5 Calcolabilita'

Questa sezione integra il capitolo 3 del Papadimitriou con ulteriori risultati attinenti al modello di calcolo basato sulle Turing Machines (TM), e con la riformulazione di alcuni risultati validi per TM nel contesto di un' algebra combinatoria parziale.

Il modello di calcolo delle Macchine di Turing (TM) puo' essere utilizzato sia per risolvere problemi di decisione (o accettazione), sia per calcolare funzioni (parziali) su Σ^*. Per far questo e' sufficiente aggiungere un nastro di output (per semplicita' si puo supporre che il nastro di input e' read-only, quello di output write-only, e che su tali nastri si possano trovare solo caratteri in un alfabeto Σ prefissato, mentre sui nastri di lavoro si possono usare caratteri in un alfabeto $\Gamma \supseteq \Sigma$ che varia da TM a TM). Un modo conciso di esprimere una serie di risultati fondamentali sul modello di calcolo delle TM e' l'esistenza di una pCA $X = (\Sigma^*, \cdot)$ t.c. le funzioni parziali *rappresentabili* in tale pCA sono esattamente quelle Turing-calcolabili. La definizione di $\cdot$ e' data in termini di una codifica delle TM (canoniche) in Σ^*: $y \cdot x = z$ sse y codifica una TM M e M con input x termina e produce z in output. Scriveremo M sia per indicare una TM che la sua codifica come stringa.

Indichiamo con X l'insieme Σ^* delle stringhe, fissata la codifica delle TM in X si definisce la funzione parziale $U_m: X^{m+1} \rightharpoonup X$ t.c. $U_m(M; x_1; \ldots; x_m) = y$ sse M con input $x_1; \ldots; x_m$ termina e produce y in output. (Si suppone di estendere Σ con il simbolo separatore ";"). U_m e' detta **funzione universale**.

Theorem 5.1 (TM universale) *La funzione U_m e' calcolabile, e la TM che la calcola e' detta TM universale.*

Questo risultato implica che le funzioni (unarie) rappresentabili nella struttura applicativa X sono tutte e sole le funzioni su X calcolabili da TM. Per dimostrare che X e' una pCA, ci occorre un'altra proprieta' delle funzioni calcolabili.

Theorem 5.2 (S-m-n) *Data $f: X^{m+n} \rightharpoonup X$ parziale calcolabile, esiste $g: X^m \to X$ totale calcolabile t.c. $U_n(g(\overline{x}); \overline{y}) \simeq f(\overline{x}; \overline{y})$ per ogni $\overline{x} \in X^m$ e $\overline{y} \in X^n$.*

Per dimostrare l'esistenza di K mediante il Teorema s-m-n si procede come segue: sia $f(x; y) = x$ (ovviamente clacolabile), sia g t.c. $U(g(x); y) = x$ data dal teorema s-1-1, infine sia K t.c. $U(K; x) = g(x)$ data dal teorema s-0-1. Per sostituzione e' facile verificare che $K \cdot x \cdot y = x$, cioe' $U(U(K; x); y) = x$.

Molti risultati di calcolabilita' si possono formulare in termini della pCA $(\Sigma^*, \cdot)$, ed alcuni di questi si possono vedere come casi particolari di risultati che valgono in ogni pCA. Prima di fare cio', riformuliamo alcuni concetti relativizzandoli ad una **proper pCA** (cioe' t.c. esiste $0 \in X$ che rappresenta la funzione ovunque divergente):

- una funzione parziale $f: X \to X$ e' **calcolabile** sse e' rappresentabile, cioe' $\exists e: X. \forall x: X. f(x) \simeq e\ x$
- $L \subseteq X$ e' (internamente) **decidibile** sse $\exists e: X. \forall x: X. (x \in L \supset e\ x = T) \land (x \notin L \supset e\ x = F)$ (si utilizza la codifica standard dei booleani in una pCA)
- $L \subseteq X$ e' (internamente) **semi-decidibile** sse $\exists e: X. \forall x: X. (x \in L \leftrightarrow e\ x \downarrow)$
- L_1 e' (internamente) **riducibile** a L_2 (notazione $L_1 \leq L_2$) sse esiste $f: X \to X$ totale e (internamente) calcolabile t.c. $\forall x: X. (x \in L_1 \leftrightarrow f(x) \in L_2)$.

