di **informale**, una costruzione umana, mentre una derivazione è un'applicazione **meccanica** di regole.

Per dimostrare la coerenza e la completezza del calcolo proposizionale è necessario uscire dal sistema ed elaborare una teoria sulla teoria. Questa è una **metateoria** costituita da teoremi che parlano di teoremi. Anche una volta elaborata la metateoria e ottenuta la dimostrazione cercata chi ci assicura però che la metateoria sia coerente e completa? È necessaria una prova del fatto che va cercata in una **metametateoria** e di quest'ultima allora? Anche formalizzando le metateorie non si risolverebbe il problema: ad un certo punto è necessario arrendersi ed accettare il fatto che questa teoria sia coerente nella sua interpretazione in quanto evidente. In fondo dubitare del calcolo proposizionale è dubitare della logica stessa e della propria mente.

1.3.9 Teoria dei Numeri Tipografica (TNT)

Per poter scrivere la proposizione usata da Gödel nel primo teorema di incompletezza è necessario un sistema capace di esprimere le **proprietà dei numeri**

oltre che a quelle della logica proposizionale. Per questo useremo una formalizzazione della teoria dei numeri: la teoria dei numeri tipografica (**TNT** in breve)

Questo sistema formale utilizza tutti i precedenti simboli e regole del calcolo proposizionale oltre a definire i numeri naturali e le loro proprietà tramite tramite **l'arimetica di Peano**.

1.3.9.1 Numerali

I numerali sono stringhe che rappresentano numeri. A differenza del familiare sistema posizionale e delle cifre indoarabe si ricorre soltanto a due simboli: S e O per poter scrivere qualsiasi numero naturale.

Numerale	Interpretazione
O	zero
SO	uno
SSO	due
$SSSO$	tre
...	...

O rappresenta quindi il numero zero e S e il numero successore di un qualunque numero.

1.3.9.2 Variabili

Per riferirsi a numeri non specifici in una formula si utilizzano delle variabili. Utilizziamo una lettera a e come nel calcolo proposizionale l'apice $'$ per produrre nuove variabili a', a'', a''', ...

1.3.9.3 Operazioni

Le operazioni consentite sono l'addizione e la moltiplicazione. Si utilizzano i normali simboli $+$ e $\times$ e le parentesi tonde. Per esempio $((a \times a'') + a')$.

Sono entrambe operazioni binarie e sono sempre necessarie le parentesi: non è possibile scrivere $a + a' + a''$ come si fa comunemente.

1.3.9.4 Formule aperte e formule chiuse

Non tutte le stringhe TNT sono proposizioni. Le stringhe la cui interpretazione non ha un valore di verità sono dette **formule aperte** o predicati, in quanto sono analoghe ad una frase senza soggetto, e contengono sempre **varibili libere**.

Per esempio la stringa $SSO = Sa$ in cui a è la variabile libera. La stringa fa un' affermazione su un generico numero a e non è né vera né falsa: esprime semplicemente una proprietà.

Una variabile è **legata** quando invece è associata ad un quantificatore. In questo caso la stringa fa un'affermazione sull'esistenza di un numero o una proprietà valida per tutti i numeri e si dice **formula chiusa**. Esistono due quantificatori: quello universale $\forall$ (pronunciato "per ogni") e quello esistenziale $\exists$ (pronunciato "esiste almeno").

Per esempio nella stringa $\forall a : Sa = O$ la variabile a è legata e la stringa è sicuramente falsa.

1.3.9.5 Formule ben formate

Ecco le regole per stabilire se una formula è ben formata:

1. **Numerali**: sono formule ben formate tutti i numerali
2. **Variabili**: sono formule ben formate tutte le variabili
3. **Termini**:
   - sono termini tutti i numerali e le variabili
   - tutti i termini preceduti da S sono termini
   - $(x \times y)$ e $(x + y)$ sono termini se x e y sono termini
   - tutti i **termini** sono formule **ben formate**
4. **Atomi**: se x e y sono termini allora x = y è un atomo
5. **Molecole**: se x e y sono formule ben formate sono ben formate:
   - $\sim x$
   - $\langle x \wedge y \rangle$
   - $\langle x \vee y \rangle$
   - $\langle x \supset y \rangle$
6. **Quantificazioni**: sono ben formate le formule: $\forall v : x$ e $\exists v : x$ con v una variabile e x una formula aperte in cui compare v come variabile libera

1.3.9.6 Assiomi

TNT definisce 5 assiomi:

1. $\forall a :\sim Sa = O$
2. $\forall a : (a + O) = a$
3. $\forall a : \forall a' : (a + Sa') = S(a + a')$
4. $\forall a : (a \times O) = O$
5. $\forall a : \forall a' : (a \times Sa') = ((a \times a') + a)$

L'assioma 1 afferma che non esiste un numero il cui successore è zero: cioè non esistono numeri minori di zero. Gli assiomi 2 e 4 affermano che lo zero è l'elemento neutro dell'addizione e assorbente della moltiplicazione. Gli assiomi 3 e 5 affermano le proprietà dell'addizione e della moltiplicazione.

Da notare che il primo assioma è uno dei postulati di Peano.

1.3.9.7 Regole di produzione

1. **regola della specificazione**: sia x una formula in cui compare la variabile v e t un termine in cui non compaiono variabile legate in x: se $\forall v : x$ è un teorema allora x e tutte le formule ottenute sostituendo v con t sono teoremi
2. **regola della generalizzazione**: se x è un teorema in cui compare la variabile libera v allora $\forall v : x$ è un teorema. (Attenzione: **Non** è possibile generalizzare una variabile libera posta come premessa in una fantasia)
3. **regola dello scambio**: sia v una variabile: le stringhe "$\forall v :\sim$" e "$\sim \exists v :$" sono intercambiabili ovunque occorrano in un teorema
4. **regola dell'esistenza**: sia x un teorema in cui è presente un termine t e y la formula ottenuta sostituendo t con una variabile v: $\exists v : y$ è un teorema
5. **regole dell'uguaglianza**:
    1. simmetria: siano t e s due termini: se $t = s$ è un teorema allora $s = t$ è un teorema
    2. transitività: siano t, s, r tre termini: se $t = s$ e $s = r$ sono teoremi allora $t = r$ è un teorema
6. **regole del successore**:
    1. siano t e s due termini: se $t = s$ è un teorema allora $Ss = St$ è un teorema
    2. siano t e s due termini: se $Ss = St$ è un teorema allora $s = t$ è un teorema

1.3.9.8 Esempi di proposizioni e dimostrazioni:

La proposizione "esiste un numero pari diverso da zero" è traducibile in TNT come

$$\exists a : \exists a' : Sa = (Sa' \times SSO)$$

Usando S è possibile fissare un valore minimo che a può assumere: in questo caso SSO. Usiamo lo stesso metodo per scrivere "undici è un numero primo":

$$\sim \exists a : \exists a' : (SSa \times SSa') = SSSSSSSSSSO$$

Una più complessa "non è possibile separare un cubo in due cubi":

$$\forall a : \forall a' : \forall a'' :\sim (((a \times a) \times a) + ((a' \times a') \times a')) = ((a'' \times a'') \times a'')$$

Il sistema è sicuramente molto espressivo tuttavia diventa già molto complicato scrivere proposizioni come "a è una potenza di b" per la mancanza di operatori.

