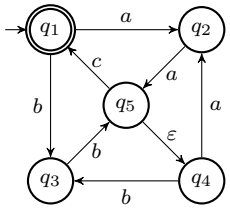
Automi e Linguaggi **(M. Cesati)**

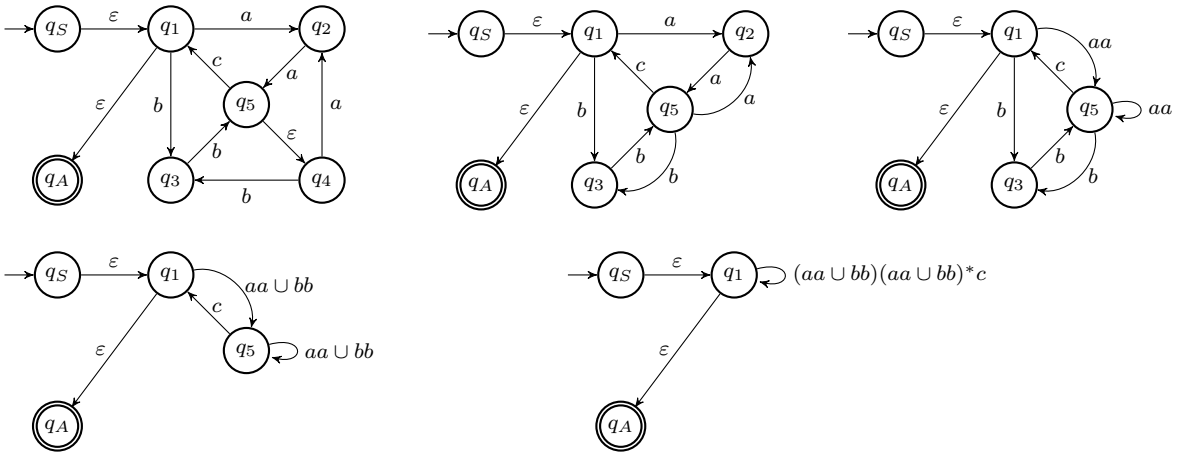
Facoltà di Ingegneria, Università degli Studi di Roma Tor Vergata

Compito scritto del 28 giugno 2021

Esercizio 1 [6] Derivare una espressione regolare per il linguaggio riconosciuto dall'automa:

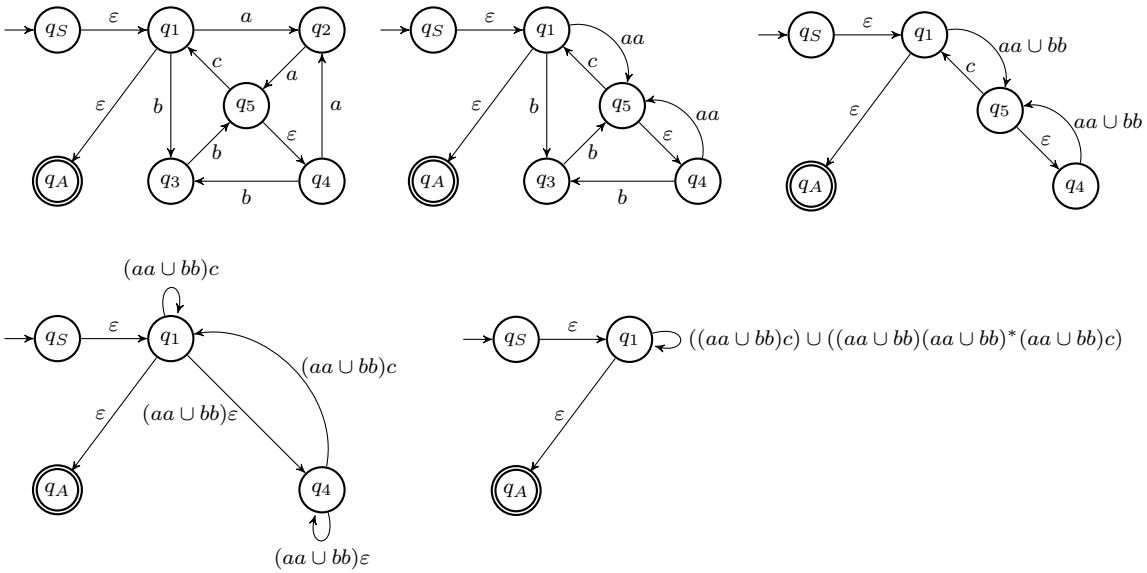


Soluzione: Convertiamo in un GNFA ed eliminiamo iterativamente i nodi dell'automa (per semplificare gli schemi non sono disegnati gli archi etichettati con '$\emptyset$'):



Eliminando ora il nodo q_1 si ottiene l'espressione regolare $((aa \cup bb)\,(aa \cup bb)^*\,c)^*$, ovvero $\Big((aa \cup bb)^+\,c\Big)^*$.

