Indecidibilità della logica del primo ordine

Silvio Valentini
Dipartimento di Matematica Pura ed Applicata
Università di Padova
via G. Belzoni n.7, I–35131 Padova, Italy
silvio@math.unipd.it

March 25, 2003

1 Premessa

Abbiamo visto che, dato un insieme finito Σ di formule nel linguaggio proposizionale, possiamo effettivamente decidere in modo effettivo e in un numero finito di passi, che possiamo addirittura calcolare a partire dall'insieme di formule considerato, se Σ é soddisfacibile o meno. La cosa non ci é riuscita altrettanto bene quando abbiamo considerato un insieme di formule del linguaggio predicativo. Infatti, in questo caso il nostro metodo di ricerca degli alberi predicativi riesce a fornirci una risposta, vale a dire "Σ non é soddisfacibile" quando Σ é davvero non soddisfacibile, ma in generale si limita a continuare il tentativo di chiudere l'albero di confutazione nel caso in cui Σ sia soddisfacibile.

É chiaro che rivolgerci direttamente alla semantica non può fare altro che peggiorare la situazione visto che in questo caso dovremo andare ad analizzare tutti i possibili modelli nella speranza di trovarne uno che soddisfi tutte le formule in Σ ed tentare di vedere se un modello soddisfa una particolare formula in Σ é in generale già un compito superiore alle nostre forze tutte le volte che abbiamo a che fare con una struttura infinita e che tale formula richiede di verificare una formula quantificata universalmente.

Tuttavia permane naturalmente il dubbio se siamo stati noi incapaci di inventare un metodo buono abbastanza da permetterci di decidere sul problema della soddisfacibilità di ogni insieme di formule o se davvero la situazione é "disperata", vale a dire che non c'é nessun metodo che possa risolvere tale problema.

É chiaro che non esiste una risposta definitiva a questa questione, almeno se la poniamo in questi termini generali. Infatti, la risposta dipende chiaramente da quali sono i metodi che consideriamo utilizzabili: nella mente di dio la soddisfacibilità di ogni insieme di formule del linguaggio predicativo é sicuramente decidibile ma possiamo pensare di disporre degli stessi strumenti di decisione che dio può utilizzare?

2 Introduzione

Un *problema di decisione* per una certa proprietà P sugli elementi di un dato insieme S é *risolvibile* se esiste un test *effettivo* che applicato ad un qualsiasi elemento di S alla fine stabilisce se tale elemento soddisfa oppure no la proprietà P. Un test *positivo* per una proprietà P é un test che riesce a scoprire tutti gli elementi di S che soddisfano la proprietà P mentre un test *negativo* é un test che riesce a scroprire tutti gli elementi che non soddisfano la proprietà P. Quindi un certo problema di decisione é risolvibile se e solo se esistono sia un test positivo che un test negativo.

Ad esempio il metodo degli alberi proposizionali é un test sia negativo che positivo per scoprire se un certo insieme finito di formule é soddisfacibile. Infatti, in un numero finito di passi ci fornisce o un albero di confutazione per l'insieme considerato o una valutazione che ne rende vere tutte le formule. D'altra parte, il metodo degli alberi predicativi é un test negativo per scoprire se un certo insieme di formule del linguaggio del primo ordine é soddisfacibile: infatti si tratta di un metodo che ci fornisce un albero di confutazione se l'insieme considerato é non soddisfacibile ma non ci garantisce alcuna risposta se esso é soddisfacibile. Naturalmente lo stesso test può essere utilizzato anche per scoprire se una certa formula α é conseguenza logica di un certo insieme di formule Σ, visto che sappiamo comunque che $\Sigma \vdash \alpha$ se e solo se l'insieme di formule $\Sigma \cup \{\neg\alpha\}$ é non soddisfacibile. Siamo quindi in possesso di un test positivo per decidere il problema delle conseguenze logiche. Il resto di questo testo é dedicato alla dimostrazione che, una volta fissata una ragionevole nozione di *test effettivo*, non esiste nessun test negativo per il problema della decisione della conseguenza logica.

3 Decidere la logica predicativa

La struttura della nostra dimostrazione di impossibilità della decisione della logica predicativa si articolerà in una sequenza di passi: prima di tutto definiremo cosa intendiamo con test effettivo presentando delle macchine che possiamo utilizzare per effettuare dei test, poi mostreremo che esistono problemi che con tali macchine non si possono risolvere, infine dimostreremo che se si potesse risolvere il problema della decisione della consequenza logica per la logica predicativa allora si potrebbe risolvere anche uno di tali problemi irresolvibili.

3.1 Macchine di Turing

Le macchine di Turing costituiscono un modello meccanico di un processo di calcolo. Per capire la loro importanza bisogna pensare che esse sono state introdotte ben prima che esistessero gli attuali calcolatori. Lo scopo del loro inventore, Alan Turing, non era tanto quello di creare un calcolatore efficente quanto piuttosto quello di determinare che cosa significa che una certa funzione é calcolabile. Il problema di Turing era quindi da un lato quello di essere si-

curo che le operazioni previste dalla macchina fossero sicuramente effettive e dall'altro che con tale macchina fosse possibile calcolare tutto quello che viene considerato ragionevolmente calcolabile.