Un risultato che vale per ogni proper pCA e' l'indecidibilita' dell'halting problem $H \triangleq \{(x, y) | x\ y \downarrow\}$, che segue dal seguente risultato (e dalla proprieta' di chiusura per riduzione dei sottoinsiemi decidibili).

Theorem 5.3 (indecidibilita' dell'halting problem per pCA) *Se $(X, \cdot)$ e' una proper pCA, allora $D \triangleq \{x \in X | x \cdot x \downarrow\}$ e'* internamente indecidibile, *cioe' non esiste $d \in X$ t.c. $d \cdot x = T$ se $x \in D$ e $d \cdot x = F$ se $x \notin D$.*

Proof La dimostrazione procede per assurdo. Supponiamo che d esista, allora possiamo definire (mediante astrazione) $e \in X$ t.c. $e\ x = if(d \cdot x, 0, I) \cdot x$. Vediamo che e non puo' appartenere ne' a D ne' al suo complementare:

- se $e \in D$ allora $e \cdot e \downarrow$, e quindi si deve avere $d \cdot e = F$, cioe' $e \notin D$ (assurdo);
- se $e \notin D$ allora $e \cdot e \uparrow$, e quindi si deve avere $d \cdot e = T$, cioe' $e \in D$ (assurdo).

∎

Proposition 5.4 *Se $L_1 \leq L_2$ e L_2 e' (semi-)decidibile, allora anche L_1 e' (semi-)decidibile.*

Anche il teorema di Rice, che asserisce l'indecidibilita' delle proprieta' *estensionali* non banali dei programmi, e' generalizzabile ad una proper pCA $(X, \cdot)$. Introduciamo un po' di terminolgia:

- $F(X) \subseteq X \rightharpoonup X$ e' l'insieme delle funzioni rappresentabili in X
- fissato $A \subseteq F(X)$, il sottoinsieme $A' \subseteq X$ e' t.c. $x \in A'$ sse la funzione rappresentata da x e' in A.

Theorem 5.5 (Rice per pCA) *Se $(X, \cdot)$ e' una* proper pCA *e $A \subset F(X)$, allora A' e' internamente decidibile sse A e' banale, cioe' $A = \emptyset$ o $A = F(X)$.*

Proof Si sfrutta la tecnica della riduzione, ovvero facciamo vedere che esiste una $f: X \to X$ totale e rappresentabile t.c. $x \in D$ sse $f(x) \in A'$. Quindi se A' fosse decidibile lo sarebbe anche D.
Supponiamo che A non sia banale e che $0 \notin A'$ (altrimenti si considera il complementare di A), inoltre sia $a \in A'$. Definiamo $f(x) = \lambda^* y.K\ (a\ y)\ (x\ x)$. Chiaramente f e' totale e rappresentabile, p.e. da $\lambda^* x, y.K\ (a\ y)\ (x\ x)$, inoltre e' una riduzione di D ad A'. Infatti, se $x \in D$ allora $f(x)$ rappresenta la stessa funzione rappresentata da a e quindi $f(x) \in A'$. Se invece $x \notin D$ allora $f(x)$ rappresenta la funzione ovunque divergente, e quindi $f(x) \notin A'$. ∎

Vi sono anche proprieta' degli insiemi decidibili e semidecidibili secondo la definizione data in termini di TM, che non sono veri per ogni proper pCA. Poiche' la pCA $(\Sigma^*, \cdot)$ corrispondente al modello di calcolo delle TM e' isomorfa alla pCA $(N, \cdot)$ di Kleene (ogni biiezione *effettiva* tra N e Σ^* fornisce un isomorfismo), formuleremo tali proprieta' nella pCA che ci torna piu' conveniente.

Definition 5.6 *Fissata una pCA $(X, \cdot)$ ed una codifica delle coppie, per ogni $L \subseteq X$ definiamo*

$$\exists(L) = \{x: X | \exists y: X.(Pair\ x\ y) \in L\}$$

tale sottoinsieme e' detto **la quantificazione esistenziale** *di L. In modo analogo si definisce la quantificazione universale di L.*

Proposition 5.7 *Nella pCA $(\Sigma^*, \cdot)$ il* **predicato T di Kleene**, *$T(M; x; y) \stackrel{\Delta}{\iff}$ "y e' la computazione della TM M con input x", e' decidibile.*

Il predicato di Kleene e' utile per dimostrare alcuni dei seguenti risultati.

