B è NP-completo, ogni linguaggio appartenente a NP è riducibile in tempo polinomiale a B, e B a sua volta è riducibile in tempo polinomiale a C. Le riduzioni di tempo polinomiale possono essere composte; cioè, se A è riducibile in tempo polinomiale a B e B è riducibile in tempo polinomiale a C, allora A è riducibile in tempo polinomiale a C. Pertanto, ogni linguaggio appartenente a NP è riducibile in tempo polinomiale a C.

Il Teorema di Cook e Levin

Una volta che disponiamo di un problema NP-completo, possiamo ottenerne altri attraverso riduzioni polinomiali da esso. Tuttavia, determinare il primo problema NP-completo è più difficile. Lo facciamo ora, provando che SAT è NP-completo.

TEOREMA 7.37

SAT è NP-completo.[2]

Questo teorema implica il Teorema 7.27.

IDEA. Mostrare che SAT appartiene a NP è facile, e lo faremo velocemente. La parte difficile della dimostrazione è far vedere che qualsiasi linguaggio appartenente a NP è riducibile in tempo polinomiale a SAT.

Per far ciò, costruiremo una riduzione di tempo polinomiale per ciascun linguaggio A appartenente a NP a SAT. La riduzione per A prende una stringa w e produce una formula booleana ϕ che simula la macchina NP per A su input w. Se la macchina accetta, ϕ ha un assegnamento che la soddisfa che corrisponde ad una computazione accettante. Se la macchina non accetta, nessun assegnamento soddisfa ϕ. Pertanto, w appartiene ad A se e solo se ϕ è soddisfacibile.

In realtà, costruire la riduzione per farla funzionare in questo modo è un compito concettualmente semplice, sebbene debbano essere gestiti diversi dettagli. Una formula booleana può contenere le operazioni booleane AND, OR, e NOT, e queste operazioni costituiscono la base per la circuiteria usata nei calcolatori elettronici. Quindi, il fatto che possiamo progettare una formula booleana per simulare una macchina di Turing non è sorprendente. I dettagli sono nell'implementazione di questa idea.

DIMOSTRAZIONE. Prima di tutto, mostriamo che SAT appartiene a NP. Una macchina non deterministica di tempo polinomiale può ipotizzare un assegnamento per una data formula ϕ ed accettare se l'assegnamento soddisfa ϕ.

[2]Una dimostrazione alternativa del teorema viene data in Sezione 9.3.

Successivamente, prendiamo un qualsiasi linguaggio A appartenente a NP e facciamo vedere che A è riducibile in tempo polinomiale a SAT. Sia N una macchina di Turing non deterministica che decide A in tempo n^k per qualche costante k. (Per comodità in effetti assumiamo che N computi in tempo $n^k - 3$; ma soltato i lettori interessati ai dettagli dovrebbero preoccuparsi di questo punto minore.) La nozione che segue aiuta a descrivere la riduzione.

Un ***tableau*** per N su w è una tabella $n^k \times n^k$ le cui righe sono le configurazioni di una diramazione della computazione di N su input w, come mostrato nella figura seguente.

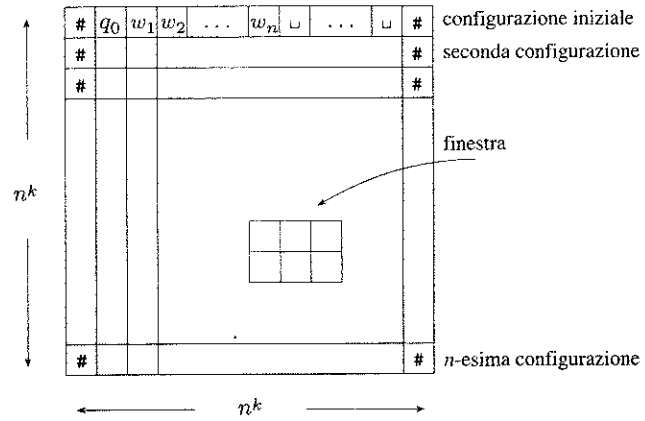

FIGURA 7.38
Un tableau è una tabella $n^k \times n^k$ di configurazioni

Per convenienza nel seguito assumiamo che ciascuna configurazione inizi e finisca con un simbolo #. Pertanto, la prima e l'ultima colonna di un tableau sono tutti #. La prima riga di un tableau è la configurazione iniziale di N su w, e ciascuna riga segue dalla precedente in accordo alla funzione di transizione di N. Un tableau è ***accettante*** se qualche riga del tableau è una configurazione accettante.

