Tutorato di Automi e Linguaggi Formali

Soluzioni Homework 10: Riducibilità mediante Funzione e Classe P

Gabriel Rovesti
Corso di Laurea in Informatica – Università degli Studi di Padova

Tutorato 10 – 26-05-2025

1 Riducibilità mediante Funzione

Esercizio 1. Sia $HALF_{TM} = \{\langle M \rangle \mid M$ è una TM che accetta almeno la metà delle stringhe di lunghezza n per qualche $n \geq 1\}$.

a) Dimostrare che $A_{TM} \leq_m HALF_{TM}$ fornendo una funzione di riduzione esplicita f tale che per ogni coppia $\langle M, w \rangle$, si ha $\langle M, w \rangle \in A_{TM}$ se e solo se $f(\langle M, w \rangle) \in HALF_{TM}$.

b) Utilizzare la riduzione precedente per dimostrare che $HALF_{TM}$ è indecidibile.

c) Analizzare se $HALF_{TM}$ è Turing-riconoscibile o co-Turing-riconoscibile. Giustificare la risposta.

Soluzione. a) Riduzione da A_{TM} a $HALF_{TM}$

Teorema 1. $A_{TM} \leq_m HALF_{TM}$.

Proof. Definiamo una funzione di riduzione $f : \{\langle M, w \rangle\} \to \{\langle M' \rangle\}$ come segue: data una coppia $\langle M, w \rangle$, costruiamo una nuova macchina di Turing M' che opera così:
Su input x:

1. Se $|x| = 1$, M' simula M su input w.
2. Se M accetta w, allora M' accetta x.
3. Se M non accetta w (rifiuta o non si ferma), allora M' rifiuta x.
4. Se $|x| \neq 1$, allora M' rifiuta immediatamente x.

Verifichiamo che questa è una riduzione valida:

Caso 1: $\langle M, w\rangle \in A_{TM}$ (cioè M accetta w). Allora M' accetta tutte le stringhe di lunghezza 1. Poiché esistono esattamente 2 stringhe di lunghezza 1 su un alfabeto binario $\{0,1\}$, M' accetta $\frac{2}{2} = 1 \geq \frac{1}{2}$ delle stringhe di lunghezza 1. Quindi $\langle M'\rangle \in HALF_{TM}$.

Caso 2: $\langle M, w\rangle \notin A_{TM}$ (cioè M non accetta w). Allora M' non accetta nessuna stringa di alcuna lunghezza. Quindi M' non accetta almeno la metà delle stringhe di alcuna lunghezza $n \geq 1$, e $\langle M'\rangle \notin HALF_{TM}$.

Pertanto, $\langle M, w\rangle \in A_{TM}$ se e solo se $f(\langle M, w\rangle) \in HALF_{TM}$. □

b) Indecidibilità di $HALF_{TM}$

Teorema 2. *$HALF_{TM}$ è indecidibile.*

Proof. Abbiamo dimostrato che $A_{TM} \leq_m HALF_{TM}$. Poiché A_{TM} è indecidibile, per il teorema della riducibilità mediante funzione, anche $HALF_{TM}$ è indecidibile. □

c) Turing-riconoscibilità di $HALF_{TM}$

Teorema 3. *$HALF_{TM}$ è Turing-riconoscibile ma non co-Turing-riconoscibile.*

Proof. **Turing-riconoscibilità:** Costruiamo una macchina R che riconosce $HALF_{TM}$:

1. Su input $\langle M\rangle$, per ogni $n = 1, 2, 3, \ldots$:
2. Enumera tutte le stringhe di lunghezza n.
3. Simula M su ciascuna stringa per al massimo n passi.
4. Se M accetta almeno la metà delle stringhe di lunghezza n entro n passi, accetta $\langle M\rangle$.

Se M accetta almeno la metà delle stringhe di qualche lunghezza n, R eventualmente lo scoprirà e accetterà.