Cambiando l'ordine di eliminazione dei nodi si produce una diversa espressione regolare, comunque equivalente. Ad esempio:



Eliminando ora il nodo q_1 si ottiene l'espressione regolare

$$(((aa \cup bb)\, c) \cup ((aa \cup bb)\, (aa \cup bb)^*\, (aa \cup bb)\, c))^*$$

che può essere semplificata in $\Big(((aa \cup bb)\, c) \cup \Big((aa \cup bb)\, (aa \cup bb)^+\, c\Big)\Big)^*$ e successivamente in $\Big((aa \cup bb)^+\, c\Big)^*$.

Esercizio 2 [6] Derivare un automa a pila (PDA) per il linguaggio $A = \{x\texttt{\#}y \mid x, y \in \{\texttt{0}, \texttt{1}\}^*$ e x è la codifica binaria $\langle |y| \rangle$ della lunghezza di $y\}$, ovvero dimostrare che non è possibile derivare un tale PDA.

Soluzione: In effetti non è possibile costruire un PDA per il linguaggio A. È tuttavia estremamente difficile dimostrare questo fatto ragionando direttamente sul PDA, perché si dovrebbe dimostrare che *qualsiasi* algoritmo concepibile (tra gli infiniti possibili) fallirebbe nel riconoscere gli elementi di A. Possiamo invece ottenere il risultato ricordando che ogni linguaggio riconosciuto da un PDA è CFL e riconoscendo che A non è CFL. Per dimostrare l'ultimo asserto utilizziamo il "pumping lemma" per i CFL.

Supponiamo per assurdo che A sia CFL. A causa del "pumping lemma", deve pertanto esistere una lunghezza $p > 0$ tale che, per ogni elemento s di lunghezza maggiore o uguale a p, s può essere suddiviso come $s = uvxyz$ in modo tale che $|vy| > 0$, $|vxy| \leq p$, e per ogni $i \geq 0$, $uv^ixy^iz \in A$. Consideriamo la stringa $s = \texttt{1}^p\texttt{\#0}^{2^p-1}$: è immediato verificare che $|s| > p$ e $s \in A$. Dimostriamo che per ogni possibile suddivisione di s esiste un valore di $i \geq 0$ per il quale la stringa "pompata" *non* fa parte di A. In effetti una dimostrazione abbastanza semplice può

essere basata sul fatto che ogni simbolo `1` aggiunto a sinistra comporta almeno il raddoppio (in effetti la triplicazione) della lunghezza della stringa a destra; dunque il numero di simboli da aggiungere dovrebbe essere esponenziale in p, mentre la lunghezza della porzione y non può superare p. La stessa idea fondamentale viene sfruttata nella seguente dimostrazione formale in cui si utilizza esclusivamente il "pompaggio verso il basso".

Cominciamo con l'osservare che ogni suddivisione in cui $'\texttt{\#}' \notin x$ non può soddisfare il lemma. Infatti se $'\texttt{\#}' \notin x$, e considerando che $|vy| > 0$, allora o vxy deve essere tutto a sinistra del simbolo `#` oppure deve essere tutto alla sua destra (altrimenti il simbolo `#` sarebbe rimosso o duplicato, e la stringa non potrebbe far parte di A in quanto malformata). In entrambi i casi pompare verso il basso produce una stringa in cui la codifica binaria a sinistra non coincide con il numero di simboli a destra.

Assumiamo dunque che $'\texttt{\#}' \in x$, e pertanto $v \in \texttt{1}^*$ e $y \in \texttt{0}^*$. Analizziamo i seguenti casi basati sulla lunghezza della sottostringa v:

- $|v| = 0$: in questo caso, poiché $|vy| > 0$, deve valere $|y| > 0$. Pertanto pompando verso il basso, ossia considerando $i = 0$, si ottiene una stringa uxz in cui la codifica binaria a sinistra è identica ma il numero di elementi a destra è diminuito; tale stringa non può appartenere ad A.