Di seguito la dimostrazione di $1 + 1 = 2$:

1. $\forall a : \forall a' : (a + Sa') = S(a + a')$	assioma 3
2. $\forall a' : (SSO + Sa') = S(SSO + a')$	specificazione: a, SSO
3. $(SSO + SO) = S(SSO + SO)$	specificazione: a', SO
4. $\forall a : S(a + O) = a$	assioma 2
5. $S(SO + O) = SO$	specificazione: a, SO
6. $SS(SO + O) = SSO$	successore 1
7. $(SO + SO) = SSO$	transitività linee 3, 6

1.3.9.9 ω-incompletezza e ω-incoerenza

Una cosa particolare di questa sistema è che spesso capita di poter dimostrare una serie di teoremi ma non è possibile dimostrare la formula che li riassume tutti. Per esempio potremmo riuscire a dimostrare:

$$(SO \times SO) = SO$$
$$(SSO \times SO) = SSO$$
$$(SSSO \times SO) = SSSO$$
$$(SSSSO \times SO) = SSSSO$$
$$\dots$$

ma non la formula $\forall a : (a \times SO) = a$. Per questo fatto la teoria si dice **ω-incompleta**.

In generale un sistema formale S si dice ω-incompleto quando esiste una formula $\varphi(x)$ tale che $\varphi(n)$ è un teorema in S per ogni numero naturale n ma $\forall x : \varphi(x)$ non è un teorema. Con ω infatti si intende la totalità dei numeri naturali.

Se non dovesse essere un teorema neanche la sua negazione $\forall x :\sim \varphi(x)$ φ è indecidibile nel sistema S che si dice allora **ω-incoerente**.

L'ω-incoerenza è essenzialmente diversa dall'incoerenza: solo una delle due formule tra $\varphi(n)$ e $\sim \varphi(n)$ risulta un teorema per tutti gli n tuttavia nella loro espressione generica potremmo accettarle entrambe come assiomi. Per fare ciò dobbiamo ammettere che esistano alcuni numeri che verificano $\sim \varphi$ ma poichè

φ ci dice che nessun numero naturale ha tale la proprietà questi numeri non possono essere dei numeri normali.

Se ammettiamo l'esistenza di tali numeri, noti come numeri naturali generalizzati o **soprannaturali**, entriamo nel campo della **teoria dei numeri non-standard** in cui $\sim \varphi$ è un teorema mentre φ è una proprietà di alcuni numeri (i naturali) e non c'è una contraddizione.

1.3.9.10 Induzione

Per rendere TNT ω-completa è necessario un meccanismo che ci permetta di rendere φ un teorema. Per fare ciò formalizziamo il quinto postulato di Peano noto come **principio di induzione** matematico e lo aggiungiamo alle regole. Quindi:

7. **regola di induzione**: sia v una variabile e $\varphi(v)$ una formula aperta che contiene v libera: se $\forall v : \langle \varphi(v) \supset \varphi(Sv) \rangle$ e $\varphi(O)$ sono teoremi allora $\forall v : \varphi(v)$ è un teorema

1.3.10 Conclusioni

Con quest'ultima regola TNT è diventata ω-completa ed equivalente al sistema formale di *Principia Mathematica*. Se TNT fosse anche completa ogni possibile proposizione vera della teoria dei numeri esprimibile in in una formula sarebbe un teorema e potremmo determinare meccanicamente tramite una procedura di decisione se una formula qualunque è un teorema. In altre parole i matematici rimarrebbero senza lavoro perché basterebbe una macchina che enumeri tutti i teoremi TNT e controlli che la stringa fornitale sia presente o no. Vediamo allora perché questo non è accaduto.

1.4 La dimostrazione

Ora che abbiamo definito il sistema formale e le basi che ci servono possiamo passare a capire la dimostrazione del primo teorema di incompletezza in cui Gödel ha dimostrato che se una teoria è coerente non può essere anche completa.

1.4.0.1 Antinomie

L'idea centrale è quella di costruire un'**antinomia** all'interno di TNT: cioè una proposizione che non può essere né vera né falsa, una contraddizione. In campo linguistico per esempio c'è l'antichissimo *Paradosso di Epimenide* o **paradosso del mentitore**: "Epimenide di Creta afferma che tutti i cretesi mentono" o la versione moderna "questa frase è falsa". Se la frase fosse vera essa afferma di

essere falsa portando ad una contraddizione, se fosse invece falsa di conseguenza l'opposto di quanto dice deve essere vero cioè che è vera portando ad una **contraddizione**.

Quindi, ironia della sorte, il principio scoperto da Russell con la sua antinomia nella teoria degli insiemi è lo stesso che è stato usato contro di lui da Gödel per far cadere *Principia Mathematica*.

Queste antinomie si basano sul fatto che il soggetto del predicato è la frase stessa: si dice che sono **autoreferenziali**. Per ottenere lo stesso effetto nella teoria dei numeri Gödel ha prima dovuto trovare un modo per scrivere formule TNT che parlano di formule TNT e poi ha potuto scrivere la famosa **formula G**, il cui significato è "**G** non è un teorema TNT".

1.4.1 Numero di Gödel

È possibile includere ogni problema di un qualunque sistema formale nella teoria dei numeri tramite uno speciale isomorfismo scoperto da Gödel noto come gödellizzazione o *numero di Gödel*.

1.4.1.1 MIU

Si può per esempio compiere questa operazione con il problema "MU è un teorema del sistema MIU?".

Il primo passo è **associare un numero**, scelto arbitrariamente, ad ogni simbolo del sistema.

$$M \leftrightarrow 3$$
$$I \leftrightarrow 1$$
$$U \leftrightarrow 0$$

Così facendo si è creata una **corrispondenza** tra le **stringhe** MIU e **numeri**:

$$MIU \leftrightarrow 310$$
$$MU \leftrightarrow 30$$
$$MIII \leftrightarrow 3111$$
$$MIUI \leftrightarrow 3101$$
$$\dots$$

Allo stesso modo possiamo riscrivere la **regole** di produzione tipografiche (Stringa $\mapsto$ Stringa) come regole aritmetiche, cioè **funzioni** (Numero $\mapsto$ Numero):

1. La regola "Se xI è un teorema allora lo è anche xIU" diventa "se $10m+1$ è un numero MIU lo è anche $10(10m+1)$". Scegliendo $U \leftrightarrow 0$ questa regola è una semplice moltiplicazione per 10.

2. "Se Mx è un teorema allora lo è anche Mxx" diventa "Se $3 \cdot 10^m + n$ è un numero MIU allora lo è anche $10^m \cdot (3 \cdot 10^m + n) + n$"

3. Questa regola è un po' più complicata: "Se $xIIIy$ è un teorema allora lo è anche xUy" viene riscritta come "Se $k \cdot 10^{m+3} + 111 \cdot 10^m + n$ è un numero MIU lo è anche $k \cdot 10^{m+1} + n$".

4. In maniera simile alla precedente "Se $xUUy$ è un teorema allora lo è anche xy" diventa "Se $k \cdot 10^{m+2} + n$ è un numero MIU allora lo è anche $k \cdot 10^m + n$".

dove m e k sono numeri naturali qualsiasi e $n < 10^m$

Il problema diventa quindi: **"30 è un numero MIU?"**. Poiché l'essere un numero MIU è una proprietà dei numeri definita in modo ricorsivo (tramite l'applicazione di funzioni) è sicuramente possibile **esprimere in TNT** (sebbene in modo incredibilmente complicato) questa proposizione tramite una formula e verificare se essa sia un teorema o meno. Possiamo quindi concludere che:
ogni problema in un sistema formale può essere trasformato in un proposizione nella teoria dei numeri e convertito in una formula TNT.