Non co-Turing-riconoscibilità: Se $HALF_{TM}$ fosse co-Turing-riconoscibile, allora poiché è anche Turing-riconoscibile, sarebbe decidibile. Ma abbiamo dimostrato che è indecidibile, quindi non può essere co-Turing-riconoscibile. □

Esercizio 2. Consideriamo il problema $REGULAR_{TM} = \{\langle M\rangle \mid M$ è una TM tale che $L(M)$ è un linguaggio regolare$\}$.

a) Costruire una riduzione da A_{TM} a $\overline{REGULAR_{TM}}$ (il complemento di $REGULAR_{TM}$). Specificare la funzione di riduzione e dimostrare la sua correttezza.

b) Utilizzare il teorema di Rice per fornire una dimostrazione alternativa dell'indecidibilità di $REGULAR_{TM}$.

c) Confrontare le due dimostrazioni: quale fornisce più informazioni sulla natura computazionale del problema?

Soluzione. a) Riduzione da A_{TM} a $\overline{REGULAR_{TM}}$

Teorema 4. $A_{TM} \leq_m \overline{REGULAR_{TM}}$.

Proof. Definiamo una funzione di riduzione $f : \{\langle M, w\rangle\} \to \{\langle M'\rangle\}$. Data una coppia $\langle M, w\rangle$, costruiamo M' che opera come segue:

Su input x:

1. Se x non ha la forma 0^n1^n per qualche $n \geq 0$, M' rifiuta.
2. Se $x = 0^n1^n$ per qualche $n \geq 0$, M' simula M su input w.
3. Se M accetta w, allora M' accetta x.
4. Se M non accetta w, allora M' rifiuta x.

Verifichiamo la correttezza:

Caso 1: $\langle M, w\rangle \in A_{TM}$ (cioè M accetta w). Allora $L(M') = \{0^n1^n \mid n \geq 0\}$, che è un linguaggio context-free ma non regolare. Quindi $\langle M'\rangle \in \overline{REGULAR_{TM}}$.

Caso 2: $\langle M, w\rangle \notin A_{TM}$ (cioè M non accetta w). Allora $L(M') = \emptyset$, che è regolare. Quindi $\langle M'\rangle \notin \overline{REGULAR_{TM}}$.

Pertanto, $\langle M, w\rangle \in A_{TM}$ se e solo se $f(\langle M, w\rangle) \in \overline{REGULAR_{TM}}$. □

b) Dimostrazione via Teorema di Rice

Teorema 5. *$REGULAR_{TM}$ è indecidibile per il teorema di Rice.*

Proof. Il teorema di Rice afferma che ogni proprietà non banale della funzione calcolata da una macchina di Turing è indecidibile.

La proprietà "il linguaggio riconosciuto è regolare" è:

- **Non banale:** Esistono macchine che riconoscono linguaggi regolari (es. $L = \{0\}^*$) ed esistono macchine che riconoscono linguaggi non regolari (es. $L = \{0^n1^n \mid n \geq 0\}$).
- **Proprietà del linguaggio riconosciuto:** Dipende solo dal linguaggio $L(M)$ e non dalla particolare implementazione di M.

Per il teorema di Rice, $REGULAR_{TM}$ è indecidibile. □

c) Confronto delle dimostrazioni

- **Riduzione esplicita:** Fornisce una costruzione concreta che mostra come trasformare istanze di A_{TM} in istanze di $\overline{REGULAR_{TM}}$. Questo approccio è più costruttivo e mostra esattamente come l'indecidibilità si trasferisce tra i problemi.
- **Teorema di Rice:** È più generale e immediato, ma meno informativo sulla natura specifica del problema. Dice che il problema è indecidibile perché rientra in una classe generale di problemi indecidibili, ma non spiega il "perché" computazionale specifico.
- **Informazioni computazionali:** La riduzione esplicita fornisce più insight sulla relazione tra accettazione di stringhe specifiche e regolarità dei linguaggi, mostrando come la decidibilità dell'accettazione implichi la decidibilità della regolarità.