Theorem 5.8 *Nella pCA $(\Sigma^*, \cdot)$ valgono i seguenti fatti*

- *L e' decidibile sse L ed il suo complementare $\overline{L}$ sono; semidecidibili.*
- *L e' semidecidibile sse $L = \exists(R)$ per qualche R decidibile.*

Proof Le direzioni difficili di tali caratterizzazioni si dimostrano dando degli algoritmi che usano il predicato di Kleene.

- supponiamo che la TM $M1$ accetta L e la TM $M2$ accetta $\overline{L}$ allora una TM M che decide L e' data da

  ```
  M(x) = for y string do
            if T(M1;x;y) and y accepting then accept
            if T(M2;x;y) and y accepting then reject
  ```

  prima o poi troveremo un y per cui uno dei due if e' vero.
- supponiamo che M accetta L, allora definiamo

  ```
  R(x,y) = T(M;x;y) and y accepting
  ```

  chiaramente R e' decidibile e $L = \exists(R)$.

∎

Theorem 5.9 *Nella pCA $(\Sigma^*, \cdot)$ i sottoinsiemi semidecidibili sono chiusi per intersezione, unione e quantificazione esistenziale.*

Proof Siano $M1$ ed $M2$ TM che accettano i linguaggi L_1 ed L_2 allora

- il linguaggio $L_1 \cap L_2$ e' accettato dalla TM `M(x) = if M1(x) then if M2(x) the accept`
- il linguaggio $L_1 \cup L_2$ e' accettato dalla TM

```
M(x) = for y string do
         if T(M1;x;y) and y accepting then accept
         if T(M2;x;y) and y accepting then accept
```

una definizione alternativa di M e' data dalla *composizione parallela* di $M1$ ed $M2$

Sia M una TM che accetta L, allora $\exists(L)$ e' accettato dalla TM

```
M('x) = for (y,z) string do
         if T(M;(x,y);z) and z accepting then accept
```

si osservi che si ha un unico ciclo for, che enumera tutte le (codifiche) di coppie di stringhe. ∎

E' immediato vedere che in ogni (proper) pCA i sottoinsiemi decidibili sono chiusi per intersezione, unione e complemento (mentre quelli semidecidibili sono chiusi per intersezione).

Theorem 5.10 *Nella pCA* $(N, \cdot)$ *le seguenti asserzioni su* $L \subseteq N$ *sono equivalenti*

- *L e' semidecidibile, cioe' il dominio di una funzione parziale calcolabile f*
- *L e' l'immagine di una funzione parziale calcolabile g*
- *L e' vuoto o e' l'immagine di una funzione totale calcolabile h.*

Proof Facciamo vedere una catena di implicazioni. Data f definiamo `g(x)=if f(x) defined then x`.
Data g calcolata dalla TM M e $z \in L$ definiamo `h(x;y)=if T(M;x,y) and y accepting then g(x) else z`.
Data h definiamo `f(x)=for y string do if h(y)=x then return(x)`. ∎

Theorem 5.11 *Nella pCA* $(N, \cdot)$ *le seguenti asserzioni su* $L \subseteq N$ *sono equivalenti*

- *L e' decidibile, cioe' la funzione caratteristica f e' calcolabile*
- *L e' finito o e' l'immagine di una funzione totale calcolabile h strettamente crescente.*

Proof Supponiamo che L sia infinito, e quindi la funzione caratteristica f soddisfa la proprieta' $\forall m{:}\, N.\exists n > m.f(n) = 1$. Data f calcolabile definiamo

```
h(n) = m:=0; x:=0;
       while m<n or f(x)=0 do
         if f(x)=1 then m:=m+1
         x:=x+1
       return(x)
```

in altre parole se ordiniamo gli elementi di L in modo crescente $x_0 < x_1 < \ldots$, allora $h(n) = x_n$. Viceversa data h definiamo

```
f(x) = for n:=0 to x do
         if h(n)=x then return(1)
       return(0)
```

∎

Mentre la tecnica della riducibilita' serve essenzialemnte a dimostrare risultati negativi (non decidibilita' e non semidecidibilita' di problemi), le proprieta' di chiusura servono soprattutto a dimostrare la decidibilita' e semidecidibilita' di un problema, in alternativa alla descrizione di una TM (o di un algoritmo ad alto livello).