Ogni tableau accettante per N su w corrisponde ad una diramazione della computazione accettante di N su w. Quindi, il problema di stabilire se N accetta w è equivalente al problema di stabilire se esiste un tableau accettante per N su w.

Passiamo ora alla descrizione della riduzione di tempo polinomiale f di A a SAT. Su input w, la riduzione produce una formula ϕ. Iniziamo descrivendo le variabili di ϕ. Siano Q e Γ l'insieme degli stati e l'alfabeto del nastro di N, rispettivamente. Sia $C = Q \cup \Gamma \cup \{\#\}$. Per ciascun i e j tra 1 ed n^k e per ciascun s in C, introduciamo una variabile, $x_{i,j,s}$.

Ciascuna delle $(n^k)^2$ entrate di un tableau è chiamata ***cella***. La cella in riga i e colonna j viene denotata con $cell[i,j]$ e contiene un simbolo di C. Rappresentiamo i contenuti delle celle con le variabili di ϕ. Se $x_{i,j,s}$ assume il valore 1, significa che $cell[i,j]$ contiene s.

Progettiamo a questo punto ϕ in modo tale che un assegnamento che soddisfa le variabili di ϕ corrisponda ad un tableau accettante per N su w. La formula ϕ è l'AND di quattro parti: $\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$. Descriviamo ciascuna parte, una per volta.

Come anticipato precedentemente, porre a 1 la variabile $x_{i,j,s}$ corrisponde a posizionare il simbolo s in $cell[i,j]$. La prima cosa che dobbiamo garantire al fine di ottenere una corrispondenza tra un assegnamento ed un tableau è che l'assegnamento ponga ad 1 esattamente una variabile per ciascuna cella. La formula ϕ_{cell} garantisce questo requisito esprimendolo in termini di operazioni booleane:

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right].$$

I simboli $\bigwedge$ e $\bigvee$ stanno per AND ed OR iterati. Per esempio, l'espressione nella formula precedente

$$\bigvee_{s \in C} x_{i,j,s}$$

è un'abbreviazione per

$$x_{i,j,s_1} \vee x_{i,j,s_2} \vee \cdots \vee x_{i,j,s_l}$$

dove $C = \{s_1, s_2, \ldots, s_l\}$. Quindi, ϕ_{cell} è in realtà un'espressione grande che contiene un frammento per ciascuna cella nel tableau, poiché i e j variano da 1 a n^k. La prima parte del frammento stabilisce che almeno una variabile assume valore 1 nella cella corrispondente. La seconda parte di ciascun frammento stabilisce che non più di una variabile assume valore 1 (letteralmente, stabilisce che, in ogni coppia di variabili, almeno una assume valore 0) nella cella corrispondente. Questi frammenti sono collegati attraverso operazioni $\wedge$.

La prima parte di ϕ_{cell} all'interno delle parentesi garantisce che almeno una variabile che è associata con ciascuna cella vale 1, laddove la seconda parte garantisce che non più di una variabile vale 1 per ciascuna cella. Qualsiasi assegnamento alle variabili che soddisfa ϕ (e di conseguenza ϕ_{cell}) deve avere esattamente una variabile ad 1 per ogni cella. Pertanto, qualsiasi assegnamento che soddisfa ϕ specifica un simbolo in ciascuna cella della tabella. Le parti ϕ_{start}, ϕ_{move}, e ϕ_{accept} garantiscono che questi simboli corrispondano realmente ad un tableau accettante come segue.