Una macchina di Turing può fare solo alcune operazioni elementari: scrivere, leggere e, a seconda di quello che legge, decidere quale operazione fare, scegliendola tra un numero finito di possibilità.

L'idea di Turing era che qualsiasi calcolo si pensi di poter effettuare in modo effettivo possa essere ridotto ad una (eventualmente lunga) sequenza di queste operazioni elementari.

Sopponiamo allora di avere a disposizione un numero non limitato di foglietti di carta che possiamo immaginare disposti lungo un nastro con l'ipotesi che qualora ci serva un nuovo foglio possiamo sempre procurarcelo (non possiamo infatti limitare a priori quanti fogli saranno necessari perchè in questo caso é facile pensare ad una situazione che richieda almeno un foglio in più).

Possiamo pensare che all'inizio della computazione ogni foglio sia bianco eccetto eventualmente che per un numero finito di fogli sui quali é scritto uno tra gli n simboli $S_1, S_2, \ldots, S_n$. Per uniformità, fa comodo pensare che i foglio bianchi contengano in realtà anche loro un simbolo, il *bianco*, che possiamo indicare con S_0.

Abbiamo poi la necessità di leggere e scrivere sui foglietti che abbiamo preparato. Supponiamo allora che le operazioni ammesse per la macchina siano quelle di leggere e scrivere sul foglietto che stiamo osservardo e di spostarsi sul foglietto alla sua destra e alla sua sinistra nel nastro dei foglietti (eventualmente aggiungendo un foglietto se fosse necessario).

Ci vuole naturalmente anche un *programma* che istruisca la macchina sul da farsi a seconda del contenuto del foglio che sta leggendo. A questo scopo supporremo che in qualsiasi istante la macchina si trovi in un qualche *stato* particolare q_i scelto tra un numero finito m di stati $q_1, \ldots, q_m$ possibili. Quindi se la macchina si trova nello stato q_i allora si esegue l'istruzione del programma che dirà alla macchina cosa fare in quello stato a seconda del simbolo letto nel foglietto che la macchina sta guardando. Le possibili scelte sono

- fermarsi;
- spostarsi sul foglietto a destra
- spostarsi sul foglietto a sinistra
- scrivere uno dei simboli $S_0, S_1, \ldots, S_n$ sul foglietto in esame

Dopo aver scelto cosa fare bisognerà anche decidere quale sia la prossima istruzione da eseguire. Quindi in generale l'istruzione i-ma avrà il seguente formato

$$q_i \quad S_k \quad \begin{matrix} R \\ L \\ S_m \end{matrix} \quad q_j$$

che dice alla macchina: "se sei nello stato q_i e sul foglietto che stai guardando leggi il simbolo S_k allora vai a destra (R) o vai a sinistra (L) o scrivi S_m sul foglietto dove sei e ti prepari ad eseguire l'istruzione q_j".

Come avrete notato non esiste una esplicita istruzione di arresto per la nostra macchina; possiamo tuttavia convenire che la macchina si fermi quando non sa cosa fare, quando cioè entra in uno stato in cui non esiste alcuna istruzione relativa al simbolo letto.

Esempio

Supponiamo di far partire la nostra macchina su un nastro tutto bianco con il seguente programma

$$\begin{array}{cccc} q_1 & S_0 & S_1 & q_1 \\ q_1 & S_1 & R & q_2 \\ q_2 & S_0 & S_1 & q_2 \\ q_2 & S_1 & R & q_3 \\ q_3 & S_0 & S_1 & q_3 \end{array}$$

e proviamo ad eseguire il programma. Per capire cosa succede simuliamo il comportamento della macchina mostrando come viene modificato il nastro e in quale stato si trova la macchina in ogni momento (scriveremo lo stato della macchina sotto al foglietto letto).

$$\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & & & \end{array}$$

$$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & & & \end{array}$$

$$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ & 2 & & \end{array}$$

$$\begin{array}{cccc} 1 & 1 & 0 & 0 \\ & 2 & & \end{array}$$

$$\begin{array}{cccc} 1 & 1 & 0 & 0 \\ & & 3 & \end{array}$$

$$\begin{array}{cccc} 1 & 1 & 1 & 0 \\ & & 3 & \end{array}$$

La macchina scrive quindi tre S_1 sul nastro e poi si ferma.

Vediamo ora un esempio un po' più complicato: vogliamo costruire una macchina che duplica il numero di S_1 che trova sul nastro nel momento in cui parte.

Possiamo costruire tale programma scrivendo due volte S_1 sul lato sinistro del nastro per ogni S_1 che leggo (e poi cancello) sul lato destro del nastro. Ecco

quindi il programma adatto.

q_1	S_1	L	q_2
q_2	S_0	L	q_3
q_2	S_1	L	q_3
q_3	S_0	S_1	q_3
q_3	S_1	L	q_4
q_4	S_0	S_1	q_4
q_4	S_1	R	q_5
q_5	S_0	R	q_6
q_5	S_1	R	q_5
q_6	S_0	L	q_7
q_6	S_1	R	q_6
q_7	S_0	L	q_8
q_7	S_1	S_0	q_7
q_8	S_0	L	q_{11}
q_8	S_1	L	q_9
q_9	S_0	L	q_{10}
q_9	S_1	L	q_9
q_{10}	S_0	R	q_2
q_{10}	S_1	L	q_{10}
q_{11}	S_0	R	q_{12}
q_{11}	S_1	L	q_{11}

Per vedere come funziona questo programma possiamo provare a simularne il comportamento su un nastro che all'inizio é completamente bianco a parte due soli foglietti che contengono S_1. Tanto per fissare le idee supponiamo che nel momento iniziale la macchina stia guardando il primo foglietto a sinistra che

contiene S_1.