1.4.1.2 TNT

Poiché abbiamo detto che questa proprietà vale per qualsiasi sistema formale deve valere anche per TNT stessa. Possiamo quindi convertire un problema come il chiedersi se una formula TNT sia un teorema in una formula TNT e a sua volta quest'ultima e così via. Significa che TNT contiene stringhe che parlano di se stessa: il sistema si morde la coda!

Questa non è una particolarità di TNT ma avviene per tutti i sistemi abbastanza complessi da definire l'aritmetica.

Iniziamo come per MIU ad associare dei numeri ai simboli del sistema:

$$
\begin{aligned}
O &\leftrightarrow 10 \\
S &\leftrightarrow 11 \\
= &\leftrightarrow 12 \\
+ &\leftrightarrow 13 \\
\times &\leftrightarrow 14 \\
(&\leftrightarrow 15 \\
) &\leftrightarrow 16 \\
\langle &\leftrightarrow 17 \\
\rangle &\leftrightarrow 18 \\
a &\leftrightarrow 19 \\
' &\leftrightarrow 20 \\
\wedge &\leftrightarrow 21 \\
\vee &\leftrightarrow 22 \\
\supset &\leftrightarrow 23 \\
\sim &\leftrightarrow 24 \\
\exists &\leftrightarrow 25 \\
\forall &\leftrightarrow 26 \\
: &\leftrightarrow 27
\end{aligned}
$$

Possiamo quindi associare un numero anche ad ogni formula. Per esempio sostituendo ogni simbolo della formula $\sim \exists a : \exists a' : \langle \sim a = a' \supset Sa = Sa' \rangle$ si ottiene il numero:

$$24251927251920271724191219202311191211192018$$

Questo numero è il *numero di Gödel* della formula.

Possiamo trasformare tutte le regole di produzione in funzioni aritmetiche esattamente come per MIU tuttavia sarebbe un' operazione veramente onerosa e di poco interesse: sappiamo già che esse svolgono esattamente lo stesso compito soltanto in maniera differente.

1.4.1.3 Coppie dimostrative

Nonostante tutto questo non siamo ancora in grado di esprimere in una formula "x è un teorema": per fare questo dobbiamo definire il concetto di **coppia dimostrativa**.

Sappiamo che una stringa in qualunque sistema formale è un teorema se è possibile derivarla utilizzando le regole di produzione, cioè se esiste una sua dimostrazione: una **sequenza di stringhe e regole utilizzate**. Poiché abbiamo

associato numeri e funzioni corrispondenti a stringhe e regole tipografiche possiamo allo stesso modo creare un' ulteriore relazione che consiste tra numeri e dimostrazioni. Una **dimostrazione** corrisponde semplicemente al **numero** di Gödel della stringa ottenuta concatenando tutti i suoi passaggi.

Una coppia dimostrativa è una coppia di numeri di cui uno è la dimostrazione dell'altro secondo la definizione precedente. Questa è quindi una **proprietà** di alcuni numeri e perciò può essere espressa da una formula aperta in TNT. Indichiamo tale formula con la notazione $P(a, a')$ dove a e a' sono le due variabili libere.

Per esempio per la dimostrazione di $1 + 1 = 2$

1. $\forall a : \forall a' : (a + Sa') = S(a + a')$	261927261920271519131119201612111519 . . .
2. $\forall a' : (SSO + Sa') = S(SSO + a')$	2619202715111110131119201612111511111 . . .
3. $(SSO + SO) = S(SSO + SO)$	151111101311101612111511111013111016
4. $\forall a : S(a + O) = a$	2619271115191310161219
5. $S(SO + O) = SO$	11151110131016121110
6. $SS(SO + O) = SSO$	1111151110131016121111110
7. $(SO + SO) = SSO$	1511101311101612111110

il mostruoso numero 26192726192027151913 11. . . e 1511101311101612111110 formano una coppia dimostrativa.

Possiamo finalmente scrivere $\exists a' : P(a', x)$ dove x è il numerale di un numero di Gödel: interpretabile come **"esiste una dimostrazione di x"**.

1.4.1.4 Sostituzione

Nel precedente esempio vediamo che a sinistra avviene il solito processo di applicazione di regole e trasformazione di stringhe e allo stesso modo avviene un analogo processo tra numeri e funzioni a destra. Nel primo passo della dimostrazione in particolare avviene la sostituzione di una variabile con un numerale per ottenere una nuova formula:

1. $\forall a : \forall a' : (a + Sa') = S(a + a')$
2. SSO
3. $\forall a : \forall a' : (SSO + Sa') = S(SSO + a')$

Lo stesso avviene secondo qualche procedimento aritmetico tra i corrispettivi numeri di Gödel:

1. 261927261920271519131119201612111519131920 16
2. 111110
3. 2619272619202715111110131119201612111511111013192016

Possiamo dire, come per le coppie dimostrative, che questi tre numeri hanno la proprietà di essere una **sostituzione**. Poiché TNT può rappresentare ogni primitiva ricorsiva deve esistere una formula aperta anche per questa proprietà che chiamiamo $S(a, a', a'')$.

1.4.2 Paradosso di Quine

Con il numero di Gödel abbiamo lo strumento che ci permette di scrivere una formula TNT che parla di altre formule o teoremi TNT. Tuttavia per poter scrivere la formula **G** ci serve un modo per fare che la formula **parli di sé**. Sfortunatamente questo non è fattibile in modo diretto: una formula che contenga il proprio numero di Gödel sarebbe di lunghezza infinita e quindi impossibile da costruire. È però possibile ottenere lo stesso risultato tramite un'**autoreferenza indiretta** scoperta dal logico americano Quine. Il paradosso da lui proposto è la seguente espressione:

"produce una falsità quando preceduto dalla sua citazione" produce una falsità quando preceduto dalla sua citazione.

Per capire dove sta la contraddizione analizziamo i componenti della frase.

1.4.2.1 Predicati

Un **predicato** è una frase in cui è assente il **soggetto**. Non è una proposizione completa e non ha un valore di verità (esattamente come le formule aperte in TNT). Per esempio "_____ è una città" a cui aggiungendo un soggetto es. "Roma" otteniamo "Roma è una città" che è un'affermazione vera.

Quindi un predicato diventa un'**affermazione** quando gli viene preposto un soggetto qualunque, fra i tanti possibili vi è anche il predicato stesso. È possibile quindi avere la frase: "è una città" è una città, che è un'affermazione sicuramente falsa (quando indichiamo tra virgolette una parola ci riferiamo alla parola stessa non al suo significato). Chiamiamo in generale *quine* un frase ottenuta ponendo come soggetto di un predicato la citazione del predicato stesso.

1.4.2.2 Autoreferenza indiretta

Nella frase proposta da Quine abbiamo un predicato che chiamiamo **P**:

_____ produce una falsità quando preceduto dalla sua citazione.

e la proposizione che è il quine di **P** che chiamiamo **Q**:

"produce una falsità quando preceduto dalla sua citazione" produce una falsità quando preceduto dalla sua citazione.

P si riferisce al suo soggetto che ora è **Q** e la frase **Q** si riferisce al suo soggetto cioè **P**. Quindi **Q** tramite **P** parla di se stessa! E non finisce qui perché dicendo "produce una falsità" essa afferma anche di essere falsa. È quindi una proposizione equivalente a "Questa frase è falsa" ma che non usa direttamente un pronome per parlare di sé: proprio quello che dovrà essere la formula **G**.