6 Complessita'

Questa sezione elenca i risultati visti a lezione. In alcuni casi i risultati vengono enunciati in modo leggermente diverso dal libro di testo, evitando riferimenti espliciti al modello di computazione.

- Modelli di calcolo. Abbiamo introdotto i seguenti modelli di calcolo: le TM deterministiche (DTM), the TM non-deterministiche (NDTM),e le Random Access Machines (RAM). Per le TM abbiamo introdotto alcune varianti: input read-only, output write-only, limitazione sul numero di nastri (di lavoro).
- Risorse di calcolo. Per le TM abbiamo introdotto la risorsa tempo (misurata in numero di passi di computazione) e la risorsa spazio (misurato in numero di celle visitate sui nastri di lavoro). Per le RAM abbiamo considerato solo la risorsa tempo.

I modelli di computazione formalizzano (in modi diversi) la nozione di *algoritmo*. Quello che si vuole studiare in complessita' computazionale non e' la complessita' degli algoritmi, ma quella dei *problemi*. E' quindi necessario fissare una corrispondenza del tipo "l'algoritmo M risolve il problema L". Abbiamo introdotto le seguenti corrispondenze:

- un linguaggio $L \subseteq \Sigma^*$ **e' deciso** da una DTM M con alfabeto di input Σ
- un linguaggio $L \subseteq \Sigma^*$ **e' accettato** da una DTM M con alfabeto di input Σ
- un linguaggio $L \subseteq \Sigma^*$ **e' deciso** da una NDTM M con alfabeto di input Σ
- una funzione parziale $f: \Sigma^* \rightharpoonup \Sigma^*$ **e' calcolata** da una DTM M con alfabeto di input/output Σ
- una funzione parziale $f: Z^* \rightharpoonup Z$ **e' calcolata** da una RAM M.

Fissate la nozione di algoritmo (mediante la scelta di un modello di calcolo), una corrispondenza tra problemi ed algoritmi, ed una risorsa di calcolo (per il modello prescelto), resta definita una *classe di complessita'* (parametrizzata rispetto ad un upper-bound sulla risorsa di calcolo), ovvero l'insieme dei problemi L che sono *risolubili* con un algoritmo M che usa la risorsa entro il limite consentito. Le classi di complessita' comunemente considerate sono:

- $L \in TIME(f) \subseteq \Sigma^*$ sse esiste una DTM che decide L e che con input di lunghezza n lavora in tempo $\leq f(n)$ (in genere si suppone che $f: N \to N$ sia una funzione almeno lineare).
- $L \in NTIME(f) \subseteq \Sigma^*$ sse esiste una NDTM che decide L e che con input di lunghezza n lavora in tempo $\leq f(n)$, la differenza essenziale rispetto alla classe $TIME(f)$ e' l'uso del modello di calcolo non-deterministico, che richiede adattamenti sia alla definizione di M decide L che al calcolo della risorsa tempo.
- $L \in SPACE(f) \subseteq \Sigma^*$ sse esiste una DTM con input read-only che decide L e che con input di lunghezza n usa spazio di lavoro $\leq f(n)$.
- $L \in NSPACE(f) \subseteq \Sigma^*$ sse esiste una NDTM con input read-only che decide L e che con input di lunghezza n usa spazio di lavoro $\leq f(n)$.
- $h \in TIME(f) \subseteq \Sigma^* \to \Sigma^*$ sse esiste un DTM con che calcola h e che con input di lunghezza n lavora in tempo $\leq f(n)$.

  $h \in SPACE(f) \subseteq \Sigma^* \to \Sigma^*$ sse esiste un DTM con input read-only ed output write-only che calcola h e che con input di lunghezza n usa spazio di lavoro $\leq f(n)$.

  $h \in TIME^{RAM}(f) \subseteq Z^* \to Z$ sse esiste una RAM che calcola h e che con input di lunghezza n (usando la codifica binaria) lavora in tempo $\leq f(n)$. In questo caso e' molto importante precisare il tipo di operazioni aritmentiche disponibili su una RAM (si consente solo somma, sottrazione e divisione per 2).