0	0	0	0	0	0	1 1	1	0
0	0	0	0	0	0 2	1	1	0
0	0	0	0	0 3	0	1	1	0
0	0	0	0	1 3	0	1	1	0
0	0	0	0 4	1	0	1	1	0
0	0	0	1 4	1	0	1	1	0
0	0	0	1	1 5	0	1	1	0
0	0	0	1	1	0 5	1	1	0
0	0	0	1	1	0	1 6	1	0
0	0	0	1	1	0	1	1 6	0
0	0	0	1	1	0	1	1	0 6
0	0	0	1	1	0	1	1 7	0
0	0	0	1	1	0	1	0 7	0
0	0	0	1	1	0	1 8	0	0
0	0	0	1	1	0 9	1	0	0
0	0	0	1	1 10	0	1	0	0
0	0	0	1 10	1	0	1	0	0
0	0	0 10	1	1	0	1	0	0
0	0	0	1 2	1	0	1	0	0
0	0	0 3	1	1	0	1	0	0
0	0	1 3	1	1	0	1	0	0
0	0 4	1	1	1	0	1	0	0
0	1 4	1	1	1	0	1	0	0
0	1	1 5	1	1	0	1	0	0
0	1	1	1 5	1	0	1	0	0
0	1	1	1	1 5	0	1	0	0
0	1	1	1	1	0 5	1	0	0
0	1	1	1	1	0	1 6	0	0
0	1	1	1	1	0	1	0 6	0
0	1	1	1	1	0	1 7	0	0
0	1	1	1	1	0	0 7	0	0
0	1	1	1	1	0 8	0	0	0
0	1	1	1	1 11	0	0	0	0
0	1	1	1 11	1	0	0	0	0
0	1	1 11	1	1	0	0	0	0
0	1 11	1	1	1	0	0	0	0
0 11	1	1	1	1	0	0	0	0
0	1 12	1	1	1	0	0	0	0

Decisamente molto complesso per un programma il cui unico scopo é duplicare un numero! Vale la pena di notare che quello che abbiamo scritto é un vero e proprio programma che si può applicare ad ogni sequenza di S_1.

3.1.1 Esercizi

1. Costruire il programma per una macchina di Turing che, partendo dalla posizione più a sinistra di una stringa ininterrotta di S_1 in un nastro altrove bianco, si ferma con un nastro completamente bianco se il numero di S_1 della stringa di partenza é pari e si ferma nell'unica casella con scritto S_1 altrimenti.

2. Costruire il programma per una macchina di Turing tale che partendo dall'estremità sinistra di una stringa ininterrotta di S_1 e S_2 si ferma con un nastro completamente vuoto se e solo se ci sono tanti S_1 quanti S_2 nella stringa di partenza.

3. Supponiamo di avere un nastro completamente bianco eccetto per due sequenze ininterrotte di S_1 di lunghezza rispettivamente p e q separate da uno spazio bianco. Trovare un programma per una macchina di Turing che, partendo dalla posizione più a sinistra della stringa più a sinistra, si fermi nella posizione più a sinistra di una stringa ininterrotta di S_1 lunga $p + q$ in un nastro altrove bianco.

4. Costruire una macchina che non si arresta mai qualsiasi sia la configurazione del nastro su cui viene fatta partire.

5. Costruire una macchina di Turing che si arresta se e solo se quando parte sul nastro c'é un unico foglietto contenente S_1 mentre tutti gli altri foglietti contengono S_0.

3.2 Il problema dell'arresto

Abbiamo già visto un esempio di macchina di Turing capace di duplicare il numero di S_1 che trova sul nastro. Possiamo generalizzare questa idea e pensare di utilizzare le macchine di Turing per calcolare funzioni sui numeri naturali. A questo fine é utile fissare alcune convenzioni. Supponiamo di voler calcolare una funzione ad n argomenti, $d_1, \ldots, d_n$. Allora assumeremo che i vari argomenti siano rappresentati sul nastro da sequenze ininterrotte di S_1 di lunghezza rispettivamente $d_1, \ldots, d_n$ separate da un carattere bianco e supporremo inoltre che la macchina parta guardando il foglietto che contiene il simbolo S_1 più a sinistra del primo dato d_1. Diremo quindi che la macchina ha ottenuto un qualche risultato se e solo se essa si fermerà guardando il più a sinistra di una sequenza ininterrotta di S_1 che corrisponderà al risultato del calcolo della funzione.

Nel seguito diremo che una funzione é *Turing computabile* se esiste una programma per una macchina di Turing che la calcola.

Non é difficile vedere che é possibile costruire una lista di tutte le funzioni *Turing computabili* numerando tutte le macchine di Turing che le calcolano.