1.4.2.3 Quine aritmetico

L'analogo in TNT di porre come soggetto di un predicato il predicato stesso è **sostituire** in una variabile libera di una formula aperta il numero di Gödel (il suo numerale per la precisione) della formula stessa.

Per esempio prendiamo la formula aperta $a = (SSO \times a')$ e calcoliamone il numero di Gödel: 19121511111014192016. Il quine di tale formula è quindi

$$\underbrace{SSS\ldots O}_{19121511111014192016 \text{ volte } S} = (SSO \times a')$$

Questa nuova formula ha ovviamente un numero (mostruosamente enorme) di Gödel:

$$\underbrace{11111111\ldots}_{38243022222028380000 \text{ volte } 1}19121511111014192016$$

Il **quine aritmetico** è quindi una proprietà tra una coppia di numeri ed è un particolare tipo di sostituzione $S(a, a, a')$ dove a è il numero di Gödel originale e a' quello del quine. Indichiamo allora la formula TNT per questa proprietà come $Q(a, a')$.

1.4.3 G

Finalmente abbiamo tutto il necessario per costruire la formula **G**. Partiamo dalla seguente proposizione:

$$\sim \exists a : \exists a' : \langle P(a, a') \wedge Q(a'', a') \rangle$$

Chiamiamo g il suo numero di Gödel. Dell'espansione decimale di g conosciamo la parte iniziale, il centro e il termine:

$$g = 242519272519202717\ldots 21\ldots 18$$

G è il quine di tale proposizione che otteniamo quindi inserendo il numerale di g al posto di a''. Ed ecco **G**:

$$\sim \exists a : \exists a' : \langle P(a, a') \wedge Q(\underbrace{SSS \ldots O}_{g \text{ volte } S}, a') \rangle$$

1.4.3.1 Interpretazione

Una sua traduzione letterale è "Non esistono due numeri a e a' tali da formare una coppia dimostrativa di cui uno è il quine di g". Cioè non esiste alcun numero che forma una coppia dimostrativa con il quine di g. Poiché il quine di g è la formula stessa questo significa che **G** afferma che **G non è dimostrabile**.

A questo punto dobbiamo chiederci quali sono le implicazioni. Se **G** fosse un teorema significa che la sua interpretazione deve essere vera. Ma **G** afferma di non essere un teorema portando quindi ad una contraddizione. Allora significa che **G** non è un teorema. In questo caso non ci sarebbe alcuna contraddizione ma **G** sarebbe una proposizione vera che non è un teorema e quindi TNT non è più un sistema formale completo. Inoltre la formula **G** è indecidibile in TNT in quanto né **G** né $\sim$**G** possono essere teoremi.

1.4.3.2 Conclusione

Ne concludiamo che **qualsiasi sistema formale sufficientemente complesso da contenere l'aritmetica se è coerente non può essere anche completo**. Esisterà sempre qualche verità che non può essere provata all'interno del sistema.

1.4.3.3 Osservazione

Abbiamo visto che **G** è indecidibile: come TNT senza regola d'induzione anche questa formulazione della teoria dei numeri è ω-incoerente. Potremmo allora decidere di prendere **G** come assioma per eliminare il problema ma la teoria rimarrebbe ugualmente ω-incoerente: nel momento in cui abbiamo assunto **G** come verità possiamo costruire una nuova formula **G**$'$ anch'essa indecidibile. Se invece decidessimo di aggiungere $\sim$**G** agli assiomi ammetteremmo l'esistenza di alcuni numeri che "formano una dimostrativa con il quine di g". Questi numeri sarebbero dei numeri soprannaturali. In ogni caso TNT rimane ω-incoerente.

1.4.4 Il secondo teorema

Il secondo teorema di incompletezza è un corollario del teorema appena dimostrato che riguarda invece la **coerenza**.

Proviamo a costruire la proposizione "TNT è coerente" all'interno di TNT. Dire che una teoria è coerente significa che non possono essere teoremi sia x che $\sim x$. Se fossero entrambi veri, per il calcolo proposizionale (principio di esplosione),

tutte le formule ben formate sarebbero teoremi. Per provare che TNT è coerente basterebbe quindi trovare per un solo non-teorema la dimostrazione che effettivamente non è un teorema. Per esempio di $\forall a : Sa = O$.

Possiamo quindi scrivere:

$$\sim \exists a : P(a, \underbrace{SS\ldots O}_{g \text{ volte } S})$$

dove $g = 26192711191210$ (numero di Gödel di $\forall a : Sa = O$)

Gödel è riuscito a dimostrare che una proposizione come questa per essere un teorema TNT deve necessariamente essere incoerente: cioè TNT è coerente se e solo se è incoerente. Quindi per assurdo ha dimostrato che **non è possibile dimostrare la coerenza di un sistema formale al suo interno**.

1.5 Allegati

1.5.1 MIU.hs

Un programma haskell che implementa la procedura di decisione "non molto ragionevole" per i teoremi MIU:

```
import Text.Regex (subRegex, mkRegex)

type Theorem = String
type Rule    = String -> String

axiom :: String
axiom = "MI"

rules :: [Rule]
rules = map (uncurry rewrite) transform
  where
    rewrite = flip . subRegex . mkRegex
    transform = [ ("I$"    ,      "IU")
                , ("M(.+)", "M\\1\\1")
                , ("III"   ,       "U")
                , ("UU"    ,        "") ]

theorems :: [Theorem]
theorems = concat (deriveFrom [axiom])
  where deriveFrom = iterate (rules <*>)

isTheorem :: String -> Bool
```

```
isTheorem = (`elem` theorems)

main :: IO ()
main = print (isTheorem "MIIUIIU")
```

1.5.1.1 MIU-derive.hs

Un altro programma relativo al sistema MIU che produce tutte le possibili dimostrazioni, specificata una lunghezza massima, per una stringa:

```
import Control.Monad.State
import System.Random
import Text.Regex           (subRegex, mkRegex)
import Text.Printf          (printf)
import Data.List            (nub)

type Theorem = String
type Rule    = String
type Rand    = State StdGen

axiom :: String
axiom = "MI"

rules :: [(String, String -> String)]
rules = zip ["I", "II", "III", "IV"] fs
  where
    fs = map (uncurry rewrite) transform
    rewrite = flip . subRegex . mkRegex
    transform = [ ("I$"    ,      "IU")
                , ("M(.+)", "M\\1\\1")
                , ("III"   ,       "U")
                , ("UU"    ,        "") ]

step :: (Rule, Theorem) -> String
step = uncurry (printf " -%s-> %s")

choice :: [a] -> Rand a
choice xs = (xs !!) <$> (state . randomR) (0, length xs - 1)

derive :: Theorem -> Rand [(Rule, Theorem)]
derive t = produceFrom axiom where
  produceFrom s
    | s == t    = return []
    | otherwise = do
      (name, next) <- fmap ($s) <$> choice rules
```

```
      if next /= s
        then (:) (name, next) <$> produceFrom next
        else produceFrom s

main :: IO ()
main =
 let target = "MIIUIIU"
     valid  = (==target) . snd . last . take 20
     show'  = putStrLn . (++"\n") . (axiom ++) . concatMap step
     proof  = evalState (derive target) . mkStdGen
 in mapM_ show' . (nub . filter valid . map proof) $ [1..100]
```