- $|v| = 1$: consideriamo il valore $i = 0$, ossia la stringa $\texttt{1}^{p-1}\texttt{\#}\texttt{0}^l$ ottenuta pompando verso il basso, ove il valore l dipende dalla dimensione di y. Perché tale stringa possa far parte di A deve valere $l = 2^{p-1} - 1$, e quindi $|y| = 2^p - 1 - (2^{p-1} - 1) = 2^{p-1}$. Ma allora $|vxy| = 1 + 2^{p-1} + |x| \geq 2 + 2^{p-1}$; ricordando che $|vxy| \leq p$, si ottiene $2^{p-1} \leq p - 2$. In effetti non esiste alcun valore positivo per p che possa soddisfare tale disuguaglianza, e quindi non esiste suddivisione che possa conservare l'appartenza ad A pompando verso il basso.

- $|v| > 1$: si applica lo stesso ragionamento del caso precedente. Sia $j = |v|$, e consideriamo il valore $i = 0$. Necessariamente la lunghezza di y deve essere pari a $l = 2^p - 1 - (2^{p-j} - 1) = \left(1 - 2^{-j}\right) 2^p$. Pertanto: $p \geq |vxy| \geq j + 1 + \left(1 - 2^{-j}\right) 2^p$, e poiché $\left(1 - 2^{-j}\right) > 1/2$ per $j > 1$:

  $$2^{p-1} < 2^p \left(1 - 2^{-j}\right) \leq p - j - 1 < p - 1.$$

  Poiché questa disuguaglianza non ha alcuna soluzione per $p > 0$, non esista alcuna suddivisione che possa conservare l'appartenza ad A pompando verso il basso.

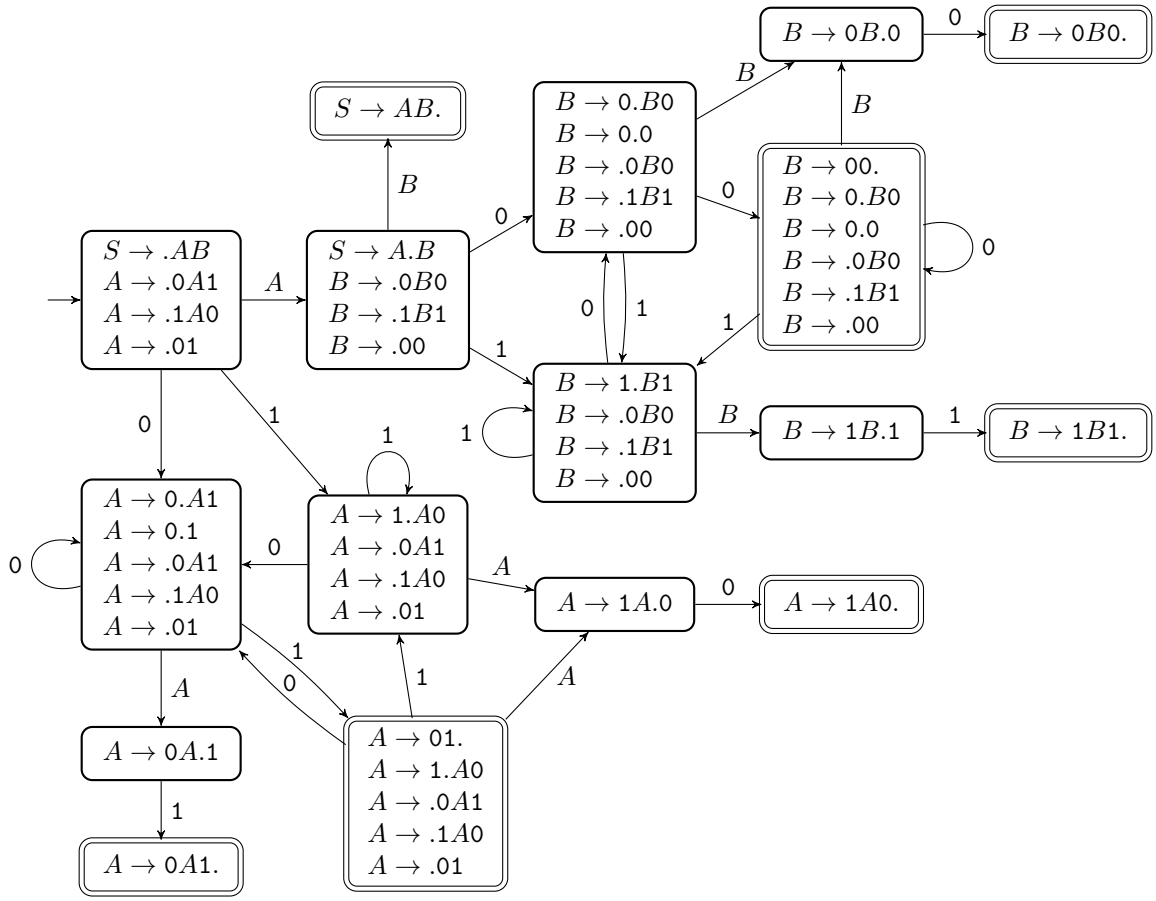
Poiché ogni possibile suddivisione di s risulta non appartente ad A quando pompata verso il basso, le conclusioni del "pumping lemma" non valgono. La contraddizione è dovuta alla