Infatti, ogni programma si può rappresentare scrivendo le quartine che lo costituiscono in una unica lunga stringa senza spazi bianchi tra una quartina e l'altra cominciano dalla quartina il cui primo simbolo é lo stato iniziale della macchina. In questo modo ogni macchina di Turing é contraddistinta da un blocco finito di lettere di un alfabeto infinito ma numerabile: R, L, S_0, q_1, S_1, q_2, S_2, ... Naturalmente non ogni "parola" che si possa scrivere in questo "alfabeto" indica una macchina di Turing, ma soltanto quelle che soddisfano le seguenti condizioni:

1. la lunghezza della parola deve essere divisibile per quattro;
2. solo i simboli q_1, q_2, ... compaiono nelle posizioni 1, 4, 5, 8, ..., $4n$, $4n+1$, ...;
3. solo i simboli S_0, S_1, ... compaiono nelle posizioni 2, 6, 10, ..., $4n+2$, ...;
4. solo i simboli R, L, S_0, S_1, ... compaiono nelle posizioni 3, 7, 11, ..., $4n+3$, ...;
5. nessuna configurazione del tipo $q_i S_J$ compare più di una volta nella parola (tanto per essere sicuri di evitare di dare istruzioni contraddittorie)

Dato che l'insieme delle parole ottenute da un alfabeto numerabile é numerabile, possiamo immaginare di elencarle in una lista w_1, w_2, w_3, ... Da questa lista possiamo poi togliere tutte le parole che non indicano una macchina di Turing e ottenere quindi una nuova lista M_1, M_2, M_3, ... dove sono elencate tutte le macchine di Turing. Visto che ogni macchina di Turing determina una ben precisa funzione sui numeri naturali questa lista é anche la lista delle funzioni *Turing computabili*.

Quindi le funzioni *Turing computabili* sono una quantità numerabile ed esistono quindi sicuramente funzioni sui numeri naturali che non sono *Turing computabili*. Questa non sembra essere una grande scoperta visto che a priori non sembra ragionevole aspettarsi che le semplici macchine di Turing che abbiamo introdotto possano poi calcolare molte funzioni visto che in fondo le operazioni che sono permesse ad una macchna di Turing sono un (piccolo) sottoinsieme di quelle permesse ad un normale calcolatore! Tuttavia le macchine di Turing non soffrono delle usuali limitazioni di un calcolatore, visto che abbiamo immaginato di poter sempre aggiungere foglietti (=memoria) quando ne dovessimo avere bisogno e visto che non ci siamo mai preoccupati di quanto a lungo dobbiamo aspettare per ottenere una risposta. Questo é il motivo per cui si può dimostrare che in pratica al momento non esiste nessuna maniera più generale di specificare una funzione sui naturali che si possa calcolare effettivamente e che non sia già calcolabile da parte di una macchina di Turing.

La questione di trovare funzioni che non sia *Turing computabile* diventa quindi più interessante e ancora più interessante diventa la questione di trovare funzioni che siano effettivamente calcolabili e non siano Turing computabili.

Ad esempio la seguente funzione non é Turing computabile.

$$u(n) \equiv \begin{cases} 1 & \text{se } f_n(n) \text{ non é definita} \\ f_n(n)+1 & \text{altrimenti} \end{cases}$$

dove con f_n intendiamo indicare la funzione di posto n nella lista delle funzioni Turing computabili. Vediamo allora che la funzione $u(-)$ non é Turing computabile. Supponiamo che lo sia. Allora dovrebbe occupare un qualche posto nella lista delle funzioni Turing computabili, vale a dire che $u(n) = f_m(n)$ per qualche numero naturale m. Ma allora si deve presentare uno di questi due casi

1. $u(m) = 1$ perchè $f_m(m)$ non é definito; ma questo non é possibile perchè $f_m(m) = u(m) = 1$ e quindi $f_m(m)$ é definito
2. $u(m) = f_m(m)+1$; ma anche questo non é possibile perchè $u(m) = f_m(m)$ e quindi otterremmo $u(m) = u(m) + 1$ che porta subito a $0 = 1$.

Tuttavia non sembra poi così difficile calcolare in pratica la funzione $u(-)$. Per calcolarla sul numero n basta infatti guardare quale funzione c'é al posto n nella lista delle funzioni Turing computabili che abbiamo costruito in modo effettivo e calcolare poi tale funzione e aggiungere 1 al risultato qualora tale funzione terminasse e rispondere 1 se questa funzione non dovesse terminare.

Il problema é perciò ridotto alla questione di vedere se una certa funzione Turing computabile si arresta oppure no su un certo valore. Vedremo che in realtà é questo il problema che non é Turing computabile.