1.5.1.2 number.py

Il programma python utilizzato per calcolare il numero di Gödel delle formule in questo documento

```
from functools import reduce

symbols = ["0","S","=","+","\\times","(",")",
           "\\langle","\\rangle","a","'",
           "\\wedge","\\vee","\\supset",
           "\\sim","\\exists","\\forall",":"]

numbers = dict(zip(symbols, range(10,30)))

g = lambda r: (
  map(lambda s:
    reduce(lambda x, y:
      x.replace(y, str(numbers[y])), sorted(numbers), s.replace(' ','')
    ),r.split('\n')
  )
)

a=r"""
\forall a:\forall a':(a+Sa')=S(a+a')
\forall a':(SSO+Sa')=S(SSO+a')
(SSO+SO)=S(SSO+SO)
\forall a:S(a+O)=a
S(SO+O)=SO
SS(SO+O)=SSO
(SO+SO)=SSO
"""

print(*g(a), sep='\n')
```

1.6 Bibliografia

1. Douglas R. Hofstadter, “Godel, Escher, Bach: An Eternal Golden Braid.” (1979).
2. Maria Rosaria, “Una costruzione del sistema dei numeri reali” (2012).
3. Enzo Ruffaldi, Gian Paolo Terravecchia, Andrea Sani, “Il Pensiero Plurale” (2008).

2 Interessanti applicazioni: il λ-calcolo

Figura 6: Alonzo Church

Il lambda calcolo o λ-calcolo è un **sistema formale** inventato dal matematico e logico americano **Alonzo Church** negli anni '30 che formalizza le computazioni di funzioni.

2.1 Il problema della decisione

Questo sistema formale fu introdotto per risolvere un altro problema proposto da Hilbert nel suo programma: il cosiddetto **Entscheidungsproblem** o problema della decisione. Il problema è il seguente:

Esiste un algoritmo, un metodo completamente meccanico, per verificare se una qualunque stringa di un sistema formale è un teorema in quel sistema?

La risposta fu fornita indipendentemente da Church e **Alan Turing** nel 1936 e fu un deciso **no**. Church ci arrivò proprio tramite questo sistema formale mentre Turing attraverso un modello ideale di macchina che oggi è noto come **macchina di Turing**. Nello stesso anno poi Turing dimostrò che i sistemi da loro inventati erano equivalenti: cioè sono ugualmente potenti perché possono calcolare la stessa classe di funzioni.

Il lambda calcolo e le macchine di Turing rendono formale la nozione intuitiva di *calcolabile* e *decidibile* che sono fondamentali per risolvere il problema.

Figura 7: Alan Turing

2.2 Oggi

I computer moderni sono tuttora basati sul modello delle macchine di Turing e così anche i linguaggi di programmazione **imperativi** basati sul modo in cui si forniscono istruzioni a tali macchine.

2.2.1 Linguaggi funzionali

Esiste tuttavia un'altra classe di linguaggi di programmazione i cosiddetti **linguaggi funzionali** che sono invece basati sul lambda calcolo. A differenza dei programmi imperativi che sono un elenco di istruzioni i programmi funzionali consistono di una singola **espressione** che contiene sia l'algoritmo che il suo input. Quest'espressione, tramite l'applicazione di regole, è **ridotta** più volte fino ad ottenere il risultato del programma. La riduzione consiste nel sostituire una parte dell'espressione con una più semplice come facciamo in matematica. Per esempio:

$$\begin{aligned}(1+2)\times(4\times 3) &\to 3\times(4\times 3)\\ &\to 3\times 12\\ &\to 36\end{aligned}$$

Partendo dall'espressione E otteniamo alla fine l'espressione E^* detta *forma normale*, ovvero il risultato. Il lambda calcolo fa esattamente questo.

2.3 Definizione

Vediamo ora la definizione del sistema formale.

2.3.1 Simboli

I simboli usati sono

1. la lettera λ
2. il punto .
3. le parentesi tonde ()
4. un insieme di lettere per le variabili

2.3.2 Lambda termini

Le espressioni (formule) ben formate del λ-calcolo si chiamano λ-termini.

I seguenti sono λ-termini:

1. **variabili**: si chiamano variabili tutti i simboli come $x, y, z, \cdots$
2. **applicazione**: si dice applicazione $(t\ s)$ dove s e t sono λ-termini.
3. **λ-astrazione** si dice λ-astrazione $\lambda v.t$ dove t è un λ-termine e v una variabile.

Alcuni esempi di termini:

$$x$$

$$y$$

$$(x\ y)$$

$$(\lambda x.(x\ y))$$

Per comodità userò operatori come $\times, +, -$, numeri e alcune metasintassi ma ricordiamoci che essi non fanno parte del sistema formale.

2.3.3 Applicazione

La prima operazione di base del λ-calcolo è l'*applicazione*: Se F è un algoritmo e A dei dati, indichiamo l'applicazione di F al valore A con:

$$F\ A$$

Con applicazione si può intendere sia il risultato dell'algoritmo sia il processo stesso di applicazione. Poiché il lambda calcolo non definisce i **tipi** possiamo anche applicare F a se stesso (per fare una ricorsione per esempio) così:

$$F\ F$$

Un esempio di applicazione è

$$(\lambda x.x + 2)2 \to 4$$

dove abbiamo applicato la λ-espressione $(\lambda x.x + 2)$ al valore 2 e ottenuto 4.

2.3.4 Astrazione

Un'altra operazione è l'*astrazione*: Se $F[x]$ è un espressione che contiene la variabile x: $\lambda x.F[x]$ è un'espressione che denota la funzione $x \mapsto F[x]$.

Quando applichiamo un valore ad un'astrazione facciamo una sostituzione:

$(\lambda x.F[x])A$ è uguale a $F[A]$, scritto anche come:

$$(\lambda x.F[x])D = F[x := D]$$

dove $F[x := D]$ indica il λ-termine ottenuto sostituendo tutte le occorrenze di x con D.

Un'astrazione quindi è la definizione di una funzione che non ha un nome specifico. Infatti vengono a volte chiamate *funzioni anonime*.

2.3.5 Variabili libere e legate

In un'espressione le variabili possono essere legate o libere. Per esempio in $\lambda x.(y\ x)\ y$ è libera mentre x è legata. Le variabili libere possono quindi essere legate da una λ-astrazione con il simbolo λ.

Quando applichiamo un valore facendo una sostituzione $[x := D]$ in un'astrazione questa avviene solo dove x è libera. Esattamente come quando abbiamo una funzione integrale tipo $F(x) = \int_0^x f(x)dx$ e calcoliamo un suo valore come $F(3) = \int_0^3 f(x)dx$.

Per esempio:

$$F[y] = \lambda y.(\lambda y.y\ y) \quad F[y := 3] = (\lambda y.y\ 3)$$

Se un λ-termine non contiene variabili libere si che è un **combinatore** o termine *chiuso*.

2.3.6 Applicazione parziale

Figura 8: Haskell Curry

Abbiamo visto che con la λ-astrazione possiamo produrre funzioni che accettano un parametro. Spesso però abbiamo la necessità di avere funzioni con **argomenti multipli** come $f(x, y, z)$ Si può rappresentare queste funzioni usando un metodo noto come applicazione parziale o *currying*, dal matematico **Haskell Curry** che lo utilizzò per primo.

Una funzione di più argomenti è rappresentata come una catena di funzioni in cui ciascuna genera a sua volta una funzione di un singolo argomento.