La formula ϕ_{start} assicura che la prima riga della tabella sia la configurazione iniziale di N su w, richiedendo esplicitamente che le variabili

corrispondenti siano ad 1:

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}.\end{aligned}$$

La formula ϕ_{accept} garantisce che una configurazione accettante sia presente nel tableau. Essa assicura che q_{accept}, il simbolo per lo stato di accettazione, compaia in una delle celle del tableau, richiedendo che una delle variabili corrispondenti sia a 1:

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}.$$

Infine, la formula ϕ_{move} garantisce che ciascuna riga del tableau corrisponda ad una configurazione che segue legittimamente dalla configurazione della riga precedente, in accordo alle regole di N. Lo fa assicurando che ogni finestra di celle 2×3 sia lecita. Diciamo che una finestra 2×3 è ***lecita*** se non viola le azioni specificate dalla funzione di transizione di N. In altre parole, una finestra è lecita se può esser presente quando una configurazione segue correttamente da un'altra.[3]

Per esempio, siano a, b, e c elementi dell'alfabeto di nastro, e q_1 e q_2 stati di N. Si assuma che quando nello stato q_1 con la testina che legge una a, N scrive una b, resta nello stato q_1, e muove a destra; e che quando nello stato q_1 con la testina che legge una b, N non deterministicamente

1. scrive una c, entra in q_2, e muove a sinistra, oppure
2. scrive una a. entra in q_2, e muove a destra.

In termini formali, $\delta(q_1, \mathtt{a}) = \{(q_1, \mathtt{b}, \mathrm{R})\}$ e $\delta(q_1, \mathtt{b}) = \{(q_2, \mathtt{c}, \mathrm{L}), (q_2, \mathtt{a}, \mathrm{R})\}$. Esempi di finestre lecite per questa macchina sono mostrati in Figura 7.39.

[3] Potremmo dare una qui definizione precisa di ***finestra lecita***, in termini della funzione di transizione. Ma far ciò è alquanto tedioso e ci distrarrebbe dal cuore dell'argomento della dimostrazione. Chiunque desiderasse maggior precisione dovrebbe far riferimento all'analisi relativa nella dimostrazione del Teorema 5.15, l'indecidibilità del Problema della Corrispondenza di Post.

(a)
a	q_1	b
q_2	a	c

(b)
a	q_1	b
a	a	q_2

(c)
a	a	q_1
a	a	b

(d)
#	b	a
#	b	a

(e)
a	b	a
a	b	q_2

(f)
b	b	b
c	b	b

FIGURA 7.39
Esempi di finestre lecite

In Figura 7.39, le finestre (a) e (b) sono lecite perché la funzione di transizione permette ad N di muovere nella direzione indicata. La finestra (c) è lecita perché, con q_1 presente sul lato destro della riga di sopra, non sappiamo su quale simbolo la testina sia. Detto simbolo potrebbe essere una a, e q_1 potrebbe cambiarlo in una b e muovere a destra. Una tale possibilità genererebbe questa finestra, quindi essa non viola le regole di N. La finestra (d) è ovviamente lecita perché la riga di sopra e quella di sotto sono identiche, situazione che si verifica se la testina non è adiacente alla posizione della finestra. Si noti che # in una finestra lecita può comparire a sinistra o a destra sia nella riga superiore che in quella inferiore. La finestra (e) è lecita perché lo stato q_1 potrebbe essere immediatamente a destra della riga superiore e la macchina, leggendo una b, si sarebbe mossa a sinistra nello stato q_2 che ora compare all'estremità destra della riga inferiore. Infine, la finestra (f) è lecita perché lo stato q_1 potrebbe essere stato immediatamente a sinistra della riga superiore, e la macchina potrebbe aver cambiato la b in una c ed effettuato una mossa a sinistra.

Le finestre mostrate nella figura seguente non sono lecite per la macchina N.

(a)
a	b	a
a	a	a

(b)
a	q_1	b
q_2	a	a

(c)
b	q_1	b
q_2	b	q_2

FIGURA 7.40
Esempi di finestre non lecite

Nella finestra (a), il simbolo centrale nella riga superiore non può cambiare perché non c'è uno stato ad esso adiacente. La finestra (b) non è lecita perché la funzione di transizione specifica che la b viene cambiata in una c ma non in una a. La finestra (c) non è lecita perché due stati compaiono nella riga inferiore.