Esercizio 3. Sia $PREFIX_{TM} = \{\langle M_1, M_2\rangle \mid M_1, M_2\{$ sono TM tali che $L(M_1)$ è un prefisso proprio di $L(M_2)\}$, dove A è un prefisso proprio di B se $A \subset B$ e per ogni $w \in A$, nessuna estensione propria di w appartiene ad A.

a) Dimostrare che $E_{TM} \leq_m PREFIX_{TM}$ costruendo una riduzione appropriata.

b) Dimostrare che $PREFIX_{TM}$ è indecidibile utilizzando la riduzione precedente.

c) Spiegare perché questo risultato è significativo nel contesto dell'analisi delle relazioni tra linguaggi.

Soluzione. a) Riduzione da E_{TM} a $PREFIX_{TM}$

Teorema 6. *$E_{TM} \leq_m PREFIX_{TM}$.*

Proof. Definiamo una funzione di riduzione $f : \{\langle M\rangle\} \to \{\langle M_1, M_2\rangle\}$. Data una macchina M, costruiamo due macchine M_1 e M_2:

Macchina M_1: Su input x:

1. Simula M su input x.
2. Se M accetta x, allora M_1 accetta x.
3. Se M non accetta x, allora M_1 rifiuta x.

Quindi $L(M_1) = L(M)$.

Macchina M_2: Su input x:

1. Se $x = \varepsilon$ (stringa vuota), M_2 accetta.
2. Altrimenti, M_2 simula M_1 su input x e fa la stessa cosa.

Quindi $L(M_2) = L(M) \cup \{\varepsilon\}$.

Verifichiamo la correttezza:

Caso 1: $\langle M\rangle \in E_{TM}$ (cioè $L(M) = \emptyset$). Allora $L(M_1) = \emptyset$ e $L(M_2) = \{\varepsilon\}$. Abbiamo $L(M_1) \subset L(M_2)$ e $L(M_1)$ è un prefisso proprio di $L(M_2)$ (ogni stringa in $\emptyset$ è prefisso di stringhe in $\{\varepsilon\}$, vacuamente vero). Quindi $\langle M_1, M_2\rangle \in PREFIX_{TM}$.

Caso 2: $\langle M\rangle \notin E_{TM}$ (cioè $L(M) \neq \emptyset$). Se $\varepsilon \in L(M)$, allora $L(M_1) = L(M_2)$, quindi $L(M_1)$ non è un prefisso proprio di $L(M_2)$. Se $\varepsilon \notin L(M)$, allora $L(M_1) \neq \emptyset$ e $L(M_2) = L(M) \cup \{\varepsilon\}$. In questo caso, $L(M_1)$ non è un prefisso proprio di $L(M_2)$ perché contiene stringhe non vuote che non sono prefissi di ε. In entrambi i sottocasi, $\langle M_1, M_2\rangle \notin PREFIX_{TM}$.

Pertanto, $\langle M\rangle \in E_{TM}$ se e solo se $f(\langle M\rangle) \in PREFIX_{TM}$. □

b) Indecidibilità di $PREFIX_{TM}$

Teorema 7. *$PREFIX_{TM}$ è indecidibile.*

Proof. Abbiamo dimostrato che $E_{TM} \leq_m PREFIX_{TM}$. Poiché E_{TM} è indecidibile, per il teorema della riducibilità mediante funzione, anche $PREFIX_{TM}$ è indecidibile. □

c) Significatività del risultato
Questo risultato è significativo per diversi motivi:

1. **Relazioni strutturali tra linguaggi:** Dimostra che anche relazioni apparentemente semplici tra linguaggi (come essere un prefisso proprio) possono essere indecidibili quando applicate a linguaggi riconosciuti da macchine di Turing arbitrarie.

2. **Complessità delle proprietà relazionali:** Mentre molti risultati di indecidibilità riguardano proprietà intrinseche di singoli linguaggi, questo mostra che anche le relazioni tra coppie di linguaggi possono essere indecidibili.

3. **Implicazioni per la verifica automatica:** Nel contesto della verifica del software, questo risultato implica che non possiamo decidere automaticamente se il comportamento di un programma è un "sotto-caso" del comportamento di un altro programma.