Per esempio la funzione $f(x, y, z) = x \times y + z$ viene così definita:

$$\lambda x.\lambda y.\lambda z.(x \times y + z)$$

Se vogliamo calcolare $f(2, 3, 5)$:

1. applichiamo la prima volta 2 e otteniamo la funzione $\lambda y.\lambda z.(2 \times y + z)$
2. applichiamo la seconda volta 3 e otteniamo la funzione $\lambda z.(2 \times 3 + z)$
3. applichiamo infine 5 e otteniamo il risultato $(2 \times 3 + 5) \rightarrow 11$

complessivamente $f(2, 3, 5) = (((\lambda x.\lambda y.\lambda z.(x \times y + z)\ 2)\ 3)\ 5)$

2.4 Regole

Vediamo ora le regole per ridurre o convertire una λ-espressione.

2.4.1 α-conversione

L'α-conversione è una regola che permette di sostituire le variabili legate in un λ-termine, cioè **rinominarle**. I λ-termini ottenuti con questa conversione si dicono tra loro **α-equivalenti**. L'α-conversione è una sostituzione del tipo $F[u := v]$ dove u è una variabile legate e v una variabile qualsiasi.

Per esempio $\lambda x.x \rightarrow_\alpha \lambda z.z$.

Non è sempre possibile effettuare un α-conversione. Per esempio non è possibile fare la sostituzione $(\lambda y.\lambda x.y)[y := x]$. Infatti $\lambda x.\lambda x.x$ è un λ-termine completamente diverso.

2.4.2 β-riduzione

La β-riduzione è un'applicazione del tipo $(\lambda x.M)N$ dove M e N sono dei λ-termini. Si può calcolare come una sostituzione $M[x := N]$.

$(\lambda x.M)N$ si dice β-redex e $M[x := N]$ la sua β-contrazione. La riduzione si indica con $g \rightarrow_\beta g'$ e si legge "g riduce a g'".

Per esempio $(\lambda x.x^3 - 1)3 \rightarrow_\beta 10$

2.4.3 η-conversione

Quando abbiamo un λ-termine del tipo $(\lambda x.M\ x)$ l'η-conversione ci permette di rimuovere la λ-astrazione se x non appare libera in M. I termini prodotti si dicono η-equivalenti e la conversione si indica con $g \rightarrow_\eta g'$.

Per esempio $\lambda y.\lambda x.y\ x \rightarrow_\eta \lambda y.y$

2.4.4 In 2 righe

Utilizzando la metasintassi di Backus-Naur possiamo riassumere la sintassi e la semantica del λ-calcolo in appena 2 righe:

$$L ::= v \mid \lambda v.L \mid (L\ L)$$
$$(\lambda v.L_\beta)L_\alpha \rightarrow L_\beta[v := L_\alpha]$$

2.5 Utilità

Vediamo ora che abbiamo definito il sistema cosa ci permette di fare.

2.5.1 Aritmetica

Il λ-calcolo calcolo contiene l'aritmetica di Peano. È possibile rappresentare i numeri naturali tramite i **numerali di Church** così definiti:

0. $\lambda y.\lambda x.x$
1. $\lambda y.\lambda x.y\ x$
2. $\lambda y.\lambda x.y\ (y\ x)$
3. $\lambda y.\lambda x.y\ (y\ (y\ x))$

   ...

I numerali di Church sono **funzioni di ordine maggiore**: una funzione che accetta una funzione come parametro e ne genera un'altra. In particolare l'n-esimo numerale prende una funzione y e la applica n volte a se stessa.

La funzione successore è definita come:

$$\text{succ} := \lambda n.\lambda y.\lambda x.y(n\ y\ x)$$

`succ` semplicemente prende come argomento un numerale n a cui applica nuovamente la funzione y per ottenere $n + 1$.

È altrettanto semplice definire le operazioni aritmetiche. Per esempio:

$$\begin{aligned}\text{mult} &:= \lambda m.\lambda n.m\ \text{succ}\ n\\ \text{plus} &:= \lambda m.\lambda n.\lambda y.m\ (n\ y)\end{aligned}$$

2.5.2 Logica booleana

I valori di vero e falso sono rappresentati per convenzione tramite i **booleani di Church**:

$$\begin{aligned}T &:= \lambda x.\lambda y.x \quad \text{(vero)}\\ F &:= \lambda x.\lambda y.y \quad \text{(falso)}\end{aligned}$$

È poi possibile formulare gli operatori logici così:

$$\begin{aligned}\text{and} &:= \lambda p.\lambda q.p\ q\ p\\ \text{or} &:= \lambda p.\lambda q.p\ p\ q\\ \text{not} &:= \lambda p.\lambda a.\lambda b.p\ b\ a\end{aligned}$$

Un esempio di uso:

$$\begin{aligned}\text{and } T\ F &\to (\lambda p.\lambda q.p\ q\ p)\ T\ F\\ &\to (T\ F)\ T\\ &\to (\lambda x.\lambda y.x\ F)\ T\\ &\to \lambda x.\lambda y.y\\ &\to F\end{aligned}$$

2.5.3 Ricorsione anonima

Una funzione ricorsiva essenzialmente è una funzione che nella sua definizione contiene se stessa. L'esempio più semplice è la funzione fattoriale $n! = 1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n$ che può essere così definita:

$$n! = \begin{cases} 1 & x = 0 \\ n \cdot (n-1)! & \text{altrimenti} \end{cases}$$

È fondamentale quindi che la funzione abbia un **nome** associato. Nel λ-calcolo nonostante le astrazioni siano anonime e quindi non possono riferirsi tramite un nome è possibile ottenere lo stesso risultato. Per fare ciò usiamo il **combinatore** Y

Il combinatore Y, o *di punto fisso*, è una qualunque funzione di ordine maggiore che soddisfa la relazione $y\ f = f\ (y\ f)$ per qualunque f. Per esempio il **combinatore di Curry** è di punto fisso:

$$Y := \lambda f.(\lambda x.f(x\ x))\ (\lambda x.f\ (xx))$$

Possiamo dimostrare che vale la relazione del combinatore Y tramite riduzione:

$$\begin{aligned} Y\ g = \lambda f.(\lambda x.f(x\ x))\ (\lambda x.f\ (x\ x))\ g \quad & \text{per definizione}\\ \to_\beta (\lambda x.g(x\ x))\ (\lambda x.g\ (x\ x)) \quad & \text{applicazione di } Y \text{ a } g\\ \to_\beta g(\lambda x.g(x\ x))\ (\lambda x.g\ (x\ x)) \quad & \text{applicazione del primo termine}\\ = g\ (Y\ g) \quad & \text{uguaglianza al secondo passaggio}\end{aligned}$$

Continuando ad applicare l'uguaglianza si ottiene:

$$g\ (Y\ g) = g\ (g\ (Y\ g)) = g\ (g\ (g\ (Y\ g))) = g\ (\ldots (Y\ g) \ldots)$$

Quindi il combinatore Y continua ad applicare la funzione argomento e questo ci permette di scrivere una funzione ricorsiva tramite una λ-astrazione senza la necessità di darle un nome.