FATTO 7.41

Se la riga superiore del tableau è la configurazione iniziale ed ogni finestra nel tableau è lecita, ciascuna riga del tableau è una configurazione che segue legittimamente dalla precedente.

Proviamo l'asserto considerando ogni coppia di configurazioni adiacenti nel tableau, indicate come la configurazione superiore e la configurazione inferiore. Nella configurazione superiore, ogni cella che contiene un simbolo di nastro e non è adiacente ad un simbolo di stato rappresenta la cella centrale superiore in una finestra in cui la riga superiore non contiene stati. Pertanto, questo stesso simbolo deve comparire al centro in basso nella finestra. Quindi, esso è presente nella stessa posizione nella configurazione inferiore.

La finestra contenente il simbolo di stato nella cella centrale superiore garantisce che le tre posizioni corrispondenti vengano aggiornate consistentemente tramite la funzione di transizione. Pertanto, se la configurazione superiore è una configurazione lecita, altrettanto lo è la configurazione inferiore, e quella inferiore discende da quella superiore in accordo alle regole di N. Si noti che questa dimostrazione, seppur immediata, dipende in maniera cruciale dalla nostra scelta di utilizzare una finestra di dimensione 2×3, come mostra il Problema 7.26.

Torniamo ora alla costruzione di ϕ_{move}. Essa garantisce che tutte le finestre nel tableau sono lecite. Ciascuna finestra contiene sei celle, che possono essere impostate in un numero fissato di modi per produrre una finestra lecita. La formula ϕ_{move} stabilisce che le impostazioni delle sei celle devono essere uno di questi modi, ossia

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} (\text{la finestra } (i,j) \text{ è lecita}).$$

La finestra (i,j) ha $cell[i,j]$ in posizione centrale in alto. Sostituiamo il testo "la finestra (i,j) è lecita" in questa formula con la formula seguente. Denotiamo il contenuto delle sei celle di una finestra con $a_1, \ldots, a_6$.

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{è una finestra lecita}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6})$$

Nel seguito analizziamo la complessità della riduzione per mostrare che essa opera in tempo polinomiale. Per far ciò, esaminiamo la dimensione di ϕ. Prima di tutto, stimiamo il numero di variabili che ha. Si ricordi che il tableau è una tabella $n^k \times n^k$, quindi contiene n^{2k} celle. Ciascuna cella ha l variabili associate ad essa, dove l è il numero di simboli in C. Poiché l dipende soltanto dalla TM N e non dalla lunghezza dell'input n, il numero totale di variabili è $O(n^{2k})$.

Stimiamo la dimensione di ciascuna delle parti di ϕ. La formula ϕ_{cell} contiene un frammento di dimensione fissata della formula per ciascuna

cella del tableau, quindi la sua dimensione è $O(n^{2k})$. La formula ϕ_{start} ha un frammento per ciascuna cella nella riga superiore, quindi la sua dimensione è $O(n^k)$. Le formule ϕ_{move} e ϕ_{accept} contengono ciascuna un frammento di dimensione fissata della formula per ciascuna cella del tableau, quindi la loro dimensione è $O(n^{2k})$. Pertanto, la dimensione complessiva di ϕ è $O(n^{2k})$. Tale limite è sufficiente per i nostri scopi perché dimostra che la dimensione di ϕ è polinomiale in n. Se fosse stata più che polinomiale, la riduzione non avrebbe avuto alcuna possibilità di generarla in tempo polinomiale. (In realtà, le nostre stime sono più basse di un fattore $O(\log n)$, poiché ciascuna variabile ha indici che possono arrivare fino a n^k e, quindi, possono richiedere $O(\log n)$ simboli da scrivere nella formula, ma questo fattore addizionale non cambia la polinomialità del risultato.)

Per rendersi conto che la formula può essere generata in tempo polinomiale, si noti la sua natura altamente ripetitiva. Ciascun componente della formula è composto da molti frammenti quasi identici, che differiscono soltanto negli indici in modo molto semplice. Pertanto, possiamo costruire facilmente una riduzione che produce ϕ in tempo polinomiale dall'input w.