supposizione che A sia CFL. Concludiamo pertanto che non esiste alcun PDA che possa riconoscere gli elementi del linguaggio A.

Esercizio 3 [6] Determinare se la seguente grammatica libera dal contesto con variabile iniziale S è deterministica:

$$\begin{aligned} S \to &\quad A\,B \\ A \to &\quad \mathtt{0}A\mathtt{1} \mid \mathtt{1}A\mathtt{0} \mid \mathtt{01} \\ B \to &\quad \mathtt{0}B\mathtt{0} \mid \mathtt{1}B\mathtt{1} \mid \mathtt{00} \end{aligned}$$

Soluzione: Per determinare se la grammatica è deterministica applichiamo il DK-test. Si ottiene il seguente automa:



Poiché esistono diversi stati finali che includono regole con il punto seguito da un carattere terminale, possiamo immediatamente concludere che la grammatica non è deterministica.

Esercizio 4 [6] Dimostrare che un linguaggio L è ricorsivamente enumerabile (ossia Turing-riconoscibile) se e solo se L riduce tramite funzione al problema di accettazione delle macchine di Turing (ossia, $L \leq_m \mathcal{A}_{\text{TM}}$).

Soluzione: Sappiamo che $\mathcal{A}_{\text{TM}}$ è ricorsivamente enumerabile. Sappiamo inoltre che se un linguaggio X riduce tramite funzione ad un linguaggio ricorsivamente enumerabile Y, allora X è ricorsivamente enumerabile. Infatti, detta T la macchina di Turing che costituisce la riduzione e N la macchina di Turing che riconosce Y, è immediato derivare una macchina di Turing M che riconosce X: su ogni input w, $M(w)$ esegue $T(w)$, poi esegue N sull'output di $T(w)$. Ovviamente $w \in X$ se e solo se l'istanza prodotta da $T(w)$ è in Y; dunque se $w \in X$ allora $M(w)$ accetta, mentre se $M(w)$ non accetta allora $w \notin X$. Perciò se $L \leq_m \mathcal{A}_{\text{TM}}$ allora L è ricorsivamente enumerabile.

Viceversa, supponiamo che L sia un linguaggio ricorsivamente enumerabile, dunque esista una macchina di Turing T che accetta un input w se e solo se $w \in L$ (e potrebbe non terminare altrimenti). Consideriamo la funzione calcolabile che mappa ogni istanza w nella istanza di $\mathcal{A}_{\text{TM}}$ $\langle T, w\rangle$: poiché $w \in L$ se e solo se $\langle T, w\rangle \in \mathcal{A}_{\text{TM}}$, tale funzione costituisce una riduzione tramite funzione da L a $\mathcal{A}_{\text{TM}}$, ossia $L \leq_m \mathcal{A}_{\text{TM}}$.

Esercizio 5 [6] Dimostrare che un linguaggio L è decidibile se e solo se L riduce tramite funzione al linguaggio $B = \{x \in \{0,1\}^* \mid x \text{ è la codifica in binario di un numero primo}\}$.

Soluzione: Il linguaggio B è decidibile; infatti consideriamo una macchina di Turing M che su input un numero codificato in binario x genera ogni numero intero compreso tra 2 e $x-1$, ed accetta se e solo se nessuno di essi divide esattamente x. M è quindi un decisore per B.