Torniamo al nostro fattoriale: scriviamo subito la definizione di fattoriale nel modo classico:

$$\text{fact} := \lambda n.(\text{if } n = 0 \text{ then } 1 \text{ else } n \times \text{fact } (n-1))$$

invece di chiamare fact usiamo una funzione f tramite la λ-astrazione $\lambda f.$:

$$\lambda f.\lambda n.(\text{if } n = 0 \text{ then } 1 \text{ else } n \times f\ (n-1))$$

infine applichiamo Y all'espressione per ottenere:

$$\text{fact} := Y\ (\lambda f.\lambda n.(\text{if } n = 0 \text{ then } 1 \text{ else } n \times f\ (n-1)))$$

In modo più formale usiamo i combinatori standard del lambda calcolo:

$$\text{fact} := Y\ (\lambda f.\lambda n.(\text{isZero } n)\ 1\ (\text{mult } n\ (f\ (\text{pred } n))))$$

2.6 Turing-equivalenza

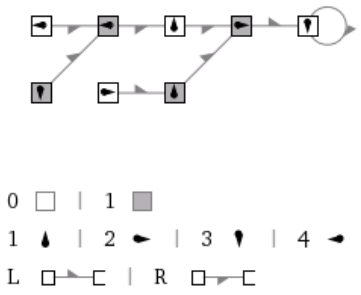


Figura 9: schema della macchina busy beaver 4,2

Con la capacità di esprimere l'aritmetica, la logica ed infine anche la ricorsione il λ-calcolo è diventato **Turing-equivalente**: può compiere qualunque operazione una macchina di Turing sia in grado di svolgere. Secondo la **tesi di Church-Turing** cioè può calcolare qualsiasi funzione intuitivamente calcolabile. È sicuramente sorprendente per un sistema che si può descrivere in solo due righe. Per accorgersi di quanto è semplice e potente basta fare un confronto con TNT che per poter esprimere ogni proprietà dei numeri necessita di ben 7 regole di inferenza, 5 assiomi ed ingloba un intero sistema formale a parte (il calcolo proposizionale).

2.7 λ-calcolo tipizzato

La versione che abbiamo visto del λ-calcolo è senza tipi: cioè **non ci sono limitazioni** su cosa si può applicare a qualunque espressione. I termini possono persino essere applicati a se stessi. Questo permette una grande capacità di espressione ma porta agli stessi problemi individuati da Russell nella teoria *naive* degli insiemi. Per esempio il combinatore Y che abbiamo utilizzato per la ricorsione può essere utilizzato per produrre un paradosso (**paradosso di Curry**) che permette di derivare qualsiasi λ-termine. Lo stesso paradosso di Russel è presente in questa formulazione.

Questo si può risolvere introducendo una **teoria dei tipi** nel λ-calcolo, esattamento quello che ha fatto Russel in *Principia mathematica*.

2.7.1 Tipi

Diciamo che $\mathcal{G}$ sia l'insieme dei tipi primitivi. Sono tipi:

- ogni $A \in \mathcal{G}$
- $A \to B$ dove A e B sono tipi

2.7.2 Pre-termini

Modifichiamo la definizione precedente di termine per inserire i tipi:

I seguenti sono pre-termini:

1. **variabili**: si chiamano variabili tutti i simboli come $x, y, z, \cdots$
2. **applicazione**: si dice applicazione $(t\ s)$ dove s e t sono pre-termini.
3. **λ-astrazione** si dice λ-astrazione $\lambda v^A.t$ dove t è un pre-termine, v una variabile e A è un tipo.

2.7.3 Test

Per confermare che un pre-termine sia un termine dobbiamo verificare che tutte le variabili presenti in esso abbiano i tipi corretti. Un test ha la seguente forma:

$$\underbrace{x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n}_{\text{contesto}} \vdash \underbrace{M : A}_{\text{termine}}$$

dove:

- $x_1, x_2, \ldots, x_n$ sono variabili presenti in M

- $A_1, A_2, \ldots, A_n$ sono tipi
- M è un pre-termine

Se il tipo di ogni variabile in M è confermato allora il test è passato e M è un λ-termine. Il processo con cui si determina il tipo delle variabili nel termine e se il tipo A esiste si chiama **type inference** e l'insieme di regole che definiscono i tipi e come si assegnano ai termini si chiama **type system**.

L'assegnazione di tipi ai termini potrebbe sembrare un inutile complicazione e una limitazione ma rende impossibile la produzione di espressioni prive di significato e spesso evita di fare errori durante l'applicazione e la riduzione ma la cosa più importante è che è possibile fare una formulazione del lambda calcolo in non esistono problemi indecidibili.

2.7.4 Haskell

Figura 10: Il logo del linguaggio

Haskell, che prende il nome da Haskell Curry, è un **linguaggio** di programmazione puramente funzionale staticamente tipizzato che si basa sulla **teoria delle categorie** e proprio sul **λ-calcolo tipizzato**. Il programma stesso utilizzato per generare questo documento è scritto interamente in haskell così come i vari programmi da me scritti allegati alla tesi.

Haskell usa un particolare type system noto come **Hindley–Milner** usato anche da altri linguaggi funzionali. HM permette di dedurre il tipo di ogni espressione anche in assenza di qualunque annotazione dei tipi e lo fa in tempo quasi-lineare rispetto alla dimensione del programma. Consente inoltre tipi polimorfici e funzioni di ordine maggiore.

Un programma in haskell è un'espressione costituita da una o più definizioni di funzioni e costanti. Essendo puramente funzionale le funzioni sono **funzioni matematiche**: un'associazione di un valore ad un altro valore. Non è possibile che fornendo lo stesso dato la funzioni produca risultati diversi nel tempo, né che la sua esecuzione possa modificare lo stato del programma o di altre funzioni (**side effects**). Non esistono stati variabili: tutte le strutture dati sono **immutabili**. Se è necessario fare delle modifiche semplicemente se ne producono di nuove.

Questa descrizione dà un idea di un linguaggio fortemente limitato ma in realtà le sue caratteristiche funzionali forniscono degli enormi vantaggi:

- Non essendoci possibili stati mutabili o side effects non possono esistere problemi dovuti all'accesso **concorrente** alla memoria. L'intero programma è *thread-safe*.
- La gestione della memoria è basata sul *garbage collecting* ma a differenza di altri linguaggi l'immutabilità delle strutture rende il processo di cancellazione dei dati deferenziati estremamente semplice e rapido.
- I risultati dell'applicazione di funzioni vengono calcolati solo quando necessario *lazy evaluation*. Questo permette di creare strutture dati **infinite** Per esempio `fib = 0 : scanl (+) 1 fib` è lista infinita di *tutti* i numeri di Fibonacci.
- Anche quando non si vuole ragionare sui tipi il compilatore è in grado dedurli da sé e impedisce di fare errori in ogni caso. Con il type system di haskell è impossibile produrre un programma che compili ma che produca degli errori in *runtime* se non volutamente.
- Per l'**isomorfismo di Curry-Howard** un programma haskell *type checked* è analogo ad una dimostrazione formale. Il type system può essere perciò usato per controllare la validità di una dimostrazione matematica.
- Lo stile funzionale e il *pattern matching* sono molto espressivi e solitamente un programma è molto più conciso della controparte imperativa.