Supponiamo infatti che il problema dell'arresto di una macchina di Turing sia Turing computabile. Possiamo allora immaginarci che esista una macchina di Turing H che lavora su un nastro che contiene all'inizio due numeri m e n. Il numero m indica la posizione nella lista delle funzioni Turing computabili della macchina che vogliamo vedere se si arresta mentre il numero n é il valore su cui vogliamo vedere se la macchina di posto m si arresta. Tanto per fissare le idee, possiamo supporre che H si arresti lasciando sul nastro il numero 2 se la macchina di posto m si arresta quando parte con input n o il numero 1 altrimenti. La macchina H calcola quindi la funzione

$$h(m,n) = \begin{cases} 2 & \text{se } M_m \text{ si arresta su } n \\ 1 & \text{altrimenti} \end{cases}$$

Ma se H é una macchina di Turing allora possiamo costruire anche una macchina di Turing M che opera su un nastro che contiene un unico numero n nel modo che segue: prima duplica il numero contenuto nel nastro, poi fa partire la macchina H sul nastro che contiene ora due numeri uguali n ed infine si arresta lasciando come risultato 1 sul nastro se e solo se dopo aver eseguito H trova un 1 sul nastro (viene lasciato come esercizio al lettore la costruzione di tale macchina di Turing sotto l'ipotesi di avere la macchina H). La macchina M calcola quindi la funzione

$$m(n) = \begin{cases} 1 & \text{se } h(n,n) = 1 \\ \uparrow & \text{altrimenti} \end{cases}$$

dove con $\uparrow$ intendiamo dire che la funzione m non é definita su n.

Sia m il posto della macchina M che abbiamo costruito nella lista delle funzioni Turing computabile. Se ora proviamo a vedere se la macchina M si arresta sul valore m ci troviamo in un bel problema! Infatti

$$M \text{ si arresta su } m \text{ sse } h(m, m) = 2 \text{ sse } M \text{ non si arresta su } m$$

Abbiamo quindi ottenuto una conclusione assurda partendo dall'ipotesi che ci sia una macchina di Turing che decide sull'arresto delle macchine di Turing. L'unica via di uscita é naturalmente convincersi che tale macchina non può esistere. Abbiamo perciò dimostrato che il problema dell'arresto di una macchina di Turing non é Turing computabile.

3.2.1 Esercizi

1. Costruire un programma che faccia la somma di due numeri naturali.
2. Costruire un programma che faccia il prodotto di due numeri naturali.
3. ???
4. Dimostrare che l'insieme delle parole ottenute da un alfabeto numerabile é numerabile

3.3 La formalizzazione delle macchine di Turing

Vogliamo ora utilizzare il risultato della sezione precedente a proposito della non Turing-computabilità del problema dell'arresto per dimostrare che non é Turing computabile anche il problema della decidibilità della logica predicativa, vale a dire che vogliamo dimostrare che non c'é nessuna macchina di Turing che possa in generale decidere se una certa formula é oppure no un teorema della logica predicativa.

Per dimostrare questo risultato faremo vedere che presa una qualsiasi macchina di Turing M e un qualsiasi numero naturale n possiamo descrivere un insieme finito di formule del primo ordine Δ e una formula H tali che $\Delta \vdash H$ se e solo se la macchina M si ferma quando sul nastro trova il numero n. L'idea che ci guiderà nella ricerca dell'insieme di formule Δ e della formula H sarà che le formule in Δ dovranno descrivere le operazioni che la macchina M compie e la formula H dovrà dire che essa si arresta. Quindi, se fosse possibile risolvere il problema della decisione della conseguenza logica, vale a dire se fosse possibile decidere se la formula H é derivabile dalle formule in Δ, allora potremo anche decidere sul problema dell'arresto.

Vediamo allora come costruire le formule in Δ. Si tratta naturalmente di vedere come possiamo formalizzare con delle formule del primo ordine una macchina di Turing generica. Cominciamo quindi a vedere come descrivere il nastro. Possiamo partire immaginando che i vari foglietti sul nastro siano numerati utilizzando i numeri interi $\ldots, -2, -1, 0, 1, 2, \ldots$ e che anche il tempo sia suddiviso in una serie di istanti in cui la macchina esegue un passo nel suo

calcolo. Possiamo inoltre supporre senza perdere di generalità, che la macchina all'istante iniziale 0 stia guardando il foglietto numerato con lo 0. Dobbiamo ora decidere in quale linguaggio descrivere la macchina. Prendiamo quindi un simbolo di predicato Q_i a due posti per ogni stato q_i che la macchina può assumere e un simbolo di predicato S_j a due posti per ogni simbolo S_j che compare nell'alfabeto che la macchina può trattare. Inoltre, assumeremo di avere la possibilità di usare anche il simbolo di uguaglianza, che indicheremo per semplicità con $=$, un simbolo $<$ per una relazione di pre-ordine a due posti, un simbolo 0 per una costante e un simbolo $'$ per una funzione unaria oltre ai soliti simboli per i connettivi e i quantificatori.

L'idea é che 0 rappresenti lo 0 e $'$ rappresenti la funzione che dato un qualsiasi numero intero ne fornisce il successivo mentre il simbolo predicativo Q_i deve essere interpretato in modo tale che $t\,Q_i\,x$ é vero se e solo se all'istante t la macchina é nello stato q_i e guarda il foglietto x mentre il simbolo predicativo S_j deve essere interpretato in modo tale che $t\,S_j\,x$ é vero se e solo se all'istante t il simbolo sul foglietto x é S_j.

Vediamo finalmente quali sono le formule in Δ. L'insieme di formule Δ deve essere una descrizione della particolare macchina M che vogliamo analizzare e quindi le formule che metteremo in Δ dipendono da M. Supponiamo allora che la macchina M utilizzi i simboli S_0, S_1, $\ldots$, S_r e descriviamo le formule da mettere in Δ guardando quali istruzioni sono previste per la macchina M.