Sappiamo che se $X \leq_m Y$ e Y è decidibile, allora X è decidibile. Infatti, detta T la macchina di Turing che costituisce la riduzione e N la macchina di Turing che decide Y, è immediato derivare una macchina di Turing M che decide X: su ogni input w, $M(w)$ esegue $T(w)$, poi esegue N sull'output di $T(w)$. Ovviamente $w \in X$ se e solo se l'istanza prodotta da $T(w)$ è in Y; dunque se $w \in X$ allora $M(w)$ accetta, mentre se $w \notin X$ allora $M(w)$ rifiuta. Perciò se $L \leq_m B$ allora L è decidibile.

Viceversa, supponiamo che L sia decidibile, dunque esista una macchina di Turing M che decide L. Consideriamo la macchina di Turing T che su input w dapprima esegue $M(w)$; se $M(w)$ accetta allora $T(w)$ produce in output la codifica in binario del numero primo 3; se invece $M(w)$ rifiuta allora $T(w)$ produce in output la codifica in binario del numero composto 4. Perciò $T(w)$ produce un elemento in B se e solo se $w \in L$; pertanto $L \leq_m B$.

Esercizio 6 [10] Dato un grafo diretto, un *kernel* è un sottoinsieme di nodi K tale che (1) non esistono archi tra i nodi di K, e (2) per ogni nodo $u \notin K$, esiste un arco da u ad un nodo di K. Dimostrare che il problema KERNEL $= \{\langle G \rangle \mid$ il grafo diretto G contiene un kernel$\}$ è NP-completo.

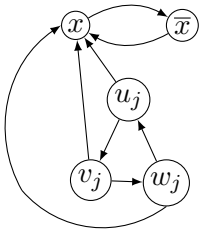
Soluzione: Il problema KERNEL appartiene alla classe NP in quanto ogni istanza $\langle G \rangle$ che fa parte del linguaggio ammette come certificato il sottoinsieme di nodi K del grafo che costituiscono il kernel: è certamente di dimensione non superiore al numero di nodi di G, e dunque alla dimensione dell'istanza, e verificare che il sottoinsieme è effettivamente un kernel può essere facilmente realizzato da un algoritmo che esegue in tempo polinomiale nella dimensione dell'istanza del problema:

M= "On input $\langle G, C \rangle$, where G is a directed graph:
1. If C is not a list of nodes $K \subseteq V(G)$, then reject
2. for each pair of elements $u, w \in K$:
    3. if $(u, w) \in E(G)$ then reject
4. for each element $x \in V(G) \setminus K$:
    5. for each element $y \in K$:
        6. if $(x, y) \in E(G)$ then continue with next element in step 3
    7. Reject, because element x is not adjacent to any node in K
8. Accept, because all elements in $V(G) \setminus K$ are adjacent to nodes in K"

È immediato stabilire che M termina in un numero di passi in $O(n^4)$ nel numero di nodi n del grafo G, e quindi è un verificatore polinomiale.

Per dimostrare che KERNEL è NP-hard utilizziamo una riduzione polinomiale da un problema NP-hard. Poiché KERNEL è un problema riguardante grafi, potrebbe sembrare opportuno ridurre da un problema su grafi come VERTEX COVER oppure INDEPENDENT SET. Tuttavia questi problemi hanno nella istanza un parametro numerico (la dimensione del sottoinsieme di nodi), che manca completamente in KERNEL. Poiché la costruzione deve tenere in considerazione tale parametro numerico in modo sostanziale, la riduzione polinomiale non sembra essere immediata. Piuttosto, cerchiamo di ridurre da un problema senza parametro numerico nelle istanza, ad esempio il problema 3SAT. Sia dunque $\Phi = \bigwedge_j C_j$ una formula booleana in CNF in cui ogni clausola C_j è la disgiunzione di esattamente tre letterali. Sia $\mathrm{Var}(\Phi)$ l'insieme delle variabili booleane utilizzate in Φ.

Il grafo diretto G_Φ risultante dalla riduzione polinomiale da Φ contiene per ogni $x \in \text{Var}(\Phi)$ due nodi $\textcircled{x}$ e $\textcircled{\overline{x}}$ connessi da due archi diretti. Il grafo inoltre contiene per ogni clausola C_j in Φ tre nodi $\textcircled{u_j}$, $\textcircled{v_j}$ e $\textcircled{w_j}$ connessi da tre archi diretti. Infine, per ogni letterale l incluso in una clausola C_j, G_Φ include tre archi dai tre nodi della clausola al nodo del letterale.