In alcuni casi si vuole imporre delle limitazioni sul numero di nastri (di lavoro) di una TM, scriveremo $TIME_k(f)$ (e similmente per le altri classe di complessita' basate sul modello di calcolo delle TM) per indicare che si usano TM con $\leq k$ nastri (di lavoro).
Riportiamo i risultati fondamentali sulle classi di complessita' basate su modello di calcolo delle TM.

- riduzione a TM con un nastro (di lavoro).

  $TIME(f) \subseteq TIME_1(O(f^2))$ [se f e' almeno lineare], un risultato analogo vale per $NTIME$.

  $SPACE(f) \subseteq SPACE_1(f)$, un risultato analogo vale per $NSPACE$.

  L'idea e' di rappresentare i k nastri (di lavoro) di M con k tracce del nastro (di lavoro) di M', cambiando in modo opportuno l'afabeto di lavoro. A parte una fase di inizializzazione, M' richiede $O(f)$ mosse per simulare una mossa di M.
- $TIME_k(f) \subseteq SPACE_k(f)$ [se f e' almeno lineare].

- teorema di speed-up.

  $TIME(f) \subseteq TIME(g)$ se $f \in O(g)$ [e $g(n)/n \to \infty$], un risultato analogo vale per $NTIME$. L'idea e di simulare molti passi di M con pochi passi di M' usando un alfabeto di lavoro molto piu' grande.

- teorema di tape compression.

  $SPACE(f) \subseteq SPACE(g)$ se $f \in O(g)$, un risultato analogo vale anche per $NSPACE$. L'idea e' di rappresentare una sequenza di caratteri di lavoro di M con un carettere di lavoro di M'. (Al contrario del teorema di speed-up, il teorema di tape compression vale anche per classi di complessita' di funzioni.)

- classi naturali di complessita'

  - $L = SPACE(\log n)$
  - $NL = NSPACE(\log n)$
  - $P = \cup_{k \in N} TIME(n^k)$
  - $NP = \cup_{k \in N} NTIME(n^k)$
  - $PSPACE = \cup_{k \in N} SPACE(n^k)$
  - $NPSPACE = \cup_{k \in N} NSPACE(n^k)$
  - $EXP = \cup_{k \in N} TIME(2^{(n^k)})$

- relazioni tra RAM e TM (e robustezza della classe P)

  $TIME^{TM}(f) \subseteq TIME^{RAM}(O(f))$ se f e' almeno lineare [modulo biiezione tra stringhe ed interi].

  $TIME^{RAM}(f) \subseteq TIME^{TM}(O(f^3))$ se f e' almeno lineare [modulo biiezione tra stringhe ed interi]. L'idea e' di usare un nastro della TM come log file, in cui si riportano in ordine cronologico tutte le modifiche effettuate allo stato della RAM, e che viene consultato tutte le volete che si deve simulare un accesso ad un registro della memoria (quelli di input si trovano invece sul nastro di input read-only). Tale nastro contiene al massimo $f(n)$ records.

  Per dimostrare che ogni record ha dimensione $O(f)$ si usa il seguente Lemma sulle RAM: dopo t passi i registri di memoria contengono interi la cui rappresentazione e' limitata da una funzione lineare in t e la dimensione degli input. (tale Lemma non vale se si include la moltiplicazione tra le operazioni aritmetiche di una RAM).

- teorema di gerarchia spazio

  $SPACE(g) \not\subseteq SPACE(f)$, se $g \notin O(f)$ e g proper (e $g \geq \log$). Dove proper vuol dire che esiste una TM con input read-only che per ogni input di dimensione n usa esattamente $g(n)$ celle sul nastro di lavoro. Il controesempio a $SPACE(g) \subseteq SPACE(f)$ e' dato dal linguaggio

  $$D_g^{space} = \{w | \exists M, k. w = M; 1^k \wedge\ U(M; w) \text{ rigetta in spazio } \leq g(|w|)\}$$

- teorema di gerarchia tempo

  $TIME(g) \not\subseteq TIME(f)$, se $g \notin O(f^2)$ e g proper. Dove proper vuol dire che esiste una TM con che per ogni input di dimensione n termina in esattamente $g(n)$ passi. Il controesempio a $TIME(g) \subseteq TIME(f)$ e' dato dal linguaggio

  $$D_g^{time} = \{w | \exists M, k. w = M; 1^k \wedge\ U(M; w) \text{ rigetta in tempo } \leq g(|w|)\}$$

- relazioni di inclusioni tra classi di complessita'