- Se in M c'é l'istruzione
$$q_i \quad S_j \quad S_k \quad q_m$$
allora metteremo in Δ la formula
$$\begin{aligned}&\forall t\forall x\{[t\,Q_i\,x\ \&\ t\,S_j\,x]\rightarrow\\ &\qquad[t'\,Q_m\,x\ \&\ t'\,S_k\,x\ \&\\ &\qquad\ \forall y[(y\neq x)\rightarrow((t\,S_0\,y\rightarrow t'\,S_0\,y)\ \&\ \ldots\ \&\ (t\,S_r\,y\rightarrow t'\,S_r\,y))]\}\end{aligned}$$
che, con la nostra interpretazione, sostiene che se al tempo t la macchina é nello stato q_i e sta guardando il foglietto x in cui c'é scritto S_j allora al tempo $t+1$ la macchina sarà nello stato q_k guardando ancora il foglietto x in cui sarà ora scritto il simbolo S_k mentre in tutti gli altri foglietti diversi da x all'istante $t+1$ ci sarà scritto esattamente lo stesso simbolo che c'era scritto all'istante t.

- Se nella macchina M c'é l'istruzione
$$q_i \quad S_j \quad R \quad q_m$$
allora metteremo in Δ la formula
$$\begin{aligned}&\forall t\forall x\{[t\,Q_i\,x\ \&\ t\,S_j\,x]\rightarrow\\ &\qquad[t'\,Q_m\,x'\ \&\\ &\qquad\ \forall y[(t\,S_0\,y\rightarrow t'\,S_0\,y)\ \&\ \ldots\ \&\ (t\,S_r\,y\rightarrow t'\,S_r\,y)]\}\end{aligned}$$
con l'ovvia interpretazione

- Infine se nella macchina M c'é l'istruzione

$$q_i \quad S_j \quad L \quad q_m$$

allora metteremo in Δ la formula

$$\begin{aligned}\forall t \forall x \{ [& t\, Q_i\, x' \ \&\ t\, S_j\, x'] \rightarrow \\ & [t'\, Q_m\, x\ \& \\ & \forall y[(t\, S_0\, y \rightarrow t'\, S_0\, y)\ \&\ \ldots\ \&\ (t\, S_r\, y \rightarrow t'\, S_r\, y)]\}\end{aligned}$$

di nuovo con l'ovvia interpretazione una volta che si noti il trucco utilizzato per rappresentare lo spostamento a sinistra nella macchina.

Alle formule precedenti dovremo aggiungere una formula che ci descriva le condizioni iniziali della macchina, vale a dire, che al momento iniziale la macchina é nello stato q_1 e che guarda una sequenza di n simboli S_1 su un nastro che per il resto é completamente bianco.

$$\begin{aligned}\text{(stato iniziale)} \quad & 0\, Q_1\, 0\ \&\ 0\, S_1\, 0\ \&\ 0\, S_1\, 0'\ \&\ \ldots\ \&\ 0\, S_1\, 0^{(n-1)}\ \& \\ & \forall y[(y \neq 0\ \&\ y \neq 0'\ \&\ \ldots\ \&\ y \neq 0^{(n-1)}) \rightarrow 0\, S_0\, y]\end{aligned}$$

dove con la notazione 0^k intendiamo il indicare il termine $0''''''''$ costruito applicando il simbolo $'$ un numero k di volte al simbolo 0. Naturalmente, se al momento iniziale il nastro fosse vuoto la formula qui sopra diventa la semplice

$$0\, Q_1\, 0\ \&\ \forall y(0\, S_0\, y)$$

Dobbiamo aggiungere ora quel minimo di formule che ci servono per poter dire che il simbolo funzionale $'$ si comporta come la funzione successore, e che il simbolo predicativo $<$ si comporta come una relazione di pre-ordine.

$$\begin{aligned}&\text{(surittività)} && \forall x \exists y\ (x = y') \\ &\text{(iniettività)} && \forall x \forall y (x' = y' \rightarrow x = y) \\ &\text{(transitività)} && \forall x \forall y \forall z\ (x < y\ \&\ y < z \rightarrow x < z) \\ &\text{(linearità)} && \forall x \forall y\ (x' = y \rightarrow x < y) \\ &\text{(antiriflessività)} && \forall x \forall y\ (x < y \rightarrow x \neq y)\end{aligned}$$

Vediamo ora quale deve essere la formula H che deve risultare vera esattamente nel momento in cui la macchina si arresta. Notiamo intanto che una macchina di Turing si ferma se esiste un istante di tempo t in cui essa si trova in uno stato q_i e legge sul nastro il simbolo S_j e non esiste nessuna istruzione per la macchina che dica cosa fare in questo caso. Possiamo quindi decidere che la formula H é la disgiunzione di tutte le formule

$$\exists t \exists x\ (t\, Q_i\, x\ \&\ t\, S_j\, x)$$

tali che non c'é alcuna istruzione che cominci con $q_i S_j$ nel programma della macchina M. Notate che si tratta comunque della disgiunzione di una quantità finita di formule. Nel caso in cui nel programma per ogni stato q_i ed ogni simbolo S_j ci sia una istruzione che comincia con $q_i S_j$ accade ovviamente che il programma non si arresta mai. In questo caso possiamo semplicemente scegliere per H la formula sempre falsa $0 \neq 0$.