Supponiamo che la formula Φ sia soddisfacibile; dunque esiste una assegnazione di verità alle variabili in $\text{Var}(\Phi)$ che rende vera ogni clausola. Dimostriamo allora che il grafo G_Φ contiene un kernel K. Per ogni variabile x a cui è stato assegnato il valore *vero*, includiamo in K il nodo $\textcircled{x}$; analogamente, per ogni variabile y con valore *falso*, includiamo in K il nodo $\textcircled{\overline{y}}$. In totale quindi K contiene un nodo per ciascuna variabile della formula. Osserviamo che, per costruzione, non esiste alcun arco tra gli elementi in K. Infatti non esistono archi tra nodi di letterali corrispondenti a variabili differenti, ed inoltre una assegnazione di verità non può assegnare ad una variabile contemporaneamente il valore *vero* e quello *falso*. Consideriamo ora un qualsiasi nodo $\textcircled{z}$ di G_Φ non incluso in K, e distinguiamo due casi. Se il nodo $\textcircled{z}$ corrisponde ad un letterale, ossia ad una variabile diretta o negata, allora il nodo $\textcircled{\overline{z}}$ associato alla negazione del letterale deve necessariamente essere incluso in K perché l'assegnazione di verità include tutte le variabili di Φ. Pertanto, per costruzione esiste un arco da $\textcircled{z}$ (non in K) a $\textcircled{\overline{z}}$ (in K). Se invece il nodo $\textcircled{z}$ corrisponde al nodo di una clausola, allora tale clausola deve essere soddisfatta dalla assegnazione di verità; dunque deve esistere in K un nodo $\textcircled{l}$ corrispondente ad un letterale che rende vera la clausola. Per costruzione esiste un arco tra $\textcircled{z}$ (non in K) e $\textcircled{l}$ (in K). Si può concludere che K è effettivamente un kernel di G_Φ.

Supponiamo ora che nel grafo G_Φ esista un kernel K, e dimostriamo allora che la formula Φ sarebbe soddisfacibile. Supponiamo che K contenga un nodo $\textcircled{u_j}$ corrispondente ad una clausola C_j; senza perdita di generalità supponiamo che da $\textcircled{u_j}$ esca un arco verso il nodo della stessa clausola $\textcircled{v_j}$ ed entri un arco proveniente dal nodo della stessa clausola $\textcircled{w_j}$. Osserviamo che K non può includere né $\textcircled{v_j}$ né $\textcircled{w_j}$ poiché non possono esistere archi tra elementi del kernel. Pertanto, deve esistere un altro nodo $\textcircled{l}$ in K, necessariamente associato ad un letterale, che è la destinazione di un arco uscente da $\textcircled{v_j}$. Per costruzione, il nodo $\textcircled{l}$ è anche la destinazione di archi uscenti da $\textcircled{u_j}$ e $\textcircled{w_j}$. Dunque, il sottoinsieme di nodi ottenuto rimuovendo $\textcircled{u_j}$ da K è ancora un kernel per G_Φ. Possiamo dunque assumere che K contenga esclusivamente nodi associati a variabili (dirette o negate) di Φ. Consideriamo la assegnazione di verità (parziale) corrispondente ai nodi inclusi in K: se K include il nodo $\textcircled{x}$ assegniamo ad x il valore *vero*; se invece include il nodo $\textcircled{\overline{x}}$ assegniamo ad x il valore *falso*. Poiché non possono esistere archi tra i nodi del kernel, non è possibile che ad una stessa variabile debba essere assegnato sia il valore *vero* che *falso*. Consideriamo ora una qualunque clausola C_j di Φ ed un generico nodo associato alla clausola $\textcircled{u_j}$. Sappiamo che deve esistere un arco da $\textcircled{u_j}$ ad

un nodo $\textcircled{l}$ in K; tale arco è stato inserito in G_Φ perché il letterale l è incluso nella clausola C_j. L'assegnazione di verità ha assegnato al letterale l il valore *vero*, quindi la clausola C_j è soddisfatta dalla assegnazione di verità. Tutte le clausole sono pertanto soddisfatte dalla assegnazione di verità corrispondente a K, e quindi la formula Φ è soddisfacibile. Si osservi che non è necessario che K includa un nodo per ciascuna variabile di Φ: è possibile che la formula possa essere soddisfatta a prescindere dal valore di alcune variabili, e K potrebbe non includere i nodi corrispondenti a variabili il cui valore non è determinante per soddisfare Φ.

È immediato verificare che la costruzione di G_Φ da Φ può essere completata in tempo polinomiale. Pertanto KERNEL è NP-hard, e dunque NP-completo, come volevasi dimostrare.