  - $TIME(f) \subseteq NTIME(f)$ e $SPACE(f) \subseteq NSPACE(f)$. Dimostrazione immediata.
  - $NTIME(f) \subseteq SPACE(f)$. Si esaminano tutte le possibili computazioni della NDTM M che lavora intempo f usando un contatore fino a $d^{f(n)}$, dove d e' il grado di non-determinismo di M, per risolvere le scelte non-deterministiche.
  - $NSPACE(f) \subseteq \cup_k TIME(k^{(\log n)+f(n)})$. Si usa il reachability method. Cioe' si riformula il problema di determinare l'accettazione di una stringa x da parte di una NDTM M con nastro read-only che lavora in spazio f, al problema dell'esistenza di un cammino dalla configurazione iniziale alla configurazione finale (canonica) nel grafo delle ID di M di dimensione $\leq f(n)$.
  - teorema di Savitch

    $NSPACE(f) \subseteq SPACE(f^2)$ (se $f \geq \log$). Si usa il reachability method.

- relazioni di inclusione tra classi naturali di complessita'

  $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXP$, le uniche inclusioni strette (che si riescono a dimostrare usando i teoremi di gerarchia) sono $NL \subset PSPACE$ e $P \subset EXP$.

- riducibilita' logspace $\leq_L$ e polytime $\leq_P$

  Proprieta' riflessiva e transitiva delle nozioni di riducibilita' (per dimostrare la transitivita' di $\leq_L$ occorre usare un algoritmo demand driven per implementare la composizione delle funzioni di riduzione log-space).

  Chiusura delle classi naturali di complessita' rispetto a $\leq_L$, cioe' $L_1 \leq_L L_2 \in C$ implica $L_1 \in C$. Un risultato analogo vale per $\leq_P$ se $C \supseteq P$.

- Problemi C-completi e C-hard (rispetto ad una riduzione $\leq$)

  H e' RE-completo per $\leq_L$ (addirittura per riduzioni in tempo lineare). Non esistono problemi R-completi per $\leq_P$. Problemi C-completi per $\leq_L$ per le classi naturali di complessita', ottenuti come varuanti dell'halting problem.

- proprieta' di chiusura delle classi di complessita' rispetto ad operazioni logiche

  Chiusura delle classi di complessita' per intersezione. Chiusura delle classi $TIME(f)$ e $SPACE(f)$ per complementazione (se $f \geq \log$). Chiusura delle classi $NTIME(f)$ e $NSPACE(f)$ per disgiunzione (se $f \geq \log$). Le dimostrazioni sono adattamenti delle analogo proprieta' di chiusura di R ed RE.

  Teorema di Immerman: chiusura di $NSPACE(f)$ per complementazione (se $f \geq \log$ e proper).

- proprieta' di chiusura delle classi naturali di complessita' rispetto a quantificazione limitata

  L, NL e P sono chiuse per qunatificazione esistenziale log-limitata, cioe' $L \in P$ implica $\exists_{\log}(L) defeq \{x | \exists y.\langle x, y\rangle \in L \wedge |y| \leq \log |x|\}$. NP, $PSPACE$ ed EXP sono chiuse per quantificazione esistenziale poly-limitata.

- caratterizzazione di NP in termini di P.

  Un linguaggio sta in NP sse e' la qunatificazione esistenziale poly-limitata di un linguaggio in P. Data una NDTM M che lavora in polytime L_M si ottiene come quantificazione esistenziale poly-limitata del linguaggio $\{\langle x, y\rangle | y$ codifica una computazione accettante di M con input $x\}$.

  Questo risultato si puo' riformulare dicendo che i problemi NP si possono vedere come dei problemi di esistenza di una soluzione in uno spazio di possibili soluzioni di cardinalita' esponenziale nella dimensione dell'istanza del problema, pero' data una soluzione potenziale e' facile erificare se e' una soluzione accettabile.

- esempi di problemi completi.

  CVP (Circuit Value Problem) e' P-completo. SAT (Satisfability of boolean formulas in CNF) e' NP-completo. Dimostrazione mediante codifica di tabelle di computazione.

  REACHABILITY e' NL-completo. QBF (verita' di una Quantified Boolean Formula) e' $PSPACE$-completo.

- Altri esempi di problemi NP-completi (dimostrazione per riduzione).