4. **Gerarchia di Chomsky:** Il risultato è interessante anche nel contesto della gerarchia di Chomsky, dove le relazioni di inclusione tra classi di linguaggi sono fondamentali. Mostra che tali relazioni, quando coinvolgono linguaggi ricorsivamente enumerabili arbitrari, sono intrattabili.

2 Classe P e Algoritmi Polinomiali

Esercizio 4. Sia $MODEXP = \{\langle a, b, c, p\rangle \mid a, b, c, p$ sono numeri interi positivi in rappresentazione binaria tali che $a^b \equiv c \pmod{p}\}$.

a) Dimostrare che $MODEXP \in P$ fornendo un algoritmo polinomiale esplicito. Analizzare attentamente la complessità temporale considerando la rappresentazione binaria degli input.

b) Spiegare perché un approccio naive di calcolare a^b e poi ridurre modulo p non è polinomiale nella dimensione dell'input.

c) Descrivere l'algoritmo di esponenziazione modulare veloce e dimostrare formalmente che la sua complessità temporale è $O((\log b)^3)$.

Soluzione. a) Dimostrazione che $MODEXP \in P$

Teorema 8. *$MODEXP \in P$.*

Proof. Presentiamo un algoritmo polinomiale per $MODEXP$:
Algoritmo:

1. Input: $\langle a, b, c, p\rangle$ dove tutti sono rappresentati in binario

2. Calcola a^b mod p usando l'esponenziazione modulare veloce

3. Confronta il risultato con c

4. Accetta se sono uguali, rifiuta altrimenti

Analisi della complessità: Sia n la dimensione totale dell'input, cioè $n = \log a + \log b + \log c + \log p$.

- Il passo 2 (esponenziazione modulare) richiede $O((\log b) \cdot (\log p)^2)$ tempo.
- Il passo 3 (confronto) richiede $O(\log p)$ tempo.
- Poiché $\log b \leq n$ e $\log p \leq n$, la complessità totale è $O(n^3)$.

Quindi l'algoritmo è polinomiale nella dimensione dell'input. □

b) Inefficienza dell'approccio naive
L'approccio naive di calcolare prima a^b e poi ridurre modulo p non è polinomiale perché:

- Il numero a^b può avere fino a $b \cdot \log a$ bit.
- Se b è rappresentato in binario con k bit, allora b può essere grande quanto 2^k.
- Quindi a^b può avere fino a $2^k \cdot \log a$ bit, che è esponenziale nella dimensione dell'input k.
- Calcolare un numero così grande richiede tempo esponenziale.
- Anche se riuscissimo a calcolarlo, l'operazione di riduzione modulo richiederebbe tempo proporzionale alla dimensione del numero, che è ancora esponenziale.

c) Algoritmo di esponenziazione modulare veloce
Algoritmo:

Listing 1: Esponenziazione Modulare Veloce

```
1 def mod_exp(a, b, p):
2     result = 1
3     a = a % p
4     while b > 0:
5         if b % 2 == 1:
6             result = (result * a) % p
7         b = b // 2
8         a = (a * a) % p
9     return result
```

Teorema 9. *L'algoritmo di esponenziazione modulare veloce ha complessità temporale* $O((\log b)^3)$.

Proof. **Analisi dettagliata:**

1. **Numero di iterazioni:** Il ciclo while esegue $O(\log b)$ iterazioni, poiché b viene diviso per 2 ad ogni iterazione.

2. **Costo per iterazione:** Ogni iterazione involve:
   - Test $b\%2$: $O(\log b)$ tempo
   - Moltiplicazione $(result \cdot a)\%p$: $O((\log p)^2)$ tempo
   - Divisione $b//2$: $O(\log b)$ tempo
   - Moltiplicazione $(a \cdot a)\%p$: $O((\log p)^2)$ tempo
3. **Dimensione dei numeri:** Durante l'algoritmo:
   - $result < p$, quindi ha al massimo $\log p$ bit
   - $a < p$, quindi ha al massimo $\log p$ bit
   - Le moltiplicazioni coinvolgono numeri di al massimo $\log p$ bit
4. **Complessità totale:**
   - Costo per iterazione: $O((\log p)^2)$
   - Numero di iterazioni: $O(\log b)$
   - Complessità totale: $O(\log b \cdot (\log p)^2)$
5. Nel caso peggiore, $\log p = O(\log b)$ (quando p e b hanno dimensioni simili), quindi la complessità diventa $O((\log b)^3)$.