Quindi, abbiamo concluso la dimostrazione del teorema di Cook e Levin, mostrando che *SAT* è NP-completo. Mostrare la NP-completezza di altri linguaggi generalmente non richiede una dimostrazione così lunga. Al contrario, la NP-completezza può essere provata con una riduzione di tempo polinomiale da un linguaggio che è già noto essere NP-completo. Possiamo usare *SAT* per questo scopo; ma usare *3SAT*, il caso speciale di *SAT* che abbiamo definito a pagina 323, è solitamente più facile. Si ricordi che le formule appartenenti a *3SAT* sono in forma normale congiuntiva (cnf) con tre letterali per clausola. Prima di tutto, dobbiamo dimostrare che *3SAT* stesso è NP-completo. Proviamo questo asserto come corollario del Teorema 7.37.

COROLLARIO 7.42

3SAT è NP-completo.

DIMOSTRAZIONE. Ovviamente *3SAT* appartiene a NP, quindi dobbiamo provare solamente che tutti i linguaggi appartenenti a NP si riducono a *3SAT* in tempo polinomiale. Un modo per farlo è facendo vedere che *SAT* si riduce in tempo polinomiale a *3SAT*. Invece, preferiamo farlo modificando la dimostrazione del Teorema 7.37, in modo tale che produca direttamente una formula in forma normale congiuntiva con tre letterali per clausola.

Il Teorema 7.37 produce una formula che è già quasi in forma congiuntiva normale. La formula ϕ_{cell} è un grosso AND di sottoformule, ciascuna delle quali contiene un grosso OR ed un grosso AND di diversi OR. Quindi, ϕ_{cell} è un AND di clausole e, di conseguenza, è già in forma cnf. La formula ϕ_{start} è un grosso AND di variabili. Prendendo ciascuna di queste variabili come clausole di dimensione 1, notiamo che ϕ_{start} è in forma cnf. La formula ϕ_{accept} è un grosso OR di variabili, ed è perciò una singola clausola. La formula ϕ_{move} è l'unica che non è già in forma cnf, ma possiamo facilmente convertirla in una formula che è in forma cnf come segue.

Si ricordi che ϕ_{move} è un grosso AND di sottoformule, ciascuna delle quali è un OR di diversi AND che descrive tutte le possibili finestre lecite. La legge distributiva, come descritto nel Capitolo 0, stabilisce che possiamo sostituire un OR di vari AND con un equivalente AND di vari OR. Fare ciò può incrementare significativamente la dimensione di ciascuna sottoformula, ma tale operazione può incrementare la dimensione complessiva di ϕ_{move} solamente di un fattore costante, poiché la dimensione di ciascuna sottoformula dipende solo da N. Il risultato è una formula che è in forma normale congiuntiva.

Ora che abbiamo scritto la formula in forma cnf, la convertiamo in una con tre letterali per clausola. In ciascuna clausola che correntemente ha uno o due letterali, duplichiamo uno dei letterali, fino a quando il numero totale diventa tre. Ciascuna clausola che ha più di tre letterali la dividiamo in più clausole e aggiungiamo ulteriori variabili per preservare la soddisfacibilità o non soddisfacibilità della clausola originale.

Per esempio, sostituiamo la clausola $(a_1 \vee a_2 \vee a_3 \vee a_4)$, dove ciascun a_i è un letterale, con l'espressione di due clausole $(a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$, in cui z è una variabile nuova. Se qualche impostazione degli a_i soddisfa la clausola originale, possiamo trovare una impostazione di z in modo tale che le due nuove clausole siano soddisfatte e viceversa. In generale, se la clausola contiene l letterali,

$$(a_1 \vee a_2 \vee \cdots \vee a_l),$$

possiamo sostituirla con le $l-2$ clausole

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee a_l).$$

È possibile verificare facilmente che la nuova formula è soddisfacibile se e solo se la formula originale lo era, quindi la dimostrazione è completa.

7.5 ULTERIORI PROBLEMI NP-COMPLETI

Il fenomeno della NP-completezza è molto diffuso. Problemi NP-completi emergono in svariati campi. Per ragioni che non sono ben comprese, molti