2.7.5 Esempi di programma

2.7.5.1 Game of Life

Un'implementazione del Game of Life di John Conway su una griglia rettangolare

```
import Control.Applicative
import Data.List  ((\\))
import Data.Maybe (fromMaybe)
import Matrix

type Cell = Int
type Grid = Mat Cell
```

```haskell
(!) :: Grid -> Pos -> Cell
(Mat g) ! (x, y) = fromMaybe 0 (g ?? y >>= (?? x))

near :: Grid -> Pos -> [Cell]
near g (x, y) = map state $ neighbours \\ [(0,0)]
  where
    neighbours = liftA2 (,) [-1..1] [-1..1]
    state (x', y') = g ! (x+x', y+y')

alive :: Grid -> Pos -> Cell
alive g p
  | v == 0 &&  n == 3            = 1
  | v == 1 && (n == 2 || n == 3) = 1
  | otherwise                    = 0
  where (n, v) = (sum (near g p), g ! p)

next :: Grid -> Grid
next g = alive g <$> indeces g

main :: IO ()
main = mapM_ print (iterate next grid)

grid = Mat
  [ [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  , [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  , [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  , [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  , [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  , [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  , [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] ]
```

2.7.5.2 RPN

Un programma che risolve espressioni in notazione polacca inversa

Per esempio:

```
> 5 8 9 2 * cos ^ - 2 0.452 tan / +
= 5.1717299516
```

```haskell
{-# LANGUAGE ViewPatterns #-}

import Data.List
import Data.Maybe
import Text.Read
```

```haskell
import Text.Printf
import Control.Monad
import System.Console.Haskeline

main :: IO ()
main = runInputT defaultSettings repl

repl :: InputT IO ()
repl = do
  line <- getInputLine "> "
  case fromMaybe "" line of
    "q" -> return ()
    ""  -> outputStrLn "" >> repl
    exp -> outputStrLn (result (rpn exp) ++ "\n") >> repl

result :: Either String Double -> String
result (Left err) = "!! " ++ err
result (Right x)  = printf format x where
  format | ceiling x == floor x = "= %.0f"
         | otherwise            = "= %.10f"

rpn :: String -> Either String Double
rpn = foldM parse [] . words >=> return . head where
  parse (y:x:xs) (flip lookup dyad  -> Just f) = Right (f x y : xs)
  parse (x:xs)   (flip lookup monad -> Just f) = Right (f x : xs)
  parse xs       (flip lookup nilad -> Just k) = Right (k : xs)
  parse xs       (readMaybe -> Just x) = Right (x : xs)
  parse _        _                     = Left "syntax error"

dyad = [ ("+", (+))
       , ("-", (-))
       , ("*", (*))
       , ("/", (/))
       , ("^", (**)) ]

monad = [ ("sin"  , sin )
        , ("asin" , asin)
        , ("cos"  , cos )
        , ("acos" , acos)
        , ("tan"  , tan )
        , ("atan" , atan)
        , ("ln"   , log )
        , ("sqrt" , sqrt)
        , ("sgn"  , signum)
        , ("abs"  , abs)
        , ("floor", fromIntegral . floor)
```

```
        , ("ceil" , fromIntegral . ceiling) ]

nilad = [ ("pi" , pi)
        , ("e"  , exp 1)
        , ("phi", (1 + sqrt 5)/2) ]
```

2.8 Bibliografia

1. Henk Barendregt, Erik Barendsen, “Introduction to Lambda Calculus” (1998).
2. Andrzej S. Murawski, “Typed lambda calculi” (2011).
3. Enrico Denti, “Introduzione ai linguaggi fondazionali” (2012).
4. Miran Lipovača, “Learn You a Haskell for Great Good” (2011).

3 Un'analogia: Pirandello e Metateatro

3.1 Introduzione

Uno dei metodi principali utilizzati da Gödel per dare la dimostrazione dei teoremi di incompletezza è quello produrre all'interno di una teoria matematica delle formule che parlino della teoria stessa, il cosiddetto numero di Gödel. Il fatto stesso che le teorie non possono mai essere complete deriva proprio dal fatto che possono **contenere se stesse**. Si può così dire che i sistemi formali nonostante imitino e si avvicinino per quanto si vuole alla realtà non sono mai in grado di raggiungerla completamente: esisteranno sempre delle verità che non possono essere da essi rappresentate. Troviamo questo stesso tema anche in letteratura e precisamente nell'opera di **Luigi Pirandello** (1867-1936) contemporaneo di Gödel.

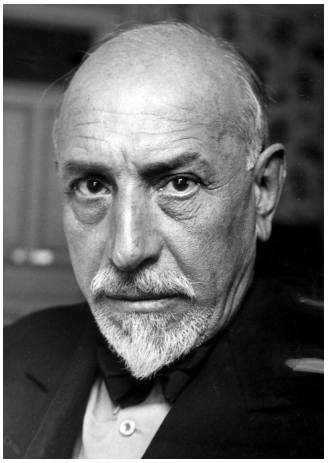


Figura 11: Luigi Pirandello

Nella sua opera mette in discussione l'immagine di un mondo ordinato, organico e interpretabile con certezza mediante gli strumenti razionali della ragione. Il risultato è un forte **relativismo**: la realtà è tanto complessa da non poter essere rappresentabile da un unico punto di vista né fissata in un unico sistema di regole come voleva Hilbert perché non esiste una sola verità ma ciascuno ne possiede una propria.

3.2 Sei personaggi in cerca d'autore

Esemplificativo di questo modo di intendere il mondo è il dramma "Sei personaggi in cerca d'autore". Si tratta di un esempio di **metateatro**: o *teatro nel teatro*. Cioè un opera in cui arte e teatro mettono in scena se sessi. Si tratta di un opera

buia, cupa ma di grande impatto intellettuale ed emotivo. Insieme a "Ciascuno a suo modo" (1924) e "Questa sera si recita a soggetto" (1929), "Sei personaggi in cerca d'autore" (1921) è il primo di una trilogia di drammi in cui l'autore affronta la tipica situazione metateatrale in cui gli attori/personaggi/autori utilizzano lo spazio del teatro per mettere in scena un'altra rappresentazione: come in una scatola cinese.

Figura 12: I sei personaggi

La trama è piuttosto complessa, il palco è spoglio perché in attesa di allestimento per la messa in scena della commedia "Il gioco delle parti", sempre di Pirandello, affidata alla *Compagnia degli Attori*. Nel corso della prima parte dello spettacolo avviene quello che si definisce *sfondamento della quarta parete*. Gli attori infatti non si trovano sul palco ma in mezzo alla platea, tra il pubblico. Dal fondo della sala, e non dalle quinte, fanno il loro ingresso *La Prima Attrice* e gli stessi *Sei personaggi*: *Il Padre, La Madre, Il Figlio, La Figliastra, La Bambina e Il Giovinetto*. Essi in realtà non sono altro che ombre, idee di un fantomatico *Autore* che, dopo averli creati, li ha abbandonati rinunciando a scriverne il romanzo o la commedia. Bloccati in questa specie di limbo i sei personaggi chiedono aiuto alla *Compagnia degli Attori* perché mettano in scena il loro dramma affinché possano raggiungere la completezza della loro creazione. La *Compagnia* accetta ma il risultato è deludente. La natura della personalità di ciascun personaggio è tanto complessa, così ricche di sfaccettature sono le loro emozioni, che il lavoro degli attori risulta finto e forzato.

3.2.1 Tema

Alla fine dello spettacolo sono i personaggi stessi ad autorappresentarsi trasformando anche gli attori in pubblico in un continuo gioco di specchi. Pirandello mette l'accento sull'impossibilità di comporre il conflitto tra vita reale e finzione

scenica e in sintesi sull'incapacità intrinseca del teatro di rendere sulla scena ciò che l'autore ha ideato.

3.2.2 Critica

Presentato per la prima volta a Roma nel 1921 "Sei personaggi in cerca d'autore" fu dapprima contestato ferocemente dal pubblico al grido di "Manicomio, manicomio!", impreparato ad un discorso di avanguardia che di fatto distruggeva le forme classiche del teatro convenzionale. Successivamente però il colossale fiasco si trasformò in un grande successo anche su scala mondiale.