3.3.1 Esercizi

-
-
-

3.4 La riduzione della decidibilità all'arresto

Vogliamo ora dimostrare che la macchina di Turing M si ferma sull'input n se e solo se $\Delta \vdash H$. Visto che la spiegazione intuitiva dei predicati si può facilmente trasformare in una vera e propria interpretazione valida é ovvio che se $\Delta \vdash H$ allora la macchina M in effetti si ferma sull'input n.

Dimostrare l'altra implicazione é più complicato. Per prima cosa ci serve decidere come lavorare quando abbiamo a che fare con un numero negativo visto che la nostra formalizzazione prevede solo l'uso dello 0 e del successore. Supponiamo allora che p sia un numero negativo, allora $-p$ é un numero positivo e useremo quindi le seguenti abbreviazioni:

$$\begin{aligned} t\, Q_i\, 0^{(p)} &\equiv \exists z\ (t\, Q_i\, z\ \&\ z^{(-p)} = 0) \\ t\, S_j\, 0^{(p)} &\equiv \exists z\ (t\, S_j\, z\ \&\ z^{(-p)} = 0) \\ y \neq 0^{(p)} &\equiv \exists z\ (t \neq z\ \&\ z^{(-p)} = 0) \end{aligned}$$

Nella prova che segue giocherà un ruolo essenziale la formula che permette di descrivere la macchina in un certo istante s. Tale formula dovrà contenere informazioni sullo stato della macchina all'istante s, su quale casella é scandita in quel momento e su quale simbolo c'é sui tutti i foglietti del nastro. Una formula adatta é:

$$\begin{aligned} 0^{(s)}\, Q_i\, 0^{(p)}\ \&\ 0^{(s)}\, S_{j_1}\, 0^{(p_1)}\ \&\ \ldots\ \&\ 0^{(s)}\, S_j\, 0^{(p)}\ \&\ \ldots\ \&\ 0^{(s)}\, S_{j_v}\, 0^{(p_v)}\ \& \\ \forall y[(y \neq 0^{(p_1)}\ \&\ \ldots\ \&\ y \neq 0^{(p)}\ \&\ \ldots\ \&\ y \neq 0^{(p_v)}) \rightarrow 0^{(s)}\, S_0\, y] \end{aligned}$$

Supponiamo ora che la macchina M che stiamo considerando si arresti quando parte su un nastro che contiene il numero n. Allora per qualche numero intero s e p e per qualche stato q_i e per qualche simbolo S_j succede che la macchina nell'istante s sarà nello stato q_i guardando il foglietto p su cui leggerà il simbolo S_j e che nessuna istruzione per M comincia con $q_i S_j$. Se succedesse ora che la formula che descrive la macchina nel istante s fosse davvero fedele alla vera situazione della macchina in quel istante allora tale descrizione dovrebbe contenere tra i suoi congiunti

$$0^{(s)}\, Q_i\, 0^{(p)}\ \text{e}\ 0^{(s)}\, S_j\, 0^{(p)}$$

e quindi da essa sarebbe possibile dedurre

$$\exists t \exists x\ (t\, Q_i\, x\ \&\ t\, S_j\, x)$$

che a sua volta implicherebbe la derivabilità di H visto che $\exists t \exists x\ (t\, Q_i\, x\ \&\ t\, S_j\, x)$ é uno dei disgiunti che appaiono in H.

Quindi per dimostrare che da Δ si può dedurre H ci basta far vedere per per ogni s non negativo, se la macchina non si ferma prima dell'istante s allora Δ implica la descrizione della macchina all'istante s. Possiamo dimostrare questo risultato per induzione su s.

- (Base) Δ contiene la formula che descrive lo stato iniziale della macchina e quindi evidentemente la implica ed essa é proprio la descrizione della macchina all'istante 0.

- (Passo induttivo) Supponiamo che fino all'istante s la macchina non sia sia ancora arrestata. Allora per ipotesi induttiva Δ implica la descrizione della macchina all'istante s, vale a dire che all'istante s la macchina si trova nello stato q_i e sta guardando il foglietto $0^{(p)}$ su cui sta scritto il simbolo S_j. Supponiamo inoltre che la macchina non si arresti all'istante s. Dobbiamo quindi dimostrare che Δ implica la descrizione della macchina M all'istante $s+1$. Dal momento che la macchina non si ferma all'istante s, tra quelle di M deve esserci una tra le seguenti istruzioni

$$\begin{array}{llll} (1) & q_i & S_j & S_k & q_m \\ (2) & q_i & S_j & R & q_m \\ (3) & q_i & S_j & L & q_m \end{array}$$

  Vediamo di analizzare ora tutti i tre casi possibili.