□

Esercizio 5. Considerare il problema $COMPOSITE = \{n \mid n$ è un numero composto rappresentato in

a) Fornire un algoritmo deterministico polinomiale per $COMPOSITE$ utilizzando il test di primalità AKS o una variante deterministica appropriata. Analizzare la complessità temporale.

b) Confrontare questo approccio con un algoritmo probabilistico come il test di Miller-Rabin. Discutere i vantaggi e svantaggi di ciascun approccio.

c) Spiegare come la dimostrazione che $COMPOSITE \in P$ (e quindi $PRIMES \in P$) ha impatti significativi sulla teoria della complessità computazionale.

Soluzione. a) Algoritmo deterministico per $COMPOSITE$

Teorema 10. *$COMPOSITE \in P$ usando l'algoritmo AKS.*

Proof. L'algoritmo AKS (Agrawal-Kayal-Saxena) del 2002 fornisce un test di primalità deterministico polinomiale.

Algoritmo per $COMPOSITE$**:**

1. Input: n in rappresentazione binaria
2. Esegui l'algoritmo AKS per testare se n è primo
3. Se AKS determina che n è primo, rifiuta

4. Se AKS determina che n è composto, accetta

Analisi della complessità:

- L'algoritmo AKS originale ha complessità $O((\log n)^{12})$ (versioni migliorate arrivano a $O((\log n)^6)$)
- Poiché n è l'input e $\log n$ è la dimensione dell'input in bit, la complessità è polinomiale nella dimensione dell'input
- Quindi $COMPOSITE \in P$

□

Esercizio 6. Sia $BIPARTITE = \{\langle G \rangle \mid G \text{ è un grafo non diretto bipartito}\}$.

a) Dimostrare che $BIPARTITE \in P$ fornendo un algoritmo basato sulla colorazione a 2 colori. Analizzare la complessità temporale in termini del numero di vertici $|V|$ e archi $|E|$.

Soluzione. a) Algoritmo basato sulla colorazione a 2 colori

Teorema 11. $BIPARTITE \in P$.

Proof. **Algoritmo di colorazione:**

1. Input: Grafo $G = (V, E)$
2. Inizializza un array $color[v]$ per ogni $v \in V$ con valore "non colorato"
3. Per ogni componente connessa del grafo:
   (a) Scegli un vertice s non ancora colorato
   (b) Colora s con colore 0
   (c) Usa DFS/BFS partendo da s:
   - Per ogni vertice u visitato con colore c
   - Per ogni vicino v di u:
     - Se v non è colorato, colora v con colore $1 - c$
     - Se v è già colorato con colore c, restituisci "non bipartito"
4. Se tutte le componenti sono colorate con successo, restituisci "bipartito"

Analisi della complessità:

- Ogni vertice viene visitato al massimo una volta: $O(|V|)$
- Ogni arco viene esaminato al massimo due volte (una per ogni estremo): $O(|E|)$
- Complessità totale: $O(|V| + |E|)$

Quindi $BIPARTITE \in P$. □

3 Problemi Avanzati su P e Riducibilità

Esercizio 7. Sia $EQDFA = \{\langle A, B\rangle \mid A, B \text{ DFA con } L(A) = L(B)\}$.

a) Dimostrare che $EQDFA \in P$ tramite minimizzazione.

b) Dare un algoritmo col prodotto cartesiano.

c) Commentare la difficoltà di $EQNFA$.