  1. Se siamo in questo caso allora una tra le formule in Δ é

$$\begin{array}{l} \forall t \forall x \{[t\, Q_i\, x\ \&\ t\, S_j\, x] \rightarrow \\ \qquad [t'\, Q_m\, x\ \&\ t'\, S_k\, x\ \& \\ \qquad\ \forall y[(y \neq x) \rightarrow \\ \qquad\qquad ((t\, S_0\, y \rightarrow t'\, S_0\, y)\ \&\ \ldots\ \&\ (t\, S_r\, y \rightarrow t'\, S_r\, y))]\} \end{array}$$

     Non é difficile vedere che essa assieme alla descrizione della macchina all'istante s e alle proprietà che abbiamo richiesto su $'$ e $<$ implicano che

$$\begin{array}{l} 0^{(s+1)}\, Q_i\, 0^{(p)}\ \&\ 0^{(s+1)}\, S_{j_1}\, 0^{(p_1)}\ \&\ \ldots\ \& \\ \qquad 0^{(s+1)}\, S_k\, 0^{(p)}\ \&\ \ldots 0^{(s+1)}\, S_{j_v}\, 0^{(p_v)}\ \& \\ \qquad \forall y[(y \neq 0^{(p_1)}\ \&\ \ldots\ \&\ y \neq 0^{(p)}\ \&\ \ldots\ \&\ y \neq 0^{(p_v)}) \rightarrow \\ \qquad\quad 0^{(s+1)}\, S_0\, y] \end{array}$$

     che é una descrizione della macchina all'istante $s+1$.

  2. Se siamo in questo caso allora una tra le formule in Δ é

$$\begin{array}{l} \forall t \forall x \{[t\, Q_i\, x\ \&\ t\, S_j\, x] \rightarrow \\ \qquad [t'\, Q_m\, x'\ \& \\ \qquad\ \forall y[((t\, S_0\, y \rightarrow t'\, S_0\, y)\ \&\ \ldots\ \&\ (t\, S_r\, y \rightarrow t'\, S_r\, y))]\} \end{array}$$

Non é difficile vedere che essa assieme alla descrizione della macchina all'istante s e alle proprietà che abbiamo richiesto su $'$ e $<$ implicano che

$$0^{(s+1)}\, Q_i\, 0^{(p+1)}\ \&\ 0^{(s+1)}\, S_{j_1}\, 0^{(p_1)}\ \&\ \ldots\ \&\ 0^{(s+1)}\, S_k\, 0^{(p+1)}\ \&\ \ldots 0^{(s+1)}\, S_{j_v}\, 0^{(p_v)}\ \&\ \forall y[(y \neq 0^{(p_1)}\ \&\ \ldots\ \&\ y \neq 0^{(p+1)}\ \&\ \ldots\ \&\ y \neq 0^{(p_v)}) \rightarrow 0^{(s+1)}\, S_0\, y]$$

che é una descrizione della macchina all'istante $s+1$.

3. Se siamo in questo caso allora una tra le formule in Δ é

$$\forall t \forall x\{[t\, Q_i\, x'\ \&\ t\, S_j\, x'] \rightarrow [t'\, Q_m\, x\ \&\ \forall y[((t\, S_0\, y \rightarrow t'\, S_0\, y)\ \&\ \ldots\ \&\ (t\, S_r\, y \rightarrow t'\, S_r\, y))]\}$$

Non é difficile vedere che essa assieme alla descrizione della macchina all'istante s e alle proprietà che abbiamo richiesto su $'$ e $<$ implicano che

$$0^{(s+1)}\, Q_i\, 0^{(p-1)}\ \&\ 0^{(s+1)}\, S_{j_1}\, 0^{(p_1)}\ \&\ \ldots\ \&\ 0^{(s+1)}\, S_k\, 0^{(p-1)}\ \&\ \ldots 0^{(s+1)}\, S_{j_v}\, 0^{(p_v)}\ \&\ \forall y[(y \neq 0^{(p_1)}\ \&\ \ldots\ \&\ y \neq 0^{(p-1)}\ \&\ \ldots\ \&\ y \neq 0^{(p_v)}) \rightarrow 0^{(s+1)}\, S_0\, y]$$

che é una descrizione della macchina all'istante $s+1$.

Quindi in tutti tre i casi Δ implica una descrizione della macchina all'istante $s+1$. Abbiamo quindi finito la dimostrazione dell'indecidibilità della logica del primo ordine per mezzo delle macchine di Turing.

3.4.1 Esercizi

1. Dimostrare che

$$0\, Q_1\, 0\ \&\ 0\, S_1\, 0\ \&\ 0\, S_1\, 0'\ \&\ \ldots 0\, S_1\, 0^{(n-1)}\ \&\ \forall y[(y \neq 0\ \&\ y \neq 0'\ \&\ \ldots\ \&\ y \neq 0^{(n-1)}) \rightarrow 0\, S_0\, y]$$

dove con la notazione 0^k intendiamo il indicare il termine $0'''''''$ costruito applicando il simbolo $'$ un numero k di volte al simbolo 0, é una descrizione della macchina all'istante 0.

2. Scrivere le formule di Δ e la formula H per la seguente macchina di Turing

$$\begin{array}{cccc} q_1 & S_1 & R & q_1 \\ q_1 & S_0 & L & q_2 \\ q_2 & S_1 & S_0 & q_2 \end{array}$$

e verificare che $\Delta \vdash H$ nel caso che la macchina sia applicata ad un nastro completamente bianco.

3.