Soluzione. **a) Minimizzazione**. Minimizziamo A e B in $O(n \log n)$ (Hopcroft): otteniamo A_{min} e B_{min}. Due DFA minimi riconoscono lo stesso linguaggio sse sono isomorfi, verificabile in $O(n)$. Quindi $EQDFA \in P$.

b) Prodotto cartesiano. Costruiamo C con stati $Q_A \times Q_B$ e finali $F_A \times (Q_B \setminus F_B) \cup (Q_A \setminus F_A) \times F_B$. $G = L(C)$ è vuoto sse $L(A) = L(B)$. Il test di vuotezza è una ricerca in $O(|Q_A||Q_B||\Sigma|)$.

c) $EQNFA$**.** Non si conosce algoritmo polinomiale; il problema è PSPACE-completo. Gli NFA mancano di una forma minima unica; la conversione a DFA può richiedere $2^{\Theta(n)}$ stati, facendo esplodere la complessità ([2]).

Esercizio 8. Un linguaggio A è *star-closed* se $A = A^*$. Sia $STAR\text{-}CLOSED_{DFA} = \{\langle M\rangle \mid M \text{ DFA tale che } L(M) \text{ è star-closed}\}$.

a) Caratterizzare quando $L(M)$ è star-closed in termini del DFA minimo.

b) Progettare un algoritmo polinomiale che lo decida.

c) Mostrare che $STAR\text{-}CLOSED_{DFA} \in P$.

Soluzione. **a) Caratterizzazione**. Sia M_{min} il DFA minimo di L. Il linguaggio è star-closed sse (i) q_0 è accettante e (ii) ogni stato accettante è *assorbente*, cioè da esso qualunque $w \in \Sigma^*$ porta ancora in uno stato accettante (equivalentemente, il sotto-automa indotto da F è un componente fortemente connesso senza uscite).

b) Algoritmo.

1) Minimizza M in $O(n \log n)$.

2) Controlla se $q_0 \in F$; se no, rifiuta (perché $\varepsilon \notin L$ e quindi $L \neq L^*$).

3) Esegui una BFS dagli stati in F seguendo tutte le transizioni: se si raggiunge uno stato non accettante, rifiuta.

4) Se tutti gli stati raggiungibili da F sono in F, accetta.

c) Complessità. Minimizzazione $O(n \log n)$, BFS $O(n + |\Sigma|n)$. Complessità polinomiale, dunque $STAR\text{-}CLOSED_{DFA} \in P$.

Esercizio 9. Sia $A \leq_p B$ la riducibilità polinomiale.

a) Provare che se $B \in P$ allora $A \in P$.

b) Dare un esempio dove $A \leq_m B$ ma non necessariamente $A \leq_p B$.

c) Discutere il ruolo di $\leq_p$ in NP e nell'ipotesi $P = NP$.

Soluzione. **a)** Sia f la riduzione polinomiale: $x \in A \iff f(x) \in B$. Poiché f si calcola in tempo n^k e la decisione di B richiede n^c, comporre le due procedure produce algoritmo $O(n^{\max\{k,c\}})$ per A, quindi $A \in P$.

b) Sia $A = HALT$ e $B = A_{TM}$. Vale $A \leq_m B$ (riduzione identità), ma non $A \leq_p B$ perché nessuna riduzione polinomiale può esistere fra due linguaggi non decidibili se A richiede l'informazione di fermarsi esattamente; la definizione stessa di $\leq_p$ esige che i linguaggi siano in Σ^*.

c) Le riduzioni polinomiali sono il «collante» della classe NP: un linguaggio è NP-completo se è in NP e ogni altro problema di NP si riduce ad esso in tempo polinomiale. Se esistesse un singolo NP-completo appartenente a P, avremmo $P = NP$.

References

[1] M. Sipser, *Introduction to the Theory of Computation*, 3rd ed., Cengage, 2012.

[2] J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed., Pearson, 2006.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 4th ed., MIT Press, 2022.

[4] M. Agrawal, N. Kayal, N. Saxena, "PRIMES is in P,*Ann. Math.*, 160(2):781-793, 2004.

[5] R. M. Karp, "Reducibility among Combinatorial Problems, in *Complexity of Computer Computations*